GEMOT: TOWARDS MECHANISTIC WORLD MODELS FOR BIOLOGY

Anonymous authors
Paper under double-blind review

ABSTRACT

Scientific discovery seeks mechanisms that explain observations and predict beyond the measurements that produced them. Whereas large language models (LLMs) encode knowledge implicitly, mechanistic world models organise it as a parsimonious set of explicit, modular, reusable mechanisms whose predictions can be scored against data. In biology, where measurements are sparse and noisy, mechanistic world modelling must discover latent states and governing equations jointly, yet the prior knowledge that could constrain this search is largely unstructured. We introduce `gemot`, an agentic framework for mechanistic world modelling, and evaluate it on 18 published biological problems spanning molecular biology, epidemiology, and immune-cell differentiation, with up to 1,755 training measurements, 65 observables, 170 experimental conditions, and 3 data modalities per problem. `gemot` is auditable by construction: a semantic layer records each hypothesis, and a Model Context Protocol layer separates hypothesis formulation from numerical evaluation, so hypothesis scoring cannot be fabricated. On every problem, autonomously constructed models match or exceed the reference models in fit and parsimony, and generalise where held-outs exist. We demonstrate that `gemot` can formulate novel, biologically plausible mechanistic hypotheses. This work will enable decoding of interventional biological data into competing mechanistic hypotheses and design experiments that distinguish between them, and may serve as a blueprint for mechanistic world modelling in other domains.

1 INTRODUCTION

Mechanisms and models. A mechanism consists of entities structured by causal interactions, working in concert to generate a system's behaviour. A mechanistic model instantiates these mechanisms in simulatable form, enabling quantitative comparison against experimental and counterfactual predictions under interventions. This naturally connects to concepts developed in the philosophy of science such as *causal models* (Woodward, 2003; Pearl, 2009) and *real patterns* (Dennett, 1991; Ladyman, 2026). Among many formalisms for mechanistic models, Ordinary Differential Equations (ODE) models enjoy wide popularity across fields, and with novel symbolic regression (Cranmer, 2023; Mangan et al., 2016) and distillation (Cranmer et al., 2020) approaches it is now becoming possible to directly discover corresponding governing equations *de novo* from data.

Biological underdetermination. In biology, mechanistic models link hypotheses to measurements across scales, from molecular networks and interacting cells to tissues and populations. Models have helped relate local regulatory and cell–cell interactions to tissue organisation and generate testable predictions for developmental experiments (Tomlin & Axelrod, 2007). At the population scale, models of transmission and immunity connect individual processes to epidemic dynamics and inform disease control (Keeling & Rohani, 2008). Causal explanation, however, requires experimental and intervention evidence beyond reproducing an observed outcome (DiFrisco & Jaeger, 2020; Briscoe & Kicheva, 2017). In periodic patterning, for example, different molecular, cellular, and mechanical processes can yield similar patterns, so agreement with a reaction–diffusion model does not establish that mechanism (Hiscock & Megason, 2015). The same underdetermination holds at the level of the equations themselves, where distinct model topologies can be indistinguishable (Szederkényi et al., 2011). This makes *de novo* mechanism discovery in biology inherently prone to underspecification.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Triangulation and parsimony. The practical response of experimental biology is triangulation: combining complementary data modalities to constrain a shared mechanistic hypothesis (Ladyman, 2025, Section 4). ODE models recapitulate triangulation, as governing equations describe the temporal evolution of latent variables that are shared across experiments and contextualised by the interventions applied in each. Although triangulation narrows the space of admissible hypotheses, several hypotheses may survive, where the canonical statistical recourse is Occam's razor, favouring the simplest model consistent with the evidence (Forster & Sober, 1994), where the Akaike and Bayesian Information Criteria (AIC, BIC) are common complexity criteria. Crick (1988, p. 138), however, cautioned that evolution adds mechanisms on top of existing mechanisms, so whether arguments about simplicity are useful for establishing mechanistic hypothesis remains debated.

Mechanistic world modelling. Mechanistic world modelling, proposed by Posner, Lei, and Schölkopf (2026), formalises the process of mechanistic model formulation and casts it as knowledge organisation: mechanistic models explain a particular system via a structure of *bindings* that instantiate *mechanisms* over latent *variables*. Scientific discovery is then the joint recovery of these three elements, with parsimony and compositionality as inductive pressures, and where experimental verification and knowledge recombination implement active inquiry. As we outlined above, systems biology has already established counterparts to most of these components. State variables of ODE models correspond to latent variables, and the functional forms of interactions between them to mechanisms. Moreover, frameworks such as BioPAX and INDRA represent biological statements semantically and assemble them into executable models (Demir et al., 2010; Gyori et al., 2017). The binding structure can be encoded in machine-readable formats like SBML (Hucka et al., 2003) and human-readable front-ends such as Antimony (Smith et al., 2009), with PEtab (Schmiester et al., 2021) binding latent states to measurements collected under different interventions and assays, enabling the quantitative scoring of agreement of models with data. What systems biology lacks is the operationalisation of tacit judgement required to assemble these components: which latent states to posit, which mechanisms to bind to them, and what to simplify or omit. This is the variable, mechanism and structure discovery of a mechanistic world model. While some operationalisation has been attempted in textbooks (Klipp et al., 2016; Alon, 2019), most practitioners regard modelling as an art that draws on unstructured literature, expert intuition and tribal folklore.

Our Contribution. Here, we introduce `gemot`, an agentic system that operationalises the variable, mechanism and structure discovery of mechanistic world models, using large language models (LLMs) to turn biological knowledge and experimental data into scorable hypotheses. Agents jointly construct and revise biological mechanisms and readout mappings, choosing latent states and linking them to measurements, so that one hypothesis can be triangulated against many assays and modalities. An agentic judgement layer retrieves knowledge and assembles hypotheses into models and a model context protocol (MCP) layer insulates numerical scoring, ensuring auditability and preventing fabrication. `gemot` does not resolve underdetermination; it makes alternatives cheap, producing sets of admissible, auditable mechanistic hypotheses that a scientist can adjudicate and discriminate between with new experiments. We scale evaluation to 18 published biological problems from the PEtab Benchmark Models collection (Hass et al., 2019; Grein et al., 2026), comprising 6,638 training measurements across 11 measurement modalities. Agents receive experimental measurements, curated assay and experimental information and knowledge priors, while direct access to the published model and narrative study context is withheld.

Relation to prior work. Relative to semantic regularisation approaches (Kacprzyk & van der Schaar, 2025; Kacprzyk et al., 2025), our semantic representation describes the biology encoded in mechanistic hypothesis rather than qualitative behaviour of the dynamic systems, enabling validation of predictions on held-out interventional experimental data. Within LLM-based symbolic regression and model discovery (Shojaee et al., 2025a; Ziaei Bideh & Gryak, 2026; Song et al., 2026; Krongauz et al., 2026; Abhyankar et al., 2026), our focus is the joint inference of mechanism and measurement interpretation, which enables consideration of multi-modal data and triangulation based hypothesis filtering. Building on empirical discovery precedents (Holt et al., 2024), we scale from individual case studies to heterogeneous biological datasets requiring shared latent mechanisms to reconcile many observables, interventions and measurement modalities. Compared with synthetic equation- and model-discovery evaluations (d'Ascoli et al., 2024; Ziaei Bideh et al., 2026; Shojaee et al., 2025b; Murphy, 2026) and interactive-discovery benchmarks (Wiemann et al., 2026; Kabra et al., 2026; Gandhi et al., 2025; Zheng et al., 2026; Cerrato et al., 2026; Duan et al., 2025), we evaluate `gemot` on real data, enabling us to assess the relevance of prior knowledge.

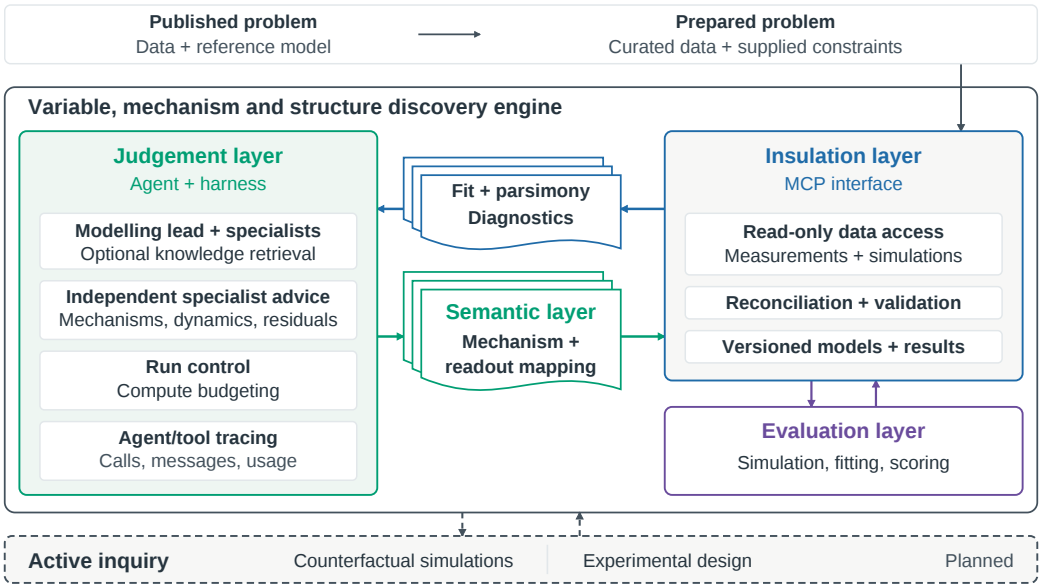

Figure 1: **The variable, mechanism and structure discovery engine.** Reference models and held-out data are withheld during model construction; feedback uses training data only. The MCP interface protects supplied constraints and preserves model versions and numerical results independently of agent traces. Dashed elements denote planned capabilities. Implementation details are in Appendices A–C.

2 Results

2.1 GEMOT: A layered architecture for trustworthy agentic discovery

As LLMs become more capable, auditability by humans is likely going to be one of the main measures of success for any agentic scientific discovery framework. We address this need architecturally (Figure 1) by separating the agentic *judgement layer*, which retrieves knowledge, analyses model fit, and proposes new hypotheses, from the numerical *evaluation layer*, which trains the model on experimental data, evaluates parsimony and compares simulations with data. The two layers are decoupled by a *semantic layer*, which instantiates hypotheses as human readable models, and an MCP *insulation layer*, which passes the formulated model from *judgement* to *evaluation* layer and makes data and model diagnostics query-able in the *judgement layer*. Together, these layers implement two separable design decisions: a trust boundary, across which the agent can submit models and query data and diagnostics but cannot alter data, fits or scores (insulation and evaluation layers); and an intermediate representation, in which each hypothesis is a versioned, human-readable model that can be inspected apart from the agent's narration of it (semantic layer). Hypotheses and corresponding model instantiations are constructed through iterative loops between *judgement* and *evaluation layers*, where model–data mismatch and other diagnostics inform modelling decisions while BIC serves as the quantitative objective that the LLM is instructed to optimise.

The ***judgement layer*** is an agent harness built with Deep Agents (LangChain, Inc., 2025) that supplies tools and computational budgets; a modelling lead coordinates mathematical, biological, and signal-analysis specialist subagents. Harness-level tracing records agent interactions and tool calls. Prompts specify a build–fit–diagnose loop with independent specialist advice (Listings 114–119), using benchmark-specific tasks, supplied constraints and entity lists (Appendix I.5). The optional biocurator subagent has access to UniProt (The UniProt Consortium, 2025), Reactome (Milacic et al., 2024), and OmniPath (Türei et al., 2026).

The ***evaluation layer*** is a functional layer controlled by the MCP server that implements model training through established systems biology tools: repeated local, gradient based optimisation from different initial parameter vectors (Grein et al., 2026). Simulation and gradient computation use

AMICI (Fröhlich et al., 2021), likelihood-based parameter estimation uses pyPESTO (Schälte et al., 2023) with trust-region optimisation in fides (Fröhlich & Sorger, 2022).

The ***semantic layer*** instantiates sets of hypotheses as human readable models in the Antimony (Smith et al., 2009) format that defines proposed states, reactions, rate laws, and readout mappings linking latent states to measured quantities (Appendix A.3).

The ***insulation layer*** is implemented as MCP server providing read-only access through DuckDB (Raasveldt & Mühleisen, 2019) to training measurements, simulations, raw and noise-normalised residuals, and noise scales by observable, condition, time, and model version. Separately from agent traces, the server preserves model versions, fitted parameters, diagnostics, and justifications. The layer also computes diagnostic information like residual bias, temporal structure, worst-fitting points, parameters at bounds, and local parameter uncertainties and correlations from the Fisher information matrix. Lastly the layer translates human readable models to SBML (Hucka et al., 2003), reconciles and validates them against PEtab's experimental and observation specification while protecting supplied constraints, and passes the full problem to the evaluation layer.

The insulation layer is also responsible for imposing deterministic safeguards. For example, despite explicit instructions, agents hardcoded numeric literals (unnamed fixed constants) in 12/90 GLM models across 11/18 benchmarks (Figure 2; Section 2.2), and in 0/5 Astra models with gemot versus 5/5 without (Figure 3; Section 2.4). Although constants can encode unit normalisation or cooperativity, unjustified fixed values can escape BIC penalties and understate uncertainty. During reconciliation, gemot therefore counts literals towards complexity penalties and reports them to the agent; automatic promotion to fitted parameters remains under consideration. This illustrates how deterministic safeguards support reliable model construction and reduce the need for human audit.

2.2 GEMOT MATCHES OR EXCEEDS PUBLISHED MODELS IN FIT AND PARSIMONY

We first tested whether gemot could construct parsimonious mechanistic models competitive with human-built references. We selected 18 problems from the PEtab Benchmark Models collection (Hass et al., 2019; Grein et al., 2026): 13 smaller problems, followed by five larger problems with more training measurements and generally more distinct interventions (conditions), observables (measured signals), and data modalities. The selection spans molecular biology, interacting cell populations, and epidemiology. The training datasets span 1–65 observables, 16–1,755 datapoints, 1–170 conditions, and 1–3 measurement modalities per problem (Figure 2, row 1; Table 1). Modalities comprise immunoblotting, mass spectrometry, qRT-PCR, ELISA, flow cytometry, HPLC, patch-clamp recordings, cell imaging, immunofluorescence microscopy, and viral RNA assays, alongside epidemiological surveillance. Agents receive measurements, experimental inputs, assay annotations, and curated biological constraints, but start from an empty model without the published mechanism or narrative study context (Appendix A.2).

We applied gemot using GLM-5.3-Flash as the underlying LLM, with 5 runs per problem (90 in total; model listings in Appendices G and H). Median usage per run was 2.0 million LLM tokens and 2.0 CPU-hours for smaller problems, versus 7.9 million tokens and 514 CPU-hours for larger problems (Appendix D.2), for a total of 400.3 million tokens and 17,230 CPU-hours. Overall, 78/90 models matched or improved on the reference by BIC (Schwarz, 1978), using the conventional $\Delta\mathrm{BIC} < 2$ threshold (candidate minus reference; (Kass & Raftery, 1995)). At least two runs achieved lower BIC than the reference in every problem, and lower AIC in every problem except Isensee (Figure 2, row 2/3). In 63/90 models (15/18 problems; exception Perelson, Bruno, and Blasi), runs scored $\Delta\mathrm{BIC} < -10$, conventionally indicating very strong evidence favouring the candidate (Kass & Raftery, 1995). 55/90 runs achieved a better fit to the data, indicated by a negative candidate-minus-reference training negative log-likelihood difference ($\Delta\mathrm{NLLH} < 0$) (Figure 2, row 4). In sum, gemot constructs models competitive with human references in fit and parsimony.

2.3 GEMOT PROPOSES NOVEL MECHANISTIC HYPOTHESIS

Published benchmarks may occur in LLM training data. We therefore compared candidate and reference structures to assess whether gemot proposes mechanisms beyond the published models.

Per-problem median differences ranged from -9 to $+3$ parameters (Figure 2, 5th row) and -4 to $+2$ states (Figure 2, 6th row) on the 13 smaller problems, versus -44 to $+1$ parameters and -19 to

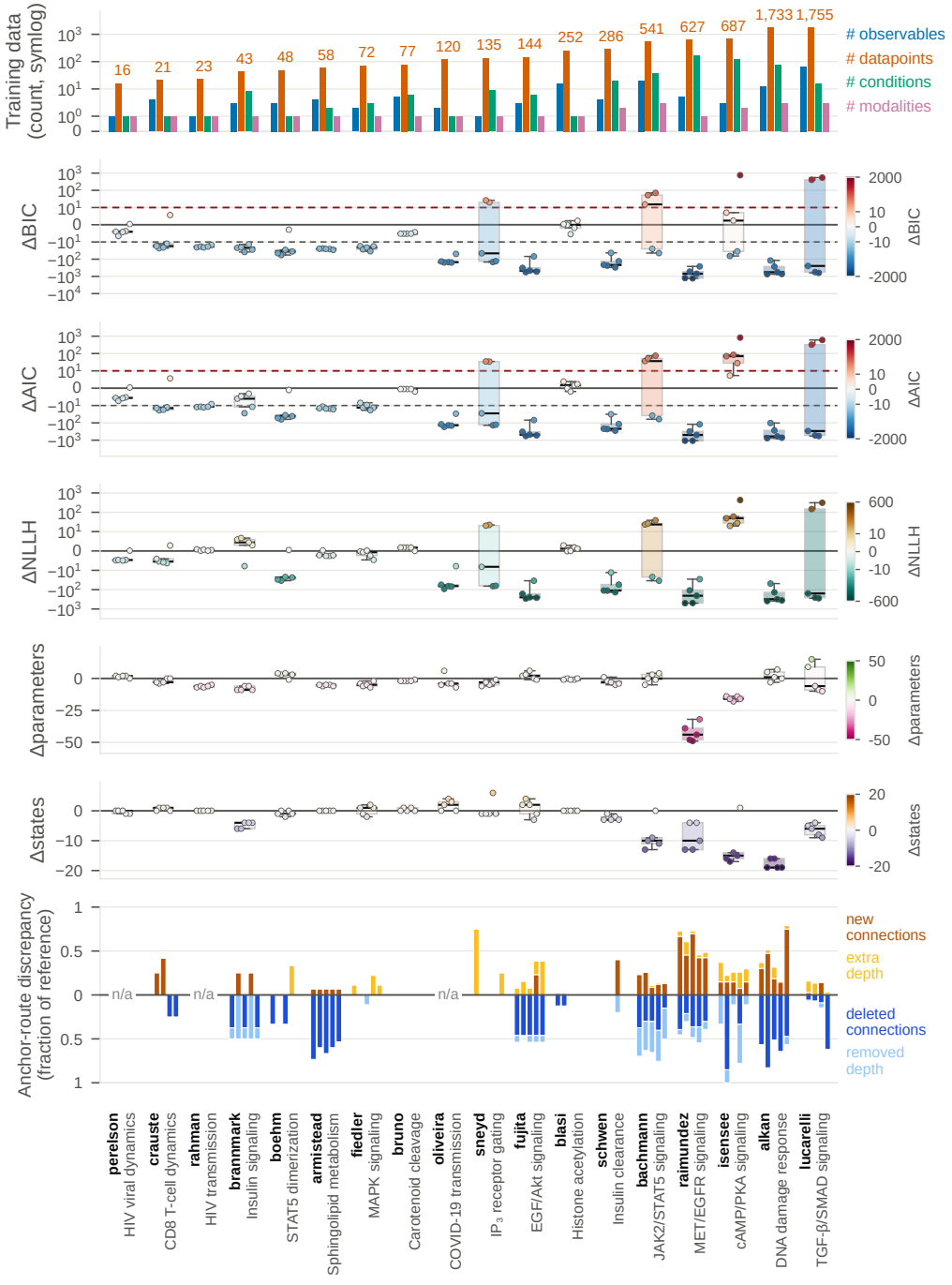

Figure 2: **Training performance, model size, and structural differences across 18 published problems.** Columns are ordered by training measurement count. Points show individual runs; boxplots summarise available values. ΔBIC and ΔAIC are candidate minus reference; ΔNLLH denotes the training negative-log-likelihood difference. Bottom-row stacked bars show each candidate's normalised anchor-route discrepancy: new connections (brown) and extra depth on shared routes (yellow) above zero; deleted connections (dark blue) and removed depth on shared routes (light blue) below. Normalisation uses the reference route summary, not the full network's edge count (Appendix D.3).

-6 states on the five larger problems, suggesting that parsimony gains are likely achieved through coarse-graining. To identify novel mechanisms, we computed model topology in terms of weighted graphs that quantify the latent depth—the shortest path along directly interacting state variables that are not part of the readout mapping— between interventions and readouts (anchors; Appendix D.3). Comparing these graphs to published models (SBGN Activity Flow maps in Appendix F), we found differences in 56/75 defined comparisons: 49 changed connectivity (brown/dark blue bars) and seven only route lengths (yellow/light blue bars; Figure 2, bottom row). All 25 larger-problem runs added connections, despite generally using fewer states. As extensive connection deletions and route shortening make their new connections and extra depth difficult to interpret, we concentrated on models with smaller anchor-route discrepancies:

Added connections. Two Crauste runs (Listings 35 and 36) added an early-effector-to-memory influence (brown) through pathogen clearance relieving suppression of memory differentiation, consistent with antigen-withdrawal experiments (Sarkar et al., 2008). One of these runs (Listing 36) also adds a direct differentiation step from naive to late effectors cells, for which we found neither strong supporting nor refuting evidence in the literature.

Latent intermediates. Longer routes (yellow) separated IP_3-dependent priming from channel opening in Sneyd (Listing 78), introduced an unnamed dynamic mediator of MAPK feedback in Fiedler (Listing 63), and inserted mTORC1–S6K signalling between Akt and S6 in Fujita (Listing 4), consistent with literature (Rossi et al., 2009; Dougherty et al., 2005; Inoki et al., 2002) (Appendix D.3).

We discuss more findings in Appendix D.3. In summary, `gemot` thus deviates substantially from published models, proposing both novel hypotheses that are consistent with, and, therefore, likely informed by, prior literature, as well as likely data-driven hypotheses with little literature support.

2.4 Held-out interventions partly discriminate mechanisms

To evaluate whether `gemot` generated mechanistic hypotheses generalise beyond training data, we evaluate predictive performance on the Fujita EGF–Akt–S6 benchmark (Fujita et al., 2010), where held-out data was available from the PEtab Benchmark Collection. Models trained on 144 constant-EGF-dose measurements predicted 276 held-out EGF-pulse and EGF-ramp measurements with all fitted parameters fixed (Figures 7–12). We report ΔNLL as candidate minus reference where negative values favour the candidate. To identify what drives performance, we evaluated `gemot` in six configurations, crossing two LLM drivers—O (open weight, blue) for GLM-5.3-Flash and F (frontier, brown) for GPT-6-Astra—with an ablation ladder (shading) over capability: 1, the full `gemot`; 2, the `gemot` without the biocurator; 3, a barebones harness with sandboxed file access and packages pre-installed (prompts in Listings 120–123); and 4 `gemot` with biology blinded and biocurator removed (prompt in Listing 124; Figure 3). We evaluated O1–O4 and F2–F3 with five runs each, labelled a–e (protocol and model listings in Appendices D.5 and G). F1 and F4 were cost cut.

Parsimony and prediction. We generally observed poor correlation between BIC and held-out performance (GLM: -0.1, Astra: 0.3 Spearman's ρ). For example, O1c achieved lower training ΔBIC than O1a (-567 versus -467), yet far worse held-out ΔNLL ($+44\,625$ versus -2974). The two best-performing models (F2c ΔNLL -3804; F3d ΔNLL -3740), recapitulated the forward activation structure of the reference model, but gained predictive power in face of added complexity such as EGFR internalisation (F2c, F3d), integral negative feedback from pAKT onto activation by pEGFR (F2c, F3d), and latent intermediates such as PI3K (F3d), MTOR (F2c, F3d) and S6K (F2c, F3d) (Figure 3B; Figure 29). This complexity is affordable in training BIC through lower NLL and coarse-graining elsewhere, e.g. ommiting EGFR/AKT and AKT/S6 complexes or merging dephosphorylation rate constants. This suggests that BIC, while likely helpful in guiding model construction, cannot discriminate mechanistic hypotheses, echoing Crick's caution about parsimony.

Implicit biological knowledge. To test whether model construction requires curated knowledge, we removed the biocurator and its external knowledge tools from the GLM panel (O2). All five models retained the reference's forward readout connections, and 3/5 models outperformed the reference on held-out data, compared with 4/5 for the full panel (Figure 3A, shaded region; C, O1/2). Thus, GLM could construct predictive mechanisms without curated retrieval, consistent with sufficient implicit biological knowledge in this LLM.

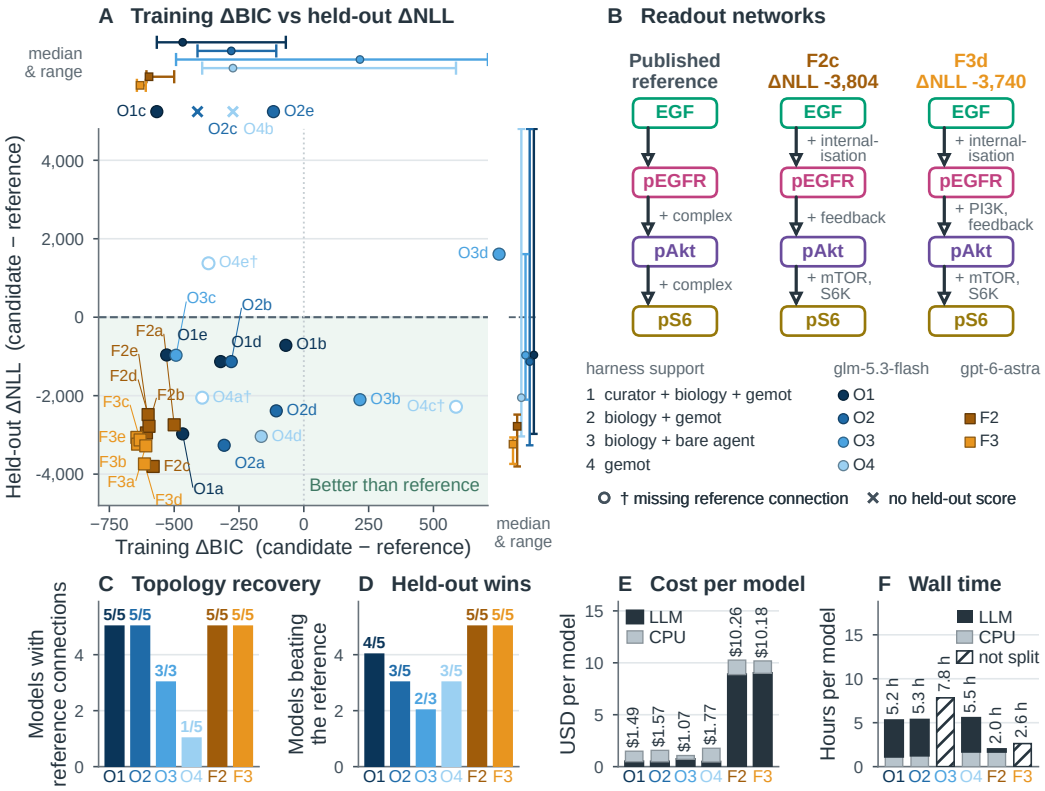

Figure 3: **Agent configurations, model structure, prediction, and cost. A,** The 28 Fujita runs that built a model (the two that built none are omitted), with linear training ΔBIC and held-out ΔNLL relative to the reference. Marginals show medians and ranges, a failed held-out integration ranking worst; off-scale values and missing scores are indicated separately; daggers in A mark models lacking a required reference connection. **B,** Readout networks of the reference and the two best held-out candidates: the EGF input and the three readouts, linked by the forward readout connections counted in C; side notes name what each model places on a connection (full maps in Figure 29). **C,** Recovery of unsigned reference connections among built models; Table 9 gives the models each configuration built. **D,** Candidates with held-out $\Delta\text{NLL} < 0$, by configuration, over every delivered model; a model whose held-out integration failed counts as not beating the reference. **E,** Cost per delivered model charges all launched runs: adjusted LLM usage plus a CPU list-price estimate. **F,** Wall time per model, split between LLM and numerical work where available; hatched bars show unsplit totals. Configurations, evaluation protocols, controls, full run mappings, and cost assumptions are in Appendix D.5.

Biological context. To test how important implicit biological knowledge is for generalisation, we blinded `gemot` by removing data annotation, removing the biocurator and obfuscating identifiers, including those of the stimulus protocol (O4). Blinding barely changed median training ΔBIC (-280 versus -273; Figure 3A, top errorbars) and reduced recovery of the forward readout connections from 5/5 to 1/5 (Figure 3C; Figure 3A, daggers mark runs that did not recover topology), but did not impair generalisation: median held-out ΔNLL improved from -1132 to -2052 (Figure 3A, right errorbars), and 3/5 blinded models beat the reference (Figure 3A, pale-blue circles). Biological context thus shaped which mechanisms were built rather than how well they predicted.

Cost and reliability. Practical model construction requires reliability at an affordable cost. Both biology-visible `gemot` GLM arms (O1/2) delivered 5/5 models, compared to 3/5 for the bare agent (O3), while Astra maintained 5/5 models for the barebones harness (F3). However, GLM delivered models in 5-8 hours at a cost of less than $2 per model versus more than $10 in 2-3 hours for Astra (Figure 3E,F). For larger models, where model training time exceeds Astra's 1h cache timeout, cost differences were more pronounced (data not shown). Astra nevertheless produced stronger

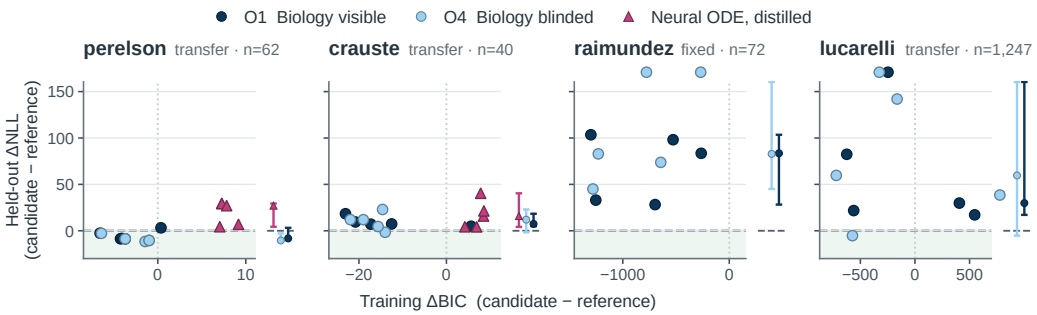

Figure 4: **Training fit against held-out prediction on four further systems.** Each panel plots training ΔBIC against held-out ΔNLL, both candidate minus reference, so the shaded region below zero is better than the reference on held-out data. Panels are ordered by training measurement count; n is the number of held-out measurements each ΔNLL is computed on. Crauste, Lucarelli, and Perelson are transfer tests, with candidate and reference parameters refitted per held-out unit; Raimundez holds dynamical parameters fixed while calibrating new per-blot observation scales. Circles are the Figure 2 runs (biology visible) and their biology-blinded counterparts; triangles are neural-ODE models distilled into rate laws (Crauste and Perelson only). Marginals show median and range per arm; runs beyond a panel's scale are drawn outside its frame, above it for held-out ΔNLL and past its right edge for training ΔBIC.

predictions more consistently: all ten candidates (F2/3) beat the reference, versus 9/13 biology-visible GLM (O1–3) candidates (Figure 3A, brown/orange squares versus blue circles). The best GLM model achieved 86% of the best Astra model's improvement over the reference (held-out ΔNLL -3265 versus -3804). Bare Astra also had lower median held-out ΔNLL than its panel (-3235 versus -2781). Thus, `gemot` supports reliable construction with cheaper LLMs, while the frontier model's more consistent predictions came at higher cost.

As the forward topology implemented by Fujita implements textbook biology, we conclude that biological context is necessary to build plausible models, and that ambiguity between competing mechanistic hypotheses persists even with additional experimental validation.

2.5 Robustness to biology blinding extends beyond Fujita

We tested whether Fujita's robustness to blinding extends to four further systems, comparing biology-visible (O1) and biology-blinded (O4) models (Figure 4). We reuse selected published validation tests for Raimundez and Crauste, but not Crauste's forecasting prediction tests. Perelson and Lucarelli use adapted held-out protocols (Raimúndez et al., 2020; Crauste et al., 2017; Perelson et al., 1996; Lucarelli et al., 2018). For Lucarelli, we refit all estimated parameters in both candidates and reference to pooled calibration and validation measurements; the original study kept gene-regulatory parameters fixed during transfer, which is not applicable to with blinded biology (Appendix D.6).

Blinding and neural baselines. Blinding did not consistently worsen held-out ΔNLL: O1 and O4 largely overlapped, although blinded runs had larger errors in some Raimundez and Lucarelli cases (dark and light blue circles; Appendix D.6). Distilled neural ODEs (Appendix D.7) performed comparably on Crauste and Perelson, with slightly higher median ΔBIC in both systems (magenta triangles). On Fujita, neural ODE integration produced unphysical simulations and failed numerically, preventing evaluation; training and distillation was not successful on Raimundez and Lucarelli.

Implications for model discovery. Competitive performance under blinding supports `gemot`'s ability to construct useful models with little to no cues for potential recalling of published models. The transfer examples assess model reuse across contexts rather than prediction under novel interventions, which may partly explain why these held-outs appear less discriminating than Fujita's fixed-parameter pulse/ramp evaluation. Biological context was therefore not needed for competitive generalisation on any system tested, pointing to a need for better experimental design.

3 DISCUSSION

Across 18 biological problems, `gemot` constructed models competitive with human references in fit and parsimony, and in doing so generated novel hypotheses. Variable, mechanism and structure discovery, the part of mechanistic world modelling that systems biology has left informal, can thus be operationalised with LLM agents. Across five benchmarks with held-out data, blinded runs generalised competitively; on Fujita, blinding changed biological plausibility but not how well models predicted. `gemot` thus leaves mechanistic underdetermination to experiment and human judgement, but makes alternative hypotheses explicit and cheap to generate.

Towards mechanistic world models. A mechanistic world model is maintained over successive experiments. After each, variables and bindings are revised, new mechanisms are added to a library, and the next intervention is chosen to discriminate between the remaining hypotheses (Posner et al., 2026). Our benchmarks provide fixed data and therefore only test the first step, which `gemot` implements as variable, mechanism and structure discovery. Furthermore, `gemot` draws mechanisms from the implicit knowledge of the LLM and does not represent them separately from their bindings, so they cannot enter a library. Structured prior knowledge could make these mechanisms explicit, as BioPAX (Demir et al., 2010) and INDRA (Gyori et al., 2017) already do for molecular systems. The models `gemot` constructs often also use coarser mechanisms than the references, which brings a general difficulty to light. Biology is multiscale, and while Posner et al. (2026) call for hierarchically organised mechanisms, they do not specify how mechanisms at different levels relate. INDRA partly addresses this (Bachman et al., 2023), but only with predefined levels for individual interactions, fixed in advance rather than discovered. Symbolic regression could complement it by discovering missing coarse-grained mechanisms from data (de Pomereu & Fröhlich, 2026). `gemot` also does not yet implement experimental design, which could be implemented by proposing interventions or measurements of latent states that maximally distinguish model candidates.

Trustworthiness by design. We expect future LLM generations to increase in capability and cost-effectiveness, rendering some of `gemot`'s components and observed capability augmentations obsolete. We therefore see `gemot`'s contribution mainly in its trustworthy architecture that make numerical claims easy to audit. Through the configured interface, the MCP layer prevents agents from replacing protected evaluation data or scores, and the judge uses these records to assess success (Appendix B.4). For mathematical modellers, this separation between judgement, semantics, and scoring enables empirical evaluation of modelling strategies.

Limits of published benchmarks. Evaluation on public data allows us to compare candidate models against a human reference, but withholding published reference models and validation data cannot exclude exposure during LLM training. As we only found current-generation LLMs capable enough (data not shown), training cutoff was not a viable mitigation strategy. Structural departures from the references demonstrate that `gemot` synthesises novel mechanistic hypotheses, which we, based on results for biology-blinded problems, expect to generalise to unseen problems (Section 2.3). The held-out evaluation is moreover unannounced to the LLM and supplies no feedback during construction. Even if a LLM could anticipate this evaluation and recall the held-out measurements, it would still have to translate them into an executable mechanism and readout mapping that can predict recalled measurements from training data. Verbatim reproductions of the reference model or recall of data cannot explain better performance on training data alone, and improvement through knowledge recombination would fulfill a stated design goal of mechanistic world models.

Complexity measures. Our study employed BIC as the only complexity criterion to guide model construction, with a parameter-count penalty that grows with sample size (Schwarz, 1978). `gemot` also supports AIC (Akaike, 1974), AICc (Hurvich & Tsai, 1989), CAIC (Bozdogan, 1987), HQC (Hannan & Quinn, 1979), and a Fisher-information approximation to TIC (Takeuchi, 1976) for online scoring (Appendix B.7). Our preliminary evaluation suggests that the choice among these criteria has no major impact on constructed models (data not shown). Predictive criteria such as WAIC (Watanabe, 2010) and Bayesian leave-one-out cross-validation (Vehtari et al., 2017) may better reflect generalisation, although their value for predicting new interventions remains to be established. Obtaining posterior samples for each candidate, with additional refitting where needed, would add substantial cost to our optimisation-based workflow and was beyond the scope of this evaluation. We anticipate that `gemot` will enable the development and empirical evaluation of better complexity measures in the future.

ETHICS STATEMENT

The authors confirm that this work aligns with the ICLR Code of Ethics. This study reanalyses published biological measurements and benchmark models and does not report new experiments involving human participants or animals. Its intended use is scientific hypothesis generation and comparison. Agent-generated mechanisms remain hypotheses: agreement with observations does not establish biological correctness, and pretrained knowledge may bias the mechanisms proposed. The evaluation interface and provenance records support scrutiny of numerical claims, while independent validation and appropriate human oversight remain necessary before using generated models to guide biological interventions or clinical decisions.

Many researchers in industry and academia build models as part of their work, whether for salaries or degrees. While we expect our work to contribute to some changes in these roles, we note that the agentic capabilities we use are already available in contemporary frontier models; we demonstrate rather than create them. Moreover, our results argue clearly for keeping a human in the loop to evaluate mechanistic hypotheses. We therefore acknowledge that our work may accelerate changes to these roles, but expect the net effect on affected researchers to be positive.

REPRODUCIBILITY STATEMENT

We do not include code with this submission, as we were unable to fully anonymise it and prepare it in a readable form in time. We are happy to share code with reviewers on request and will release all code upon publication. In its place, the Appendix provides a human-curated, LLM-narrated summary of the code together with the artefacts it generated. We document code and prompt versions throughout, but exact reproduction remains out of reach: the required compute and token budgets are high, and responses from external language models cannot be reproduced exactly. Because we observed large run-to-run variability, we performed five repeat runs for every configuration and refrained from interpreting small-to-medium performance or pass rate differences.

AI USE STATEMENT

Generative AI was used both in the evaluated system and in preparing this work.

Research ideation and execution. In the reported experiments, GLM-5.3-Flash and GPT-6-Astra agents developed mechanistic models, proposed and refined biological hypotheses, implemented these hypotheses as executable models, and interpreted fit diagnostics. Their configurations and evaluation protocols are described in Appendices C.6 and D.

During preparation of the research, OpenAI Codex and Anthropic Claude helped develop and refine conceptual frameworks, implement methods and analysis code, and interpret results. AI assistance also supported qualitative and thematic analysis and critique of research methods and experiments, including ablations, statistical tests, and held-out evaluation protocols. Anthropic Claude was used for code writing, spot-checks of agent traces, and diagnosing failed runs from them, and was the sole AI assistant used for data curation: extracting measurements, cleaning and reformatting datasets, and checking units and metadata. Claude also implemented, ran, and checked analysis and figure code on the computing cluster, including the held-out and transfer rescoring and the refits of the rebuilt distilled models, and generated the manuscript's reported values from the frozen inputs as LaTeX macros checked against the data. Codex and Claude assisted with preparing scientific figures and research artifacts.

Retrieval and discovery. Codex and Claude assisted with searching for, identifying, and analysing relevant literature.

Drafting. Codex helped draft sections of the paper. Claude drafted figure captions and appendix passages describing its analyses.

Writing and polishing. Codex helped edit and polish the manuscript for clarity, concision, and flow. Claude audited the manuscript for outdated values, statements the data no longer supported, and inconsistent notation, and applied the corrections the authors directed.

AI was not used for benchmark selection, synthetic dataset generation, formulation or proof of mathematical claims, or translation.

AI assistance identified several issues in human-curated benchmarks. Code was checked by at least one author, and held-out data curation was checked by two authors. Numerical claims are supported by archived computational outputs and statistical analyses. The human authors directed the work and retain responsibility for its scientific claims, references, code, and final text, including all AI-assisted contributions.

REFERENCES

Nikhil Abhyankar, Sha Li, Sanchit Kabra, Naren Ramakrishnan, Yulia Gel, and Chandan K. Reddy. LLM-ACES: Closed-loop discovery of dynamical systems with LLM-guided adaptive search. *arXiv preprint arXiv:2606.25039*, 2026. URL https://arxiv.org/abs/2606.25039.

Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974. doi: 10.1109/TAC.1974.1100705. URL https://doi.org/10.1109/TAC.1974.1100705.

Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman and Hall/CRC, Boca Raton, 2nd edition, 2019. ISBN 9781439837177. doi: 10.1201/9780429283321. URL https://doi.org/10.1201/9780429283321.

Joy Armistead, Sebastian Höpfl, Pierre Goldhausen, Andrea Müller-Hartmann, Evelin Fahle, Julia Hatzold, Rainer Franzen, Susanne Brodesser, Nicole E. Radde, and Matthias Hammerschmidt. A sphingolipid rheostat controls apoptosis versus apical cell extrusion as alternative tumour-suppressive mechanisms. *Cell Death & Disease*, 15:746, 2024. doi: 10.1038/s41419-024-07134-2. URL https://doi.org/10.1038/s41419-024-07134-2.

John A. Bachman, Benjamin M. Gyori, and Peter K. Sorger. Automated assembly of molecular mechanisms at scale from text mining and curated databases. *Molecular Systems Biology*, 19(5):e11325, 2023. doi: 10.15252/msb.202211325. URL https://link.springer.com/article/10.15252/msb.202211325.

Vitaly Balan, Deborah T. Leicht, Jun Zhu, Karina Balan, Alexander Kaplun, Vinita Singh-Gupta, Jun Qin, Hong Ruan, Michael J. Comb, and Guri Tzivion. Identification of novel in vivo Raf-1 phosphorylation sites mediating positive feedback Raf-1 regulation by extracellular signal-regulated kinase. *Molecular Biology of the Cell*, 17(3):1141–1153, 2006. doi: 10.1091/mbc.E04-12-1123. URL https://doi.org/10.1091/mbc.E04-12-1123.

Hamparsum Bozdogan. Model selection and Akaike's information criterion (AIC): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, 1987. doi: 10.1007/BF02294361. URL https://doi.org/10.1007/BF02294361.

Cecilia Brännmark, Robert Palmér, S. Torkel Glad, Gunnar Cedersund, and Peter Strålfors. Mass and information feedbacks through receptor endocytosis govern insulin signaling as revealed using a parameter-free modeling framework. *Journal of Biological Chemistry*, 285(26):20171–20179, 2010. doi: 10.1074/jbc.M110.106849. URL https://doi.org/10.1074/jbc.M110.106849.

James Briscoe and Anna Kicheva. The physics of development 100 years after D'Arcy Thompson's "On Growth and Form". *Mechanisms of Development*, 145:26–31, 2017. doi: 10.1016/j.mod.2017.03.005. URL https://doi.org/10.1016/j.mod.2017.03.005.

Eric J. Brown, Peter A. Beal, Curtis T. Keith, Jie Chen, Tae Bum Shin, and Stuart L. Schreiber. Control of p70 S6 kinase by kinase activity of FRAP in vivo. *Nature*, 377(6548):441–446, 1995. doi: 10.1038/377441a0. URL https://doi.org/10.1038/377441a0.

Mattia Cerrato, Lennart Baur, Jannis Brugger, Sajjad Shumaly, Nicholas Schmitt, Edward Finkelstein, Selina Jukic, Lars Münzel, Felix Peter Paul, Pascal Pfannes, Benedikt Rohr, Julius Schellenberg, Philipp Wolf, and Stefan Kramer. Science-Gym: a simple testbed for AI-driven scientific discovery. *Machine Learning*, 115:16, 2026. doi: 10.1007/s10994-025-06914-x. URL https://doi.org/10.1007/s10994-025-06914-x.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. URL https://arxiv.org/abs/1806.07366.

Miles Cranmer. Interpretable machine learning for science with PySR and SymbolicRegression.jl. *arXiv preprint arXiv:2305.01582*, 2023. URL https://arxiv.org/abs/2305.01582.

Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. In *Advances in Neural Information Processing Systems*, volume 33, pp. 17429–17442. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/c9f2f917078bd2db12f23c3b413d9cba-Paper.pdf.

Fabien Crauste, Julien Mafille, Lilia Boucinha, Sophia Djebali, Olivier Gandrillon, Jacqueline Marvel, and Christophe Arpin. Identification of nascent memory CD8 t cells and modeling of their ontogeny. *Cell Systems*, 4(3):306–317.e4, 2017. doi: 10.1016/j.cels.2017.01.014. URL https://doi.org/10.1016/j.cels.2017.01.014.

Francis Crick. *What Mad Pursuit: A Personal View of Scientific Discovery*. Basic Books, New York, 1988. ISBN 9780465091379. URL https://books.google.com/books?id=NBAhAQAAIAAJ.

Stéphane d'Ascoli, Sören Becker, Philippe Schwaller, Alexander Mathis, and Niki Kilbertus. ODEFormer: Symbolic regression of dynamical systems with transformers. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/5ed5c3c846f684a54975ad7a2525199f-Abstract-Conference.html.

Theodore de Pomereu and Fabian Fröhlich. Symbolic regression enables coarse-grained model discovery of intracellular signalling dynamics. *bioRxiv*, 2026. doi: 10.64898/2026.08.20.745973. URL https://doi.org/10.64898/2026.08.20.745973.

Emek Demir, Michael P. Cary, Suzanne Paley, Ken Fukuda, et al. The BioPAX community standard for pathway data sharing. *Nature Biotechnology*, 28(9):935–942, 2010. doi: 10.1038/nbt.1666. URL https://www.nature.com/articles/nbt.1666.

Daniel C. Dennett. Real patterns. *The Journal of Philosophy*, 88(1):27–51, 1991. doi: 10.2307/2027085. URL https://doi.org/10.2307/2027085.

James DiFrisco and Johannes Jaeger. Genetic causation in complex regulatory systems: An integrative dynamic perspective. *BioEssays*, 42(6):1900226, 2020. doi: 10.1002/bies.201900226. URL https://doi.org/10.1002/bies.201900226.

Michele K. Dougherty, Jürgen Müller, Daniel A. Ritt, Ming Zhou, Xiao Zhen Zhou, Terry D. Copeland, Thomas P. Conrads, Timothy D. Veenstra, Kun Ping Lu, and Deborah K. Morrison. Regulation of Raf-1 by direct feedback phosphorylation. *Molecular Cell*, 17(2):215–224, 2005. doi: 10.1016/j.molcel.2004.11.055. URL https://doi.org/10.1016/j.molcel.2004.11.055.

Haonan Duan, Stephen Zhewen Lu, Caitlin Fiona Harrigan, Nishkrit Desai, Jiarui Lu, Michał Koziarski, Leonardo Cotta, and Chris J. Maddison. Measuring scientific capabilities of language models with a systems biology dry lab. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3733. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a23760ba51036d530a2656c3835f826c-Abstract-Datasets_and_Benchmarks_Track.html.

Anna Fiedler, Sebastian Raeth, Fabian J. Theis, Angelika Hausser, and Jan Hasenauer. Tailored parameter optimization methods for ordinary differential equation models with steady-state constraints. *BMC Systems Biology*, 10:80, 2016. doi: 10.1186/s12918-016-0319-7. URL https://doi.org/10.1186/s12918-016-0319-7.

Malcolm Forster and Elliott Sober. How to tell when simpler, more unified, or less ad hoc theories will provide more accurate predictions. *The British Journal for the Philosophy of Science*, 45(1):1–35, 1994. doi: 10.1093/bjps/45.1.1. URL https://doi.org/10.1093/bjps/45.1.1.

Fabian Fröhlich and Peter K. Sorger. Fides: Reliable trust-region optimization for parameter estimation of ordinary differential equation models. *PLoS Computational Biology*, 18(7):e1010322, 2022. doi: 10.1371/journal.pcbi.1010322.

Fabian Fröhlich, Daniel Weindl, Yannik Schälte, Dilan Pathirana, Łukasz Paszkowski, Glenn Terje Lines, Paul Stapor, and Jan Hasenauer. AMICI: high-performance sensitivity analysis for large ordinary differential equation models. *Bioinformatics*, 37(20):3676–3677, 2021. doi: 10.1093/bioinformatics/btab227.

Kazuhiro A. Fujita, Yu Toyoshima, Shinsuke Uda, Yu-ichi Ozaki, Hiroyuki Kubota, and Shinya Kuroda. Decoupling of receptor and downstream signals in the Akt pathway by its low-pass filter characteristics. *Science Signaling*, 3(132):ra56, 2010. doi: 10.1126/scisignal.2000810. URL https://doi.org/10.1126/scisignal.2000810.

Kanishk Gandhi, Michael Y. Li, Lyle Goodyear, Agam Bhatia, Louise Li, Aditi Bhaskar, Mohammed Zaman, and Noah D. Goodman. BoxingGym: Benchmarking progress in automated experimental design and model discovery. *arXiv preprint arXiv:2501.01540*, 2025. doi: 10.48550/arXiv.2501.01540. URL https://arxiv.org/abs/2501.01540.

Stephan Grein, David R. Penas, Daniel Weindl, Polina Lakrisenko, Julio R. Banga, and Jan Hasenauer. Large-scale analysis of optimisation methods for parameter estimation problems in the life sciences. *bioRxiv*, 2026. doi: 10.64898/2026.07.11.737731.

Benjamin M. Gyori, John A. Bachman, Kartik Subramanian, Jeremy L. Muhlich, Lucian Galescu, and Peter K. Sorger. From word models to executable models of signaling networks using automated assembly. *Molecular Systems Biology*, 13(11):954, 2017. doi: 10.15252/msb.20177651. URL https://link.springer.com/article/10.15252/msb.20177651.

E. J. Hannan and B. G. Quinn. The determination of the order of an autoregression. *Journal of the Royal Statistical Society: Series B (Methodological)*, 41(2):190–195, 1979. doi: 10.1111/j.2517-6161.1979.tb01072.x. URL https://doi.org/10.1111/j.2517-6161.1979.tb01072.x.

Leonard A. Harris, Justin S. Hogg, José-Juan Tapia, John A. P. Sekar, Sanjana A. Gupta, Ilya Korsunsky, Arshi Arora, Dipak Barua, Robert P. Sheehan, and James R. Faeder. BioNetGen 2.2: advances in rule-based modeling. *Bioinformatics*, 32(21):3366–3368, 2016. doi: 10.1093/bioinformatics/btw469. URL https://doi.org/10.1093/bioinformatics/btw469.

Helge Hass, Carolin Loos, Elba Raimúndez-Álvarez, Jens Timmer, Jan Hasenauer, and Clemens Kreutz. Benchmark problems for dynamic modeling of intracellular processes. *Bioinformatics*, 35(17):3073–3082, 2019. doi: 10.1093/bioinformatics/btz020. URL https://doi.org/10.1093/bioinformatics/btz020.

Tom W. Hiscock and Sean G. Megason. Mathematically guided approaches to distinguish models of periodic patterning. *Development*, 142(3):409–419, 2015. doi: 10.1242/dev.107441. URL https://doi.org/10.1242/dev.107441.

Samuel Holt, Zhaozhi Qian, Tennison Liu, James Weatherall, and Mihaela van der Schaar. Data-driven discovery of dynamical systems in pharmacology using large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/aea8bdc42d8ba3a67a69b3f18be93f69-Abstract-Conference.html.

Michael Hucka, Andrew Finney, Herbert M. Sauro, Hamid Bolouri, John C. Doyle, Hiroaki Kitano, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003. doi: 10.1093/bioinformatics/btg015. URL https://doi.org/10.1093/bioinformatics/btg015.

Clifford M. Hurvich and Chih-Ling Tsai. Regression and time series model selection in small samples. *Biometrika*, 76(2):297–307, 1989. doi: 10.1093/biomet/76.2.297. URL https://doi.org/10.1093/biomet/76.2.297.

Ken Inoki, Yong Li, Tianquan Zhu, Jun Wu, and Kun-Liang Guan. TSC2 is phosphorylated and inhibited by Akt and suppresses mTOR signalling. *Nature Cell Biology*, 4(9):648–657, 2002. doi: 10.1038/ncb839. URL https://doi.org/10.1038/ncb839.

Sanchit Kabra, Nikhil Abhyankar, Saaketh Desai, Prasad Iyer, and Chandan K. Reddy. LLM-AutoSciLab: Closed-loop scientific discovery via active experimentation with LLMs. *arXiv preprint arXiv:2605.24043*, 2026. doi: 10.48550/arXiv.2605.24043. URL https://arxiv.org/abs/2605.24043.

Krzysztof Kacprzyk and Mihaela van der Schaar. No equations needed: Learning system dynamics without relying on closed-form ODEs. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kbm6tsICar.

Krzysztof Kacprzyk, Julianna Piskorz, and Mihaela van der Schaar. Skip the equations: Learning behavior of personalized dynamical systems directly from data. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 28619–28643. PMLR, 2025. URL https://proceedings.mlr.press/v267/kacprzyk25a.html.

Robert E. Kass and Adrian E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995. doi: 10.1080/01621459.1995.10476572. URL https://doi.org/10.1080/01621459.1995.10476572.

Matt J. Keeling and Pejman Rohani. *Modeling Infectious Diseases in Humans and Animals*. Princeton University Press, 2008. doi: 10.1515/9781400841035. URL https://doi.org/10.1515/9781400841035.

Edda Klipp, Wolfram Liebermeister, Christoph Wierling, and Axel Kowald. *Systems Biology: A Textbook*. Wiley-VCH, Weinheim, 2nd edition, 2016. ISBN 9783527336364. URL https://www.wiley-vch.de/en/areas-interest/natural-sciences/life-sciences-11ls/bioinformatics-computational-biology-11lsg/systems-biology-978-3-527-33636-4.

David Krongauz, Arad Zulti, Eran Segal, and Teddy Lazebnik. Automatic ordinary differential equations discovery for biological systems using large language model powered agentic system. *arXiv preprint arXiv:2607.13608*, 2026. URL https://arxiv.org/abs/2607.13608.

James Ladyman. Scientific empiricism and modal structure. *Synthese*, 206:245, 2025. doi: 10.1007/s11229-025-05323-w. URL https://link.springer.com/article/10.1007/s11229-025-05323-w.

James Ladyman. Patterns all the way up: Prolegomena to a future naturalized metaphysics. In Tyler Millhouse, Stephen D. Petersen, and Don Ross (eds.), *Dennett's Real Patterns in Science and Nature*, pp. 59–76. MIT Press, 2026. doi: 10.7551/mitpress/15550.003.0006. URL https://doi.org/10.7551/mitpress/15550.003.0006.

LangChain, Inc. Deep Agents. Open-source software, 2025. URL https://github.com/langchain-ai/deepagents. Accessed 22 September 2026.

Carolin Loos, Sabrina Krause, and Jan Hasenauer. Hierarchical optimization for the efficient parametrization of ODE models. *Bioinformatics*, 34(24):4266–4273, 2018. doi: 10.1093/bioinformatics/bty514. URL https://doi.org/10.1093/bioinformatics/bty514.

Carlos F. Lopez, Jeremy L. Muhlich, John A. Bachman, and Peter K. Sorger. Programming biological models in Python using PySB. *Molecular Systems Biology*, 9:646, 2013. doi: 10.1038/msb.2013.1. URL https://doi.org/10.1038/msb.2013.1.

Philippe Lucarelli, Marcel Schilling, Clemens Kreutz, Artyom Vlasov, Martin E. Boehm, Nao Iwamoto, Bernhard Steiert, Susen Lattermann, Marvin Wäsch, Markus Stepath, Matthias S. Matter, Mathias Heikenwälder, Katrin Hoffmann, Daniela Deharde, Georg Damm, Daniel Seehofer, Maria Muciek, Norbert Gretz, Wolf D. Lehmann, Jens Timmer, and Ursula Klingmüller. Resolving the combinatorial complexity of Smad protein complex formation and its link to gene expression. *Cell Systems*, 6(1):75–89.e11, 2018. doi: 10.1016/j.cels.2017.11.010. URL https://doi.org/10.1016/j.cels.2017.11.010.

Don-On Daniel Mak, John E. Pearson, King Pan Campion Loong, Suman Datta, Marisabel Fernández-Mongil, and J. Kevin Foskett. Rapid ligand-regulated gating kinetics of single inositol 1,4,5-trisphosphate receptor Ca2+ release channels. *EMBO Reports*, 8(11):1044–1051, 2007. doi: 10.1038/sj.embor.7401087. URL https://doi.org/10.1038/sj.embor.7401087.

Niall M. Mangan, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 2(1):52–63, 2016. doi: 10.1109/TMBMC.2016.2633265. URL https://doi.org/10.1109/TMBMC.2016.2633265.

Huaiyu Mi, Falk Schreiber, Stuart Moodie, Tobias Czauderna, Emek Demir, Robin Haw, Augustin Luna, Nicolas Le Novère, Anatoly Sorokin, and Alice Villéger. Systems biology graphical notation: Activity flow language level 1 version 1.2. *Journal of Integrative Bioinformatics*, 12(2):265, 2015. doi: 10.2390/biecoll-jib-2015-265. URL https://doi.org/10.2390/biecoll-jib-2015-265.

Marija Milacic, Deidre Beavers, Patrick Conley, et al. The Reactome pathway knowledgebase 2024. *Nucleic Acids Research*, 52(D1):D672–D678, 2024. doi: 10.1093/nar/gkad1025. URL https://academic.oup.com/nar/article/52/D1/D672/7369850.

Kevin Murphy. Model discovery agent: LLM-assisted Bayesian experiment design for data-efficient discovery of mechanistic world models. *arXiv preprint arXiv:2608.09696*, 2026. URL https://arxiv.org/abs/2608.09696v4.

Dilan Pathirana, Fabian Fröhlich, Sebastian Persson, Frank T. Bergmann, Matthias König, Paul F. Lang, Severin Bang, Svenja Kemmer, Jan Hasenauer, and Daniel Weindl. The parameter estimation tables (PEtab) format version 2. *Journal of Integrative Bioinformatics*, 2026. doi: 10.1515/jib-2026-0004. URL https://doi.org/10.1515/jib-2026-0004.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009. doi: 10.1017/CBO9780511803161. URL https://doi.org/10.1017/CBO9780511803161.

Alan S. Perelson, Avidan U. Neumann, Martin Markowitz, John M. Leonard, and David D. Ho. HIV-1 dynamics in vivo: virion clearance rate, infected cell life-span, and viral generation time. *Science*, 271(5255):1582–1586, 1996. doi: 10.1126/science.271.5255.1582. URL https://doi.org/10.1126/science.271.5255.1582.

Sebastian Persson, Branwen Snelling, Maren Philipps, Daniel Weindl, Marija Cvijovic, Jan Hasenauer, Dilan Pathirana, and Fabian Fröhlich. PEtab SciML: an exchange format for specifying and training dynamic scientific machine learning models. *arXiv preprint arXiv:2608.20184*, 2026. URL https://arxiv.org/abs/2608.20184.

Ingmar Posner, Anson Lei, and Bernhard Schölkopf. From observation to insight: Mechanistic world models and the quest for autonomous discovery. *arXiv preprint arXiv:2607.12474*, 2026. doi: 10.48550/arXiv.2607.12474. URL https://arxiv.org/abs/2607.12474v2.

Mark Raasveldt and Hannes Mühleisen. DuckDB: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pp. 1981–1984. Association for Computing Machinery, 2019. doi: 10.1145/3299869.3320212.

Elba Raimúndez, Simone Keller, Gwen Zwingenberger, Karolin Ebert, Sabine Hug, Fabian J. Theis, Dieter Maier, Birgit Luber, and Jan Hasenauer. Model-based analysis of response and resistance factors of cetuximab treatment in gastric cancer cell lines. *PLoS Computational Biology*, 16(3): e1007147, 2020. doi: 10.1371/journal.pcbi.1007147. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007147.

Ana M. Rossi, Andrew M. Riley, Stephen C. Tovey, Taufiq Rahman, Olivier Dellis, Emily J. A. Taylor, Valery G. Veresov, Barry V. L. Potter, and Colin W. Taylor. Synthetic partial agonists reveal key steps in IP3 receptor activation. *Nature Chemical Biology*, 5(9):631–639, 2009. doi: 10.1038/nchembio.195. URL https://doi.org/10.1038/nchembio.195.

Surojit Sarkar, Vandana Kalia, W. Nicholas Haining, Bogumila T. Konieczny, Shruti Subramaniam, and Rafi Ahmed. Functional and genomic profiling of effector CD8 T cell subsets with distinct memory fates. *Journal of Experimental Medicine*, 205(3):625–640, 2008. doi: 10.1084/jem.20071641. URL https://doi.org/10.1084/jem.20071641.

Yannik Schälte, Fabian Fröhlich, Paul J. Jost, Jakob Vanhoefer, Dilan Pathirana, Paul Stapor, Polina Lakrisenko, Dantong Wang, Elba Raimúndez, Simon Merkt, Leonard Schmiester, Philipp Städter, Stephan Grein, Erika Dudkin, Domagoj Doresic, Daniel Weindl, and Jan Hasenauer. pyPESTO: a modular and scalable tool for parameter estimation for dynamic models. *Bioinformatics*, 39(11): btad711, 2023. doi: 10.1093/bioinformatics/btad711.

Leonard Schmiester, Yannik Schälte, Fabian Fröhlich, Jan Hasenauer, and Daniel Weindl. Efficient parameterization of large-scale dynamic models based on relative measurements. *Bioinformatics*, 36(2):594–602, 2020. doi: 10.1093/bioinformatics/btz581. URL https://doi.org/10.1093/bioinformatics/btz581.

Leonard Schmiester, Yannik Schälte, Frank T. Bergmann, Tacio Camba, Erika Dudkin, Janine Egert, Fabian Fröhlich, Lara Fuhrmann, Adrian L. Hauber, Svenja Kemmer, Polina Lakrisenko, Carolin Loos, Simon Merkt, Wolfgang Müller, Dilan Pathirana, Elba Raimúndez, Lukas Refisch, Marcus Rosenblatt, Paul L. Stapor, Philipp Städter, Dantong Wang, Franz-Georg Wieland, Julio R. Banga, Jens Timmer, Alejandro F. Villaverde, Sven Sahle, Clemens Kreutz, Jan Hasenauer, and Daniel Weindl. PEtab—interoperable specification of parameter estimation problems in systems biology. *PLoS Computational Biology*, 17(1):e1008646, 2021. doi: 10.1371/journal.pcbi.1008646. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008646.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. doi: 10.1214/aos/1176344136. URL https://doi.org/10.1214/aos/1176344136.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, pp. 16054–16085, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/28df8e730c054c5331855fd4d5403ba9-Abstract-Conference.html.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K. Reddy. LLM-SRBench: A new benchmark for scientific equation discovery with large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 55325–55359. PMLR, 2025b. URL https://proceedings.mlr.press/v267/shojaee25a.html.

Lucian P. Smith, Frank T. Bergmann, Deepak Chandran, and Herbert M. Sauro. Antimony: a modular model definition language. *Bioinformatics*, 25(18):2452–2454, 2009. doi: 10.1093/bioinformatics/btp401. URL https://doi.org/10.1093/bioinformatics/btp401.

Lucian P. Smith, Michael Hucka, Stefan Hoops, Andrew Finney, Martin Ginkel, Chris J. Myers, Ion Moraru, and Wolfram Liebermeister. SBML level 3 package: Hierarchical model composition, version 1 release 3. *Journal of Integrative Bioinformatics*, 12(2):268, 2015. doi: 10.2390/biecoll-jib-2015-268. URL https://doi.org/10.2390/biecoll-jib-2015-268.

James Sneyd and Jean-François Dufour. A dynamic model of the type-2 inositol trisphosphate receptor. *Proceedings of the National Academy of Sciences*, 99(4):2398–2403, 2002. doi: 10.1073/pnas.032281999. URL https://doi.org/10.1073/pnas.032281999.

Sum Kyun Song, Bong Gyun Shin, and Jae Yong Lee. Discovering ordinary differential equations with LLM-based qualitative and quantitative evaluation. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2605.07323.

Gábor Szederkényi, Julio R. Banga, and Antonio A. Alonso. Inference of complex biological networks: distinguishability issues and optimization-based solutions. *BMC Systems Biology*, 5:177, 2011. doi: 10.1186/1752-0509-5-177. URL https://doi.org/10.1186/1752-0509-5-177.

Kei Takeuchi. Distribution of information statistics and criteria for adequacy of models. *Suri Kagaku*, 14(3):12–18, 1976. URL https://ndlsearch.ndl.go.jp/en/books/R000000004-I1701125. In Japanese.

Beth A. Tamburini, Matthew A. Burchill, and Ross M. Kedl. Antigen capture and archiving by lymphatic endothelial cells following vaccination or viral infection. *Nature Communications*, 5: 3989, 2014. doi: 10.1038/ncomms4989. URL https://doi.org/10.1038/ncomms4989.

The UniProt Consortium. UniProt: the universal protein knowledgebase in 2025. *Nucleic Acids Research*, 53(D1):D609–D617, 2025. doi: 10.1093/nar/gkae1010. URL https://academic.oup.com/nar/article/53/D1/D609/7902999.

Claire J. Tomlin and Jeffrey D. Axelrod. Biology by numbers: mathematical modelling in developmental biology. *Nature Reviews Genetics*, 8(5):331–340, 2007. doi: 10.1038/nrg2098. URL https://doi.org/10.1038/nrg2098.

Dénes Türei, Jonathan Schaul, Nicolàs Palacio-Escat, et al. OmniPath: integrated knowledgebase for multi-omics analysis. *Nucleic Acids Research*, 54(D1):D652–D660, 2026. doi: 10.1093/nar/gkaf1126. URL https://academic.oup.com/nar/article/54/D1/D652/8326458.

Aki Vehtari, Andrew Gelman, and Jonah Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5):1413–1432, 2017. doi: 10.1007/s11222-016-9696-4. URL https://doi.org/10.1007/s11222-016-9696-4.

Sumio Watanabe. Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11:3571–3594, 2010. URL https://www.jmlr.org/papers/v11/watanabe10a.html.

Matt L. Wiemann, Lindsay M. Smith, Peter Melchior, Siddharth Mishra-Sharma, Andrew Gordon Wilson, Pavel Izmailov, and Carolina Cuesta-Lázaro. DiscoverPhysics: Benchmarking LLMs for out-of-the-box scientific thinking. *arXiv preprint arXiv:2605.26087*, 2026. doi: 10.48550/arXiv.2605.26087. URL https://arxiv.org/abs/2605.26087.

James Woodward. *Making Things Happen: A Theory of Causal Explanation*. Oxford University Press, 2003. doi: 10.1093/0195155270.001.0001. URL https://academic.oup.com/book/4324.

Tianshi Zheng, Kelvin Kiu-Wai Tam, Newt Hue-Nam K. Nguyen, Baixuan Xu, Zhaowei Wang, Jiayang Cheng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Tianqing Fang, Yangqiu Song, Ginny Y. Wong, and Simon See. NewtonBench: Benchmarking generalizable scientific law discovery in LLM agents. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=Gk6umqW74m.

Amirmohammad Ziaei Bideh and Jonathan Gryak. LLM-ODE: Data-driven discovery of dynamical systems with large language models. *arXiv preprint arXiv:2603.20910*, 2026. URL https://arxiv.org/abs/2603.20910.

Amirmohammad Ziaei Bideh, Aleksandra Georgievska, and Jonathan Gryak. MDBench: Benchmarking data-driven methods for model discovery. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(24):19746–19754, 2026. doi: 10.1609/aaai.v40i24.39056. URL https://ojs.aaai.org/index.php/AAAI/article/view/39056.

ANNEX CONTENTS

A Benchmark construction and reference problems **21**

A.1 Collection and materialisation 21

A.2 Information supplied to the agent 21

A.3 Observation mappings, parameters, and initial conditions 22

A.4 Reference construction and interpretation 23

A.5 Roster and table counts 23

B Implementation and execution model **24**

B.1 Preparation, orchestration, and numerical state 24

B.2 MCP tools and resources 25

B.3 Semantic layer and mechanistic world model notation 26

B.4 Trust boundary and resistance to score manipulation 27

B.5 From a proposed revision to a stored version 28

B.6 Reconciliation of model and PEtab specification 29

B.7 Parameter estimation and numerical diagnostics 29

B.8 Data access and durable provenance 34

B.9 Agent roles and access to biological knowledge 34

B.10 Budgets, stopping, and final candidate selection 34

C Software setup, execution, and run records **35**

C.1 Installation and dependencies 35

C.2 A local agent run 35

C.3 Preparing empirical problems and reference fits 36

C.4 Selection criteria and computational budgets 36

C.5 Cluster execution and retained records 36

C.6 Configurations of the scored runs 37

D Evaluation cohorts, scoring, and numerical protocols **37**

D.1 Evidence sources and version scope 37

D.2 Cross-problem sweep summaries 39

D.3 Topological comparisons with the published models 39

D.4 Training fit per model version 44

D.5 Fujita configurations and fixed-parameter prediction 44

D.6 Additional validation systems in Figure 4 55

D.7 Neural ODE distillation protocol 69

E Fujita model diagrams and parameter uncertainty **72**

E.1 Parameter uncertainty around the plotted optima 72

F Reference models in SBGN Activity Flow notation **75**

G Agent-generated Fujita model sources **93**

H Agent-generated benchmark model sources **130**

H.1 Perelson . . . 130
H.2 Crauste . . . 133
H.3 Rahman . . . 139
H.4 Brannmark . . . 145
H.5 Boehm . . . 149
H.6 Armistead . . . 155
H.7 Fiedler . . . 158
H.8 Bruno . . . 163
H.9 Oliveira . . . 168
H.10 Sneyd . . . 175
H.11 Blasi . . . 179
H.12 Schwen . . . 191
H.13 Bachmann . . . 196
H.14 Raimundez . . . 208
H.15 Isensee . . . 216
H.16 Alkan . . . 226
H.17 Lucarelli . . . 240

I Agent prompts and benchmark inputs **264**

I.1 Modelling lead . . . 264
I.2 Specialist analysts . . . 269
I.3 Biological curator . . . 278
I.4 Control prompts . . . 280
I.5 Benchmark-specific inputs . . . 282

A BENCHMARK CONSTRUCTION AND REFERENCE PROBLEMS

The reported experiments span multiple software revisions due to time and computational cost constraints. Unless specified otherwise, differences between the revisions used for these experiments did not materially affect model construction, evaluation, or the conclusions reported here. We will rerun all experiments using a single software revision upon publication.

This appendix documents benchmark preparation in the `petab-mcp` source repository at revision `e0d6921`, with input-protocol blinding documented at `7d22712`. It describes the available implementation, rather than asserting that every historical run used that revision. The current generated package, imported at manuscript revision `b4209e6`, carries producer stamp `09af5ef+dirty`; the Figure 3 score export retains `604449e+dirty`, while the refreshed training sweep records `d958571+dirty` and its constituent run revisions. Appendix D.1 separates these evidence sources. These identifiers distinguish implementation documentation, generated summaries, and analysis provenance.

A.1 COLLECTION AND MATERIALISATION

The empirical suite comprises 18 problems selected from the Benchmark-Models-PEtab collection (Hass et al., 2019). We initially selected 13 smaller problems, then added five larger problems with more training measurements and generally more conditions and observables: Isensee, Alkan, Bachmann, Lucarelli, and Raimundez. The authoritative roster is defined in `src/petab_harness/paths.py`. It spans biochemical networks, cell-population dynamics, within-host infection, and epidemic transmission. The biological problems include dynamic and steady-state measurements, so constructing an ODE model does not imply that every dataset directly observes a trajectory.

Preparation is performed by `examples/setup.py`, which reads each example's `run.py` configuration and calls the materialisation functions in `src/petab_harness/benchmark.py`. The default collection revision is `1763dfcd66d25bfd97492cfedf6a730472c383d0`. An existing collection can instead be supplied through `BENCHMARK_MODELS_PETAB`; this override uses that checkout as provided, so its revision must be recorded separately. Ordinary materialisation performs no parameter fitting and leaves the committed reference summary unchanged.

Each problem is written as PEtab version 1 tables and a YAML specification (Schmiester et al., 2021), accompanied by an empty SBML model (Hucka et al., 2003) and an empty Antimony representation. PEtab version 2 and PEtab SciML extend the broader modelling ecosystem (Pathirana et al., 2026; Persson et al., 2026); the materialisation path described here uses version 1. The model identifier is a neutral placeholder. Measurements are retained, while identifiers and table representations undergo configured relabelling, observable merging, and observable-formula blinding (below; distinct from the biology blinding of O4). This is consequently a curated transformation of a published estimation problem. Compact observable tables are retained during preparation; timepoint-specific observable and noise parameter overrides are expanded only in a simulation-facing copy when required by AMICI.

A.2 INFORMATION SUPPLIED TO THE AGENT

Agents receive empirical measurements, their observable and condition identifiers, assay annotations, prescribed experimental inputs, and curated parameter information. The published dynamical model and narrative study context are withheld. Source-bearing identifiers are relabelled where configured, and free-text condition names are removed. The agent must construct a candidate mechanism and the mappings connecting its states to measured quantities. The measurement, condition, and observable tables provide the fixed experimental specification within which that construction takes place. With assistance from Anthropic Claude, we spot-checked agent traces to verify that the configured information-access restrictions operated as intended (Appendix B.4).

The retained information can include substantive biological priors. Per-example setup notes (Appendix I.5) specify known totals, stimulus rules, initial populations, or qualitative constraints that cannot be recovered from the supplied measurement rows alone. For example, the Crauste setup (prompt in Listing 148) identifies an unobserved pathogen and describes its growth, clearance, and role in T-cell activation. Such supplied constraints delimit the model-construction task; they must be

considered when interpreting which components an agent independently proposes. Withholding the published model does not establish that its source is unrecognisable from the remaining biological entities or observations.

These statements describe the configuration retaining biological annotations. The current biology-blinding implementation removes biological names, descriptive metadata and mechanistic coupling guidance, while preserving prescribed time-dependent inputs as anonymised numerical rules. The agent is instructed to implement these rules and use their defined stimuli in its equations (Appendix D.5).

Where a held-out measurement table is configured, preparation stores it outside the agent-facing problem directory. The implementation checks compatibility of the observable, condition, and parameter tables and rejects overlapping observable–condition–time tuples between training and held-out measurements. This allows later timepoints of an existing condition to be withheld. The documented preparation path also rejects held-out problems requiring incompatible timepoint-specific override expansion.

A.3 Observation mappings, parameters, and initial conditions

For a candidate model m, PEtab couples a dynamical system to an experimental and measurement specification (Schmiester et al., 2021). Let c identify a simulation condition together with its optional pre-equilibration condition. The model states satisfy

$$\dot{x}_c(t) = f_m\big(t, x_c(t), u_c(t); \beta_c(\theta)\big), \qquad x_c(0) = x^0_{m,c}(\theta), \tag{1}$$

where $u_c(t)$ contains prescribed inputs and $\theta \in \Theta_m$ contains estimated parameters. The map β_c combines these parameters with fixed values and condition-specific overrides; an override can be a supplied number or a reference to an estimated parameter. Each measurement row i specifies a value y_i, time t_i, observable o_i, and condition pair c_i. Replicates remain separate rows. A steady-state measurement uses the simulated equilibrium in place of $x_{c_i}(t_i)$.

Published observable formulas can reveal species names, complex composition, stoichiometry, or combinations of latent states. Preparation replaces this mechanistic content with `readout_*` placeholders that the agent must define through readout rules (model assignment rules); a placeholder is named `readout_<observableId>` unless the problem shares one readout between several observables. Assay descriptions retain selectivity, such as whether a measurement detects total protein or a phosphorylated form. This information constrains the interpretation of the readout without providing the reference model's state decomposition.

The separation is between a biological readout and its experimental calibration. An agent may need to define a sum or weighted combination of its own states, while the supplied observable template retains assay-specific scales, offsets, and noise parameters. Writing h_m for the readout mapping, the vector of agent-defined biological readouts (one readout rule each; the prompts call these rules measurement mappings), the prediction and noise scale for row i are

$$\begin{aligned} b_i(\theta) &= h_m\big(t_i, x_{c_i}(t_i), u_{c_i}(t_i); \beta_{c_i}(\theta)\big), \\ \widehat{y}_i(\theta) &= a_i\big(b_i(\theta); \theta\big), \qquad \sigma_i(\theta) = s_i\big(b_i(\theta), \widehat{y}_i(\theta); \theta\big) > 0. \end{aligned} \tag{2}$$

Here a_i and s_i are the supplied observable and noise templates for o_i, including measurement-row parameter overrides; known time and input arguments are suppressed. For example, $a_i(b;\theta) = \alpha_i b_j + \gamma_i$ calibrates readout j by an observation scale α_i and offset γ_i. Several observables can read the same readout with different calibrations, so a problem can have fewer readouts than observables. The composition of h_m and a_i gives the observation mapping g_m in Appendix B.3.

Let T_i be the declared linear, natural-log, or base-10-log observable transformation, and q_i the declared standard normal or unit-scale Laplace error density. PEtab assumes independent measurement errors, giving the measurement NLLH

$$z_i(\theta) = \frac{T_i(y_i) - T_i(\widehat{y}_i(\theta))}{\sigma_i(\theta)}, \qquad \ell_m(\theta) = -\sum_{i=1}^{n} \log\left[\frac{q_i(z_i(\theta))}{\sigma_i(\theta)}\,|T_i'(y_i)|\right]. \tag{3}$$

The noise scale is a standard deviation for normal errors and a scale parameter for Laplace errors, both in the transformed observation space. The Jacobian accounts for the density of measurements

supplied in linear space; its contribution cancels in ΔNLL when measurements and transformations are shared. Fitting searches Θ_m to minimise $\ell_m(\theta) + P_m(\theta)$, where P_m contains retained objective-prior terms and is zero when none are present (Appendix A.4). Signal-dependent noise expressions are rewritten in terms of the same `readout_*` symbol, preserving their statistical form. Keeping this noise model specification fixed was required to make NLLH values directly comparable between models constructed by agents and the published references; estimated noise parameter values can still differ between fits.

The parameter table is a curated subset of the published problem. It can retain observation parameters, fixed setup quantities, and selected initial-condition parameters, together with their estimation flags, bounds, estimation scales, and available prior annotations. Estimated kinetic parameters are introduced through the candidate model. Bounds constrain $\theta^{\min} \leq \theta \leq \theta^{\max}$ componentwise in linear parameter units; the declared linear or logarithmic optimisation coordinates are distinct from the observable transformations T_i. A quantity fixed in a publication is not automatically treated as given knowledge: a value encoding an absent reaction or a previous fitting decision would itself constrain the mechanism being sought.

Initial conditions require corresponding care. Known baselines can be supplied as fixed parameters; estimated initial populations and compound initial expressions must preserve their intended dependence on model parameters. Condition-specific initial values remain experimental inputs. When pre-equilibration is prescribed, $x^0_{m,c}(\theta)$ is constructed from the simulated steady state under the pre-equilibration condition, followed by any initial-state overrides in the simulation condition. For Fiedler, preparation adds a basal pre-equilibration condition and stimulus gate to preserve the pre-stimulus steady-state constraint after removing the published model. These choices influence both the admissible trajectories and the parameter count, and belong to the benchmark specification.

A.4 Reference construction and interpretation

Reference summaries are regenerated only through the explicit `-refit` preparation option. The reference fitting path loads the collection's published SBML model and parameter table directly. It uses AMICI, pyPESTO, and Fides, with per-example fitting budgets. The reported objective is the better of the multistart result and evaluation at the published nominal parameter vector. Objective-prior columns are removed for reference fitting and scoring, so the reference objective is the measurement NLLH. At the documented revision, candidate fits retain objective priors on supplied parameters, where present, and their contributions remain in the reported training objective; newly introduced kinetic parameters do not inherit the published model's priors. These differences in prior treatment have only small effects on scores and do not materially influence the results of this study.

Documented exceptions adjust the reference problem when necessary for comparison. Selected published-fixed parameters are freed within their published bounds; both constrained and freed fits are evaluated, the better attainable likelihood is retained, and the larger parameter count is charged. Oliveira applies matching noise-parameter changes to candidate and reference specifications. Curated fitted literals embedded in the published equations can also contribute to the reference count, as in Fujita. These adjustments mean that the reference is derived from the publication rather than universally identical to its original estimation specification. Negative ΔBIC or ΔNLL favours the candidate over this numerical reference without establishing a uniquely correct mechanism. Appendix F visualises all 18 reference models using SBGN Activity Flow notation.

A.5 Roster and table counts

Table 1 includes the generated roster without modification. Its reference values come from committed summaries; its observable and condition counts are the training counts of Figure 2, which include only observables and conditions that carry training measurements. The materialised tables can declare more: Fujita supplies sixteen condition rows but training responses for six, and Fiedler adds a basal pre-equilibration row. Neither table-row counts nor parameter counts measure the number of independently identifiable biological mechanisms.

Table 1: Published benchmark problems in the generated manuscript snapshot. k_{ref} is the reference parameter count used for scoring, n is the number of training measurement rows, and $\mathrm{nll}_{\mathrm{ref}}$ is the stored reference offset used to compute training ΔNLL. Obs. and Cond. count the observables and simulation conditions that carry training measurements, which is what the reference fit used; a problem may ship more of either, and the surplus is not part of the estimation task. Fujita declares sixteen conditions and is trained on six, the other ten being the held-out arms of Figure 3, and Perelson's second condition is its pre-equilibration. The offsets depend on each problem's measurements and noise model and should not be compared directly across rows.

Example	k_{ref}	n	$\mathrm{nll}_{\mathrm{ref}}$	Obs.	Cond.
perelson	3	16	223.49	1	1
blasi	9	252	-1090.56	15	1
rahman	9	23	21.15	1	1
crauste	12	21	190.97	4	1
armistead	14	58	-301.92	4	2
sneyd	15	135	-319.79	1	9
fiedler	22	72	-58.58	2	3
boehm	9	48	138.22	3	1
brannmark	22	43	141.82	3	8
schwen	30	286	939.88	4	19
bruno	13	77	-46.69	5	6
oliveira	14	120	379.37	2	1
fujita	20	144	-66.58	3	6
isensee	46	687	3941.46	3	123
alkan	65	1733	3171.38	12	73
bachmann	115	541	-418.41	20	36
lucarelli	84	1755	1681.61	65	16
raimundez	142	627	539.19	5	170

B Implementation and execution model

This appendix describes the implementation at revision `e0d6921`. Paths are relative to the software repository. The description explains how a proposed biological model becomes a numerical estimation problem and an inspectable record. It is distinct from the protocols of the archived evaluations: the reported runs span earlier revisions and different fitting budgets. In particular, current validation, stopping, parameter-counting, and checkpoint selection rules must not be assumed to have operated identically throughout those cohorts. The run exports (Appendix C.6) and cohort-specific methods (Appendix D) determine which settings support each result.

B.1 Preparation, orchestration, and numerical state

The implementation separates problem preparation, agent orchestration, and numerical evaluation (Table 2). Preparation converts a published estimation problem into an agent-facing construction task. The harness then creates a working copy and launches a separate MCP server process. The server owns the mutable PEtab problem; the harness owns the agent interaction, run limits, and final comparison with the withheld reference. This boundary lets different agent configurations use the same editing and estimation interface. It does not make their information access or computational budgets identical; those remain experimental settings.

Materialisation starts from a pinned source collection and applies the example's declared curation: identifier replacement, supplied parameter definitions, and observable-formula blinding, where required. It retains empirical measurements and experimental information while replacing the reference mechanism with an empty SBML model. Reference fitting is a separate operation, so preparing an agent problem does not silently regenerate its comparison score. The server receives the working problem and numerical configuration, while the benchmark judge retains the published-reference information needed for final assessment.

Communication uses MCP over Streamable HTTP. This also separates tool messages from compiler output: AMICI compilation writes to the server's output stream, which would interfere with

Table 2: Principal implementation boundaries. Modules are grouped by their responsibility in a run rather than by their individual helper functions.

Responsibility	Main entry points and role
Preparation	`examples/setup.py` and `src/petab_harness/benchmark.py`: materialise the construction problem and manage separate reference preparation.
Orchestration	`src/petab_harness/runner/` and `src/petab_harness/agent/`: launch services, configure agents, enforce run limits, and archive outcomes.
Model revision	`src/petab_mcp/server.py`, `src/petab_mcp/session.py`, and `src/petab_mcp/operations.py`: apply a proposed change through reconciliation, validation, fitting, and version creation.
Numerical evaluation	`src/petab_mcp/engine.py` and `src/petab_mcp/fitting.py`: construct the AMICI/pyPESTO objective, estimate parameters, and produce simulation and optimization diagnostics.
Inspection and storage	`src/petab_mcp/data_access.py` and `src/petab_mcp/provenance.py`: expose diagnostic tables and preserve model versions and events.

a standard-input/output JSON-RPC channel. The harness captures server logs and manages the subprocess lifecycle, including termination when its parent disappears.

B.2 MCP TOOLS AND RESOURCES

The following overview covers the `petab_mcp` interfaces relevant to the reported study. The harness assigns tools by role and evaluated ablation configuration (Appendix B.9); an available tool need not have been invoked in every run. The held-out evaluation tool is not registered with LLM agents.

Problem definition and measurements. `describe_problem` returns the problem dimensions, whether a model has been authored, the current version, and the default number of optimisation starts. `list_observables` exposes the supplied observable descriptions (`observableName`), observable formulas, and noise models; `list_conditions` exposes the experimental conditions and their prescribed model inputs. `get_measurements` filters measurements by condition and observable. Its default response summarises counts, time ranges, value ranges, means, and preview points for each observable–condition pair; setting `summarize=false` returns paginated individual measurements, with a default of 200 rows per page. These tools inspect the supplied experimental specification.

Model and parameter inspection. `get_model` returns the current model as Antimony or SBML, or a requested version's stored Antimony source. `get_model_structure` returns a structured account of species, reactions, rate laws, and parameters, including whether each species' initial value is fixed, estimated, or supplied by a condition. `get_parameters` reports parameter values, bounds, scales, priors, estimated/fixed status, and roles such as `kinetic`, `observable_scaling`, `noise_sigma`, and `initial_condition`. Filters select parameter identifiers or roles. It also identifies supplied parameters not yet declared in the current model; the SQL parameter table, in contrast, contains only the live estimation problem.

Model revision and estimation. `make_changes` accepts a complete Antimony model and/or changes to parameter bounds and estimation scales. The alternative `model_edits` argument accepts a batch of structured operations: add or remove a reaction, replace a rate law, set an initial value, set or remove an assignment rule, or rename a symbol. `base_version` applies such a batch to a stored model and its search space. A non-empty justification is required; references and an override of the multistart count are optional. The call reconciles (Appendix B.6) and validates the proposal, fits it, and records a version, returning fit scores, diagnostics, warnings, and flags indicating whether the change was applied and committed. Fitted parameter values come from estimation; they cannot be supplied through the parameter-edit argument. There is no separate agent-facing fit call. The treatment of rejected changes, failed fits, and duplicates is described in Appendix B.5.

SQL access to data and numerical records. `list_data_tables` reports available table names, columns, and row counts. `query_data` runs DuckDB SQL over `measurements`,

conditions, observables, and parameters, together with simulations, simulations_history, optimization_starts, and optimization_trace. Simulation records associate each measurement with its prediction, residual, noise-normalised residual, and noise scale. The history table adds version identifiers for comparisons between candidates; optimisation tables describe individual starts and their iteration traces. Queries support filtering, joins, and aggregation, with 50 returned rows by default and pagination through offset. Raw measurement and SQL pages are capped at 1,000 rows. Explicit ordering is needed for stable SQL pagination. SQL runs on disposable copies with external file and network access disabled, so it cannot overwrite the problem or its stored scores (Appendix B.8).

Fit diagnostics. get_fit_summary returns the current likelihood and complexity scores, active selection criterion, parameter count including charged literals, fitted values, parameters near their bounds, and residual/noise flags. get_fit_diagnosis provides detailed residual or parameter reports, selected using sections=residuals, parameters, or all. These include the worst-fitting points, observable-wise residual tests, local parameter uncertainties and correlations, and variation across near-optimal starts (Appendix B.7). get_fitting_log returns a requested tail of the optimisation log, defaulting to 200 lines.

Version history and audit records. list_versions returns stored versions and their scores; get_version retrieves a selected model, parameter table, metadata, and available simulation and optimisation records. diff_versions compares two versions through a model-text diff, parameter additions/removals/changes, and fit-score differences. restore_version reinstates a stored model and fitted parameters without refitting, while appending a restoration record rather than erasing history. get_provenance and get_errors return recent events and rejected or failed operations, respectively, with a configurable limit of 50 records by default.

Progress and computational cost. get_progress reports the best stored version under the configured criterion, the number of fitted operations, recent improvement, the non-improving streak, and whether a stopping condition has been reached. get_run_timing reports accumulated server-side compilation and fitting times and counts. These are timings of numerical work, distinct from total agent elapsed time, token usage, or CPU consumption.

Modelling guidance and resources. When guidance files are present, list_skills lists their names, descriptions, and supporting files; get_skill retrieves a selected document or supporting file. The server also exposes read-only MCP resources under petab:// for model text, PEtab tables, versions, provenance, errors, simulations, the data schema, and the fitting log, with additional resources for available guidance. These resources provide an alternative retrieval interface, rather than additional model-editing operations. External biological databases and delegation to specialist agents are supplied by the harness, not by petab_mcp itself.

Held-out evaluation for the harness. When validation data are configured, evaluate_validation_data scores the current model using the dataset's declared protocol. Fixed-parameter evaluation retains the fitted parameter vector; transfer evaluation refits the estimated parameters for the target system, optionally overriding the multistart count. The target fit is not written back to the working model. The response records the evaluation status, mode, score, parameter and measurement counts, conditions, and observables. This tool is excluded from all modelling-agent allowlists: the harness invokes it for evaluation. That separation is implemented by tool selection in the harness, not by authenticated server roles (Appendix B.4).

B.3 Semantic layer and mechanistic world model notation

We adopt the mechanistic world model notation of Posner et al. (2026) to describe the relation between our semantic layer and variable, mechanism, and structure discovery:

$$\mathcal{W} = (\mathcal{T}, Z, \mathcal{M}, S). \tag{4}$$

Here $\mathcal{T}$ denotes semantic variable types, Z the latent variables, including mechanistic parameters, $\mathcal{M}$ a library of mechanisms, and S their bindings to particular variables. Each mechanism combines a typed input/output-role signature with a transformation; a binding assigns those roles to variables of the corresponding types.

We use *semantic layer* for a model representation together with the interpretation that connects its entities and processes to executable mathematics. For the numerical implementation, let $m \in \mathcal{L}_{\mathcal{S}}$ denote a complete model proposal in the language of a semantic layer $\mathcal{S}$, and let E denote the supplied experimental and observation specification. We denote its translation to dynamics, an observation mapping, and a parameter space by $\mathcal{C}_{\mathcal{S}}$:

$$\mathcal{C}_{\mathcal{S}}(m, E) = (f_m, g_m, \Theta_m), \qquad \dot{x} = f_m(x, u; \theta), \quad \widehat{y} = g_m(x, u; \theta), \tag{5}$$

where x contains model states, u denotes prescribed experimental inputs, and $\theta \in \Theta_m$ contains the estimated parameters. Time dependence, initial conditions, and observation noise are specified by the model together with E; the displayed equations suppress these additional arguments (the full PEtab specification is given in Appendix A.3). With $V(m, E)$ denoting the implemented acceptance checks, the candidate class is

$$\mathcal{H}_{\mathcal{S},E} = \big\{\mathcal{C}_{\mathcal{S}}(m, E) : m \in \mathcal{L}_{\mathcal{S}},\ V(m, E) = 1\big\}. \tag{6}$$

In the mechanistic world model notation, the states x and mechanistic parameters in θ instantiate Z. Reaction kinetics together with their stoichiometric effects, and model rules, specify transformations; references to particular species and parameters encode their bindings S. `gemot` stores these transformations together with their bindings in each authored model. It does not yet maintain a separate reusable mechanism library $\mathcal{M}$, and semantic types $\mathcal{T}$ are not explicitly learned or enforced (Section 3). The notation therefore makes explicit both the variable, mechanism, and structure discovery implemented here and the remaining steps towards a mechanistic world model. Semantic regularisation can operate through restrictions on $\mathcal{H}_{\mathcal{S},E}$ and through biological priors guiding proposals within it. It is distinct from the parameter-count penalty used in scoring; no separate semantic loss is optimised in the reported runs.

In our implementation, the model-language component is Antimony, implemented by libAntimony (Smith et al., 2009). It represents named species, compartments, reactions, kinetic laws, and rules. In `src/petab_mcp/operations.py`, the authoring path invokes PEtab's `SbmlModel.from_antimony` to translate that description into SBML. Reconciliation connects it to the supplied parameter table and observable and noise templates (Appendix B.6); in particular, the agent's assignment rules define the biological readouts required by PEtab. The resulting problem is compiled for numerical evaluation. Thus libAntimony supplies the mechanism's formal interpretation, while the complete mechanism-to-measurement contract also depends on PEtab and the server's checks. Merely giving a variable a biological name does not validate that interpretation. The current validation settings do not establish unit or thermodynamic consistency, and no automatic biological-ontology verification is claimed.

The inspected conversion path uses `getSBMLString`, which flattens Antimony submodules into a single executable model. This is a property of the present pathway, not a limitation of Antimony's modular syntax or SBML's `comp` package (Smith et al., 2015). The implementation check used PEtab 0.8.2 and Antimony 3.1.3; these local dependency versions do not identify every historical run environment. Structured authoring operations address reactions, rules, initial values, and symbols. They do not currently expose a curated library of modules with validated biological interfaces and applicability conditions.

Changing this layer would change the units in which an agent proposes and revises mechanisms. For example, a BNGL rule can specify a transformation of molecular sites across many possible complexes, while PySB can package rule generation into reusable Python components (Harris et al., 2016; Lopez et al., 2013). These alternatives would need their own translation, validation, complexity accounting, and readout-mapping interfaces before being evaluated under the same benchmark. A Python-based authoring layer would also need an execution policy appropriate to code submission; the current MCP access restrictions should not be assumed to carry over unchanged. Comparing such layers is prospective work, not a completed ablation.

B.4 Trust boundary and resistance to score manipulation

The server separates the agent's authority to propose a model from the authority to evaluate it. The exposed authoring operations accept model definitions and parameter-search settings; they provide no operation for rewriting measurement, condition, or observable tables, changing the scoring implementation, or submitting a replacement likelihood. Fitted parameter values, likelihoods, complexity

counts, and version records are produced by the numerical pipeline. The benchmark judge reads these records when assessing a run, rather than treating the agent's final narrative as evidence of a successful fit. Consequently, compliance with the evaluation contract does not depend solely on instructions asking the agent to respect it.

The boundary is enforced at several points. Parameter-edit schemas restrict the editable fields, and server-side checks reject unsupported changes. Model proposals are reconciled and validated before replacing the working problem. Checks cover the required model structure, readout rules, use of experimental inputs, and preservation of protected supplied constants. They also reject specified forms of observation-parameter reuse, including the use of a noise parameter inside the mechanism. The current complexity calculation charges non-exempt numerical literals embedded in model equations, reducing the opportunity to hide tunable choices outside the estimated-parameter table. These are executable checks with defined scopes and exceptions, rather than assessments made by another language model. Structural checks establish the presence and use of model elements, without proving their causal meaning; some checks are best-effort when their required metadata cannot be read.

Inspection preserves the same separation. DuckDB queries use disposable copies with external access disabled, so SQL updates cannot modify the underlying problem or stored scores. Restoring an earlier model selects a recorded version without deleting the intervening history. Accepted revisions, numerical outcomes, and rejected operations remain available for auditing. Together, these controls make reported improvements traceable to an evaluated candidate and prevent direct alteration of the benchmark records through the supplied tools.

This claim is scoped to the configured harness and MCP interface. The default agents receive curated tool sets with direct access to the host filesystem disabled. The local server is not an operating-system sandbox, and withholding a tool from an agent is not equivalent to authenticating separate server roles. Broader host or endpoint access would require additional isolation. Evaluation integrity also differs from scientific adequacy: a model can improve a correctly computed objective through overfitting or poorly constrained assumptions. The checks do not prove that every possible strategy for exploiting an objective has been excluded.

B.5 From a proposed revision to a stored version

The principal authoring operation, `make_changes`, requires a non-empty justification. It accepts either a complete Antimony model or a batch of structured operations addressed to model identifiers. Parameter bounds and estimation scales can be changed in the same call. The two model-authoring routes are mutually exclusive. Structured edits may target an earlier version; the implementation retrieves that version's submitted source and parameter search space and records the resulting branch relationship. Bibliographic references can accompany an edit but are not mandatory.

Reconciliation (Appendix B.6) aligns the proposed SBML model with the PEtab parameter and observation interface. Parameter edits run after reconciliation so that they can address newly introduced parameters. These operations act on a candidate copy. Validation checks the resulting problem, including the presence of dynamics and preservation of protected supplied constants, before the candidate replaces the live state. A rejected candidate therefore leaves the previous working problem intact and returns an error explaining the rejected operation.

An accepted edit proceeds immediately to fitting. When evaluation creates a version, its model, fitted parameters, scores, and diagnostics are retained together. The submitted source is stored separately from the reconciled Antimony representation, allowing subsequent edits to refer to the agent's model text. If fitting or compilation fails before creating a version, the server restores the previous working state. The edit and its failure remain in the event record. If a later diagnostic or response-construction error occurs after a snapshot was created, that version is retained; an error response alone is therefore insufficient to infer whether the model state changed.

Duplicate detection further distinguishes tool calls from fitted versions. An unchanged candidate can return its existing score without creating another version, whereas an increased multistart budget can justify a new fit. Structure-level checks also identify repeated visits and can stop repeated fitting of the same structure. Consequently, tool-call counts, optimization counts, and version counts measure different activities.

B.6 Reconciliation of model and PEtab specification

Purpose and scope. Reconciliation makes an authored model consistent with the supplied PEtab specification (Appendix A.3). After Antimony conversion or structured model edits, `src/petab_mcp/reconcile.py` updates the candidate SBML model and parameter table before validation and fitting. Measurement, condition, and observable tables remain unchanged. The agent supplies the mechanism and its readout rules; reconciliation infers neither missing interactions nor readout rules.

Removing unused symbols. The pass first removes species and parameters that are not referenced by reactions, model equations, initial assignments, or the fixed PEtab tables. Rule targets are retained, and initial assignments belonging to removed symbols are removed with them. This is a syntactic cleanup, not a test of dynamical relevance or identifiability: a referenced but redundant state can remain. If no species survives, an internal placeholder keeps the model safe for downstream handling; it does not replace the separate requirement for an authored model with dynamics.

Making initial conditions explicit. Existing parameterised initial conditions, including compound expressions such as $x_2(0) = \theta_1\theta_2$, are preserved. Other nonzero initial values generally receive a named initial-condition parameter and become estimable. Zero or unspecified initial pools normally receive no new parameter, and an initial value of one is treated as a fixed unit normalisation; supplied parameter definitions take precedence over these defaults. Species initialised through condition-table overrides receive no additional initial-condition parameter. Assignment-rule targets also receive none, and conflicting initial assignments are removed. These conventions prevent the introduction of free parameters that are immediately overridden, while retaining the parameter dependence of the agent's initial state assumptions.

Synchronising the parameter table. New model parameters receive PEtab rows, and obsolete rows are removed. Parameters needed only by observable or noise formulas and measurement-row overrides are retained. Symbols that are species, computed rule targets, or condition-table override targets are excluded where PEtab forbids them as parameter-table entries. If a reintroduced parameter was supplied with the problem, its original row is restored, including estimation status, bounds, scale, nominal value, and available prior annotations (Appendix A.4). Other new kinetic parameters are estimated, with default bounds that include the submitted nominal value; positive values normally use logarithmic coordinates. Parameters used as exponents receive linear coordinates and narrower default bounds. Existing search settings are retained, apart from the initial-condition conventions above. Explicit edits to bounds or scales run after reconciliation, so they can address newly introduced parameters.

Validation and feedback. The reconciled candidate then undergoes PEtab and SBML checks and the additional safeguards in Appendix B.4. These check required readout rules, use of prescribed inputs, protected supplied constants, and prohibited reuse of observation parameters. Missing readouts are reported to the agent for correction, rather than filled automatically. The edit pipeline records added and removed parameters and pruned species, and reports how each species' initial condition is fixed, estimated, or set by a condition. Rejection leaves the working problem unchanged; accepted proposals proceed to fitting under the versioning rules in Appendix B.5.

Relation to complexity accounting. Reconciliation determines which named quantities enter the estimated-parameter count. The scorer separately adds non-exempt numerical literals in model equations to that count (Appendix B.7); these literals are not automatically promoted to fitted parameters. Consequently, the submitted model and its reconciled representation are both retained for audit. Reconciliation and validation check consistency with the experimental specification; they do not establish biological correctness or uniqueness of the mechanism.

B.7 Parameter estimation and numerical diagnostics

AMICI supplies compiled simulation and sensitivity calculations, pyPESTO constructs and coordinates parameter estimation, and Fides performs local trust-region optimization (Fröhlich et al., 2021; Schälte et al., 2023; Fröhlich & Sorger, 2022). Multistart optimization explores the supplied parameter bounds; the best returned solution is written back to PEtab nominal values. Reported fitted values are in linear units, with their estimation scales recorded separately. Neither multiple starts nor convergence of an individual start establishes a global optimum.

Warm starts across model iterations. To reuse numerical work after a model revision, the implementation reserves $\lfloor N_{\text{starts}}/2 \rfloor$ starts for warm initialisations when usable parameter values are available; the remaining starts are sampled afresh. The current nominal point, normally the preceding fit's best solution, provides the reference point. Distinct finite solutions from the preceding multistart are also reused, matching shared parameters by identifier and converting between estimation scales through linear values; parameters absent from that fit retain their current nominal values. For newly added parameters, additional candidates test unit values and lower or upper bounds, jointly and, when several parameters are added, one at a time against unit values for the others. These candidates receive priority within a limited part of the warm-start budget, followed by the nominal point and previous solutions ranked by objective. Candidates are clipped to the current bounds and deduplicated in the free parameter coordinates.

Any remaining warm-start slots are filled by Gaussian jitter around the clipped nominal point, with standard deviation 0.05 times each parameter's bound width in its estimation coordinates (linear, log, or $\log_{10}$), followed by clipping. Unbounded coordinates receive no jitter. A separate random-number generator with seed zero makes these perturbations reproducible without changing the fresh-start sampling stream. If warm initialisation is unavailable, all starts are sampled afresh. This combines reuse of earlier fits with exploration after structural changes; the candidate values do not guarantee recovery of the preceding model's likelihood.

For eligible observation parameters, the current implementation attempts hierarchical estimation through pyPESTO (Loos et al., 2018; Schmiester et al., 2020), with a joint-estimation fallback when that construction is unavailable. Parameters eliminated from the outer optimization remain estimated parameters in the PEtab problem and its complexity accounting. The current scorer additionally charges non-exempt numerical literals embedded in model equations. Archived parameter counts must be interpreted using each run's scoring convention; the availability of hierarchical estimation does not imply that every archived fit used it.

Diagnostic interfaces and scope. Fit results and `get_fit_summary` expose scores and compact warnings; `get_fit_diagnosis` supplies residual or parameter reports on request. DuckDB tables provide individual simulated measurements, residuals, and optimisation records for further analysis, while the fitting log exposes numerical messages (Appendix B.2). The diagnostics below describe the implementation at the documented revision for the study's tool configuration, rather than the exact feedback available in every archived run. They concern the supplied training data. Held-out evaluation is reserved for the harness and excluded from modelling-agent tool sets. Numerical agreement, diagnostic passes, and successful integration do not establish biological adequacy or uniqueness of a mechanism.

Fit scores and numerical consistency. The summary reports fit status, the likelihood objective, the number of measurements, and the complexity count k, separating estimated parameters from charged numerical literals. It returns BIC, AIC, AICc, CAIC, HQC, and TIC where defined, together with the configured criterion used for ranking versions (configuration in Appendix C.4). Undefined scores are returned as missing, for example AICc when the sample is too small relative to k, or TIC when its effective parameter count cannot be computed. Compact reports omit that effective count, while archived records retain it. Improvement reports compare the active criterion with the previous best checkpoint; a decrease of at least two units updates that checkpoint, allowing smaller successive improvements to accumulate towards the threshold.

The fit archive retains both the best optimiser objective and the likelihood obtained by independently re-simulating its fitted parameters. A finite optimiser objective is used as the authoritative fitting score. An `objective_note` identifies failed re-simulation or a discrepancy exceeding $10^{-3}\max(1, |\ell_{\text{opt}}|)$, where ℓ_{opt} is the optimiser's negative log-likelihood. The summary also counts missing simulated measurements. Thus a finite score does not imply that the independent simulation reached every measurement. The reported χ^2 requires a complete residual evaluation, rather than summing only successfully simulated points. The server also returns $\chi^2/(n-k)$ and an upper-tail χ^2_{n-k} probability when defined; these should not be interpreted as calibrated adequacy tests when noise scales are fitted.

Individual residuals and observable summaries. Each simulation-table row associates an observable, condition, and measurement time with the measurement y_i, prediction $\widehat{y}_i$, evaluated noise scale

σ_i, raw residual, and noise-normalised residual:

$$r_i = y_i - \widehat{y}_i, \qquad z_i = \frac{T_i(y_i) - T_i(\widehat{y}_i)}{\sigma_i}, \qquad \chi^2 = \sum_i z_i^2. \tag{7}$$

Here T_i is the declared linear, natural-log, or base-10-log observable transformation; the noise formula can give a different σ_i at each point. The field `residual_weighted` contains z_i, not z_i^2. Missing predictions, unevaluable noise, or zero noise leave the normalised residual missing. The history table adds a version identifier, enabling comparisons at the same measured point across models.

The detailed residual report gives each observable's usable point count, contribution to χ^2, mean z_i, largest $|z_i|$, and residual-test summaries. It also lists the eight worst individual discrepancies by default, including their measurements, predictions, conditions, and times. To bound the response, the default observable table combines the 12 largest χ^2 contributions, three largest absolute mean residuals, and six observables flagged for residual structure: at most 21 distinct entries. Omitted counts, omitted χ^2, and the number of omitted flagged observables remain visible. Setting `max_observables=0` removes this truncation; `worst_n` controls the point list separately. The aggregate residual-structure verdict is returned automatically with fit results and score summaries.

Bias and temporal structure. Sign and runs tests use points with available weighted residuals, average replicate residuals at each condition–time pair, and discard zero or non-finite residuals. Within each condition, an exact two-sided binomial sign test checks the balance of positive and negative residuals. An exact two-sided Wald–Wolfowitz runs test checks their ordering in time, conditional on the sign counts. For each observable and test, condition-specific probabilities are combined using Fisher's statistic $-2\sum_{j=1}^{J}\log p_j$, compared with χ^2_{2J} under the independence assumption. Either combined $p < 0.05$ flags the observable. There is no adjustment across observables or between the two test families, so these flags are screening diagnostics. Reports include combined probabilities, contributing-series counts, sign counts, and low-power flags. Fewer than eight nonzero design-point residuals, or no computable runs test, triggers a heuristic low-power warning; this is not a formal power calculation. Runs tests cannot assess a series containing only one residual sign.

Replicate lack of fit. A separate check compares predictions with replicate scatter without relying on the fitted noise scale, so it can remain available when that scale is zero or unevaluable. At a design point i with n_i replicates, sample standard deviation sd_i, and common prediction $\widehat{y}_i$, it computes

$$t_i = \frac{\overline{y}_i - \widehat{y}_i}{\mathrm{sd}_i/\sqrt{n_i}}, \qquad p_i = 2\Pr(t_{n_i-1} \geq |t_i|)\,. \tag{8}$$

Measurements and predictions are expressed in the noise-additive, scale-normalised coordinate; proportional noise uses an approximate logarithmic coordinate. A usable point requires at least three finite replicates, positive sample variance, and a finite prediction shared by its replicate rows. At least three usable design points must share a noise-parameter family. Probabilities are Holm-adjusted within each such family and flagged below 0.01, without additional correction across families. The compact report gives availability, usable and flagged counts, the largest standardised discrepancy, the smallest adjusted probability, and up to five worst flagged points.

An additional guardrail subtracts the number of estimated non-noise parameters from the pooled number of usable design points. Parameters that also affect observation scaling remain in this count. A non-positive difference makes the report unavailable and discards its flags; a difference of one or two marks an unflagged result as weak; larger differences receive the implementation's powered designation. These are heuristic evidence-strength categories, not formal power estimates. Unknown parameter counts prevent a power claim. Reports distinguish absent replicates, insufficient replication, too few usable points, and unresolved inputs, and identify partially untested noise families. A flagged fitted prediction indicates remaining discrepancy; it does not prove that no parameter vector could fit that point.

Estimated noise scales. For estimated noise-only parameters, the noise report gives the fitted value, effective observation standard deviation, supporting point count, empirical data spread, and their ratio. It also reports the data-derived upper-bound cap, a data-derived noise estimate where available, the method used to obtain it, and the ratio of fitted to estimated noise. Comparisons use a common noise-additive, scale-normalised coordinate rather than comparing a raw parameter directly with

measurements in incompatible units. An effective scale above 0.75 of the empirical data standard deviation is labelled inflated, and below 0.01 is labelled collapsed. These are heuristics; fewer than five supporting points or unevaluable quantities yields an unknown status. Fixed noise parameters are not included in this report.

The preferred data-derived noise estimate pools within-design-point replicate variance and requires at least six residual degrees of freedom. Otherwise, the implementation uses successive measurement differences within time series, $\widehat{\sigma}^2 \simeq \mathrm{mean}[(\Delta y)^2]/2$. These estimates do not use fitted residuals, although their scale normalisation can depend on fitted observation scales. The difference-based fallback includes true temporal change and is not a reliable lower bound on measurement noise. These additional ratios provide advice rather than formal tests. A `scale_coupled` flag identifies noise formulas in which the noise parameter is combined with another estimated parameter, such as an observation scale or an additive offset, cautioning against direct cross-model comparisons of noise parameters. Reporting these diagnostics does not itself change the parameter bounds.

Parameter values, bounds, and local uncertainty. Fitted values are reported on their original scale alongside the estimation scale. After the first fit, compact responses may list only added, removed, or changed parameters (more than a 1% relative change from the parent version), with an unchanged-parameter count; complete values remain available through parameter and version inspection. A bound warning identifies estimates within 1% of the allowed bound span of either endpoint, measured in the estimation coordinate. It reports the parameter, value, bound, and side. A bound warning does not by itself demonstrate non-identifiability.

Parameter diagnosis uses local objective geometry at the stored fitted optimum. The preferred sensitivity-based information approximation includes measurement-residual and noise contributions,

$$H = J_z^\top J_z + J_{\log\sigma}^\top J_{\log\sigma}, \qquad C = H^+, \tag{9}$$

with an objective-Hessian or residual-sensitivity fallback when required. Here J_z is the Jacobian of the noise-normalised residuals z_i defined above, and $J_{\log\sigma}$ is the Jacobian of the natural logarithms of the evaluated measurement noise scales σ_i. The subscripts identify the quantities being differentiated. Writing η_j for free parameter j in its estimation coordinate (linear, natural-log, or $\log_{10}$), the entries are $(J_z)_{ij} = \partial z_i/\partial\eta_j$ and $(J_{\log\sigma})_{ij} = \partial \ln\sigma_i/\partial\eta_j$, evaluated at the stored optimum. Rows therefore index measurements and columns index free outer parameters. Residual derivatives include dependence of both predictions and noise scales on these parameters. The second term captures parameter dependence of the noise scales and vanishes when those scales are parameter-independent. Here H^+ is the Moore–Penrose pseudoinverse; H is a local approximation, not necessarily the exact expected Fisher information. The per-parameter report gives $\mathrm{SE}_j = \sqrt{\max(0, C_{jj})}$ in the estimation coordinate. For log and log10 parameters it also expresses this as a one-standard-error multiplicative factor, $\exp(\mathrm{SE}_j)$ or 10^{SE_j}, rather than a confidence interval. For hierarchical fits, these sensitivities hold the fitted inner parameters fixed; those parameters are excluded from the outer-parameter geometry, and the report explicitly notes their omission. Pseudoinversion discards null directions, so even small finite standard errors must be read together with the rank diagnostics.

Correlations, flat directions, and multistart variation. Marginal correlations are $C_{ij}/\sqrt{C_{ii}C_{jj}}$; partial correlations use $-H_{ij}/\sqrt{|H_{ii}H_{jj}|}$. Pairs with absolute correlation at least 0.9 are reported in descending order. The report gives matrix rank, a rank-deficiency flag, and a condition number based on the largest and smallest strictly positive eigenvalues; the latter can remain finite for a rank-deficient matrix. Eigenvectors with $\lambda/\lambda_{\max} < 10^{-3}$ identify flat directions, with up to four dominant parameter loadings above 0.15 in absolute value per direction. Pair and direction lists default to eight entries each; the per-parameter table is not subject to that truncation.

The complementary multistart report gives each parameter's standard deviation and range in estimation coordinates across solutions within $\Delta\mathrm{NLL} \leq 1$ of the best objective. If fewer than three starts qualify, it instead uses up to five best recorded starts, which can lie outside this band. At least two values are needed for a parameter's dispersion estimate. Up to 12 simplification candidates are ranked by the number of warning signs: a bound flag, scaled standard error at least one, high marginal correlation, participation in a flat direction, or multistart standard deviation at least 0.5. Reasons accompany each candidate. These signals motivate refitting or sensitivity analysis; they neither justify automatic removal nor establish structural identifiability.

Suggestions for sharing parameters. The parameter report also estimates the local cost of tying two parameters in estimation coordinates. For $\omega = e_i - e_j$ and fitted vector $\widehat{\theta}$, the quadratic approximation is

$$\widehat{\Delta\mathrm{NLL}} = \frac{(\omega^\top \widehat{\theta})^2}{2\omega^\top H_{\mathrm{floor}}^{-1}\omega}, \tag{10}$$

where H_{floor} is a numerically regularised version of the local information matrix H in Equation (9). For the eigendecomposition $H = Q\ \mathrm{diag}(\lambda_1, \ldots, \lambda_d)Q^\top$, it retains the eigenvectors but replaces eigenvalues below a positive threshold τ by that threshold:

$$H_{\mathrm{floor}} = Q\ \mathrm{diag}\big(\max(\lambda_1, \tau), \ldots, \max(\lambda_d, \tau)\big)Q^\top, \qquad \tau = \max\Big\{ d\epsilon \max_r |\lambda_r|, t_{\min} \Big\}.$$

Here d is the number of free parameters represented in H, ϵ is machine precision, and $t_{\min}$ is the smallest positive normalised floating-point number. Unlike the pseudoinverse used above, this inverse retains large contributions from nearly flat directions, allowing a parameter tie along such a direction to have a small estimated cost. The calculation is unavailable for material negative curvature, $\min_r \lambda_r < -\sqrt{\epsilon}\max_r |\lambda_r|$. By default, candidate pairs must share an inferred parameter role and estimation scale, and neither may be near a bound; metadata limitations and disabled filters are disclosed. Reports include the estimated fit cost, largest required movement in scaled coordinates, and predicted change in BIC: tying one parameter saves $\log n$, before accounting for the estimated fit cost. The detailed report defaults to the eight cheapest pairs, ordered by estimated cost and then movement, and gives total priced and score-improving pair counts and reasons for skipped pairs. Up to three score-improving suggestions accompany fit summaries automatically. These are proposals for an actual constrained refit: approximate costs are not additive across multiple ties, and an empty list does not certify that the model is parsimonious.

Optimisation, revision, and stopping diagnostics. The fit result gives the actual multistart count and a histogram of termination flags, distinguishing convergence criteria from budget or numerical exits. The per-start SQL table contains objective-ranked starts, objective values, elapsed times, and function, gradient, and Hessian evaluation counts; the trace table contains available iteration-wise objectives and times. Termination flags are not columns of the per-start table. `get_fitting_log` returns the last 200 lines by default, or all lines when requested, including convergence, gradient, and conditioning messages. Server timing separately accumulates compilation and fitting counts and wall times; these are not summed CPU time or total agent elapsed time.

Revision feedback includes validation errors, reconciliation warnings, and a nesting advisory when a child gains estimated parameters but worsens the fitting objective by more than 10^{-3}. Where detectable, it names a parent parameter no longer estimated or referenced less often. This is a prompt to check whether the edit preserved the parent as a special case and whether optimisation was adequate, not a symbolic test of nesting. Non-nested alternatives remain legitimate proposals. The outstanding advisory persists across rollback until a qualifying parameter-addition comparison clears it. Progress reports additionally expose the best version, recent improvement, non-improving streak, and stopping status; their availability to agents depends on role.

Residual flags influence stopping as well as diagnosis. At this revision, any sign, runs, or replicate lack-of-fit flag prevents the server's early adequate-plateau stop. The harness's explicit finish gate refuses replicate lack-of-fit flags for up to three consecutive attempts before relenting; sign and runs flags alone do not cause that refusal. Estimated-noise inflation or collapse and a current, non-stale rank-deficient information matrix each trigger similarly bounded refusals. Low-power and weak-pass qualifications are advisory, and unavailable tests do not block completion. Consequently, reaching a stopping condition is not a statistical certificate that the mechanism is adequate.

Unavailable, stored, and stale reports. Diagnostics expose availability and reasons for missing evidence. Parameter geometry uses the frozen fitted optimum and an already compiled objective; reading the report does not compile a new model. If local-information evaluation fails, multistart-only information can remain available. Stored reports survive restoration and restart, with a stale flag for a changed structure; a live report also warns when current parameter settings differ from those used for its fitted geometry. Replaying a stored report does not expand lists that were truncated when it was generated. Residual inspection may simulate an already compiled model when needed. Missing simulation rows, unavailable uncertainty estimates, and empty candidate lists must therefore be distinguished from successful checks with negative findings. Compilation and integration failures require numerical investigation before being interpreted as evidence against a mechanism.

B.8 Data access and durable provenance

The `query_data` interface uses DuckDB to query PEtab tables, simulated measurements, residuals, and optimization diagnostics (Raasveldt & Mühleisen, 2019). A version-history table supports comparisons across stored simulations. Each query receives newly created in-memory tables copied from the session data. SQL may modify these disposable tables, but changes are not written back to the PEtab problem. External file and network access are disabled and the database configuration is locked before executing the query. Results are bounded and paged; stable pagination requires an explicit ordering. A version without readable simulation records contributes no trajectory rows and must not be interpreted as a successful fit.

The provenance store keeps the append-only event stream in `provenance.jsonl`, separate from snapshots in `versions/vN/`. A snapshot contains model source and reconciled model text, `parameters.tsv`, metadata, and available simulation and optimization records. A stored pyPESTO result can accompany the structured summaries. This distinction preserves attempted changes as well as numerical outcomes and supports restoration without discarding earlier candidates. Run-level records additionally identify configuration, code revision, timing, and available token-use measurements. A dirty-tree digest can distinguish uncommitted source states, but cannot reconstruct an unavailable patch. These records make the process auditable; they do not guarantee exact replay of external language-model or knowledge-service responses.

B.9 Agent roles and access to biological knowledge

In the panel configuration, the modelling lead (prompts in Listings 114 and 115) authors revisions and delegates bounded analyses. The mathematician (Listing 116) examines structure and estimation behavior; the systems scientist (Listing 117) assesses the biological mechanism; and the signal engineer (Listing 118) queries measurements and residuals. Tool sets are assigned by role. In particular, detailed measurement and SQL access is delegated to analysts, whose summaries return to the lead, limiting repeated inclusion of large tool outputs in its context. The biological curator (Listing 119) receives curated entity information and observable descriptions, rather than the fitting tools or authority to edit the model.

Knowledge tools are discovered from configured services and filtered through a named allowlist. The default set covers identities, protein domains, interactions, pathways, and ontologies, including UniProt, Reactome, and OmniPath (The UniProt Consortium, 2025; Milacic et al., 2024; Türei et al., 2026). These resources are useful only for models that describe molecular biology. Literature retrieval was implemented but disabled in the reported experiments. This separates structured retrieval from free-text searches that may reveal a benchmark's source study. It does not establish the absence of leakage from other knowledge sources. Unavailable services are skipped rather than replaced by unrestricted tools, and the resolved tool set is recorded. Access to a tool therefore establishes a capability, not evidence that it was used in every run.

B.10 Budgets, stopping, and final candidate selection

Run limits distinguish agent steps, model-output limits, total elapsed time, and numerical work per fit. Graph-step limits do not directly bound calls made within delegated analyses. The multistart count is a run setting that individual permitted calls can override. Per-start time limits and the number of parallel workers determine the expected fit duration; transport and stall timeouts include additional compilation and execution allowances. Budget accounting must therefore distinguish one long fit from an unresponsive agent step. Stopping and finish checks also distinguish an agent's declared completion from the existence of a scoreable model. Their thresholds and enabled controls are configuration- and revision-dependent.

In the reported `gemot` runs, the judge ranks stored candidates by BIC. It scans finite scores in order and prefers the first candidate whose available noise diagnostics do not trigger its plausibility check. Missing reports are unassessable rather than evidence of plausible noise. If none passes, the best-scoring candidate remains available for reporting with the failed check. Restoration failure is reported and leaves the current model to be assessed. Held-out predictions are reported separately from the construction-success decision. These distinctions are essential when comparing archived summaries or reselecting checkpoints for a new analysis.

C SOFTWARE SETUP, EXECUTION, AND RUN RECORDS

This section describes the implementation at code revision `e0d6921` and gives commands for installing the software and exercising the modelling workflow. This is the version documented here, rather than a claim that every reported experiment used this revision. Historical cohorts must instead be associated with their recorded code, problem, prompt, and configuration versions. The examples below are starting points for new runs; reproducing a reported cohort additionally requires its resolved configuration and retained inputs and outputs.

C.1 INSTALLATION AND DEPENDENCIES

The distribution contains both `petab_mcp`, which implements the server, and `petab_harness`, which coordinates experiments. It requires Python 3.12 or newer. The numerical stack is a core dependency: model evaluation uses AMICI, pyPESTO, and Fides (Fröhlich et al., 2021; Schälte et al., 2023; Fröhlich & Sorger, 2022). A working C++ compiler, CMake, Ninja, and SWIG are needed because AMICI compiles generated model code. Starting from the root of a source checkout with the compiler available, a local environment can be prepared as follows:

```
python3.12 -m venv .venv
. .venv/bin/activate
python -m pip install --upgrade pip
python -m pip install cmake ninja swig
python -m pip install -e '.[harness]'
```

The `agent` extra contains the Deep Agents (LangChain, Inc., 2025) and LangChain dependencies; the `harness` extra additionally installs Phoenix tracing and Weights & Biases logging. The optional `explorer` extra supports the run browser, and `dev` supplies testing and linting dependencies. An editable installation does not freeze dependency resolution: the environment's resolved package versions and compiler information must be retained with an experiment. The repository also supplies a Linux Conda environment with the compiler and HDF5 development dependencies for cluster use.

C.2 A LOCAL AGENT RUN

For a local agent run on the Fujita benchmark, configure `OPENROUTER_API_KEY` in the execution environment, prepare the empirical problem (Appendix C.3), and select the model explicitly:

```
python examples/setup.py fujita
PETAB_MCP_CRITERION=bic PETAB_MCP_FIT_N_WORKERS=4 \
python examples/fujita/run.py \
    --driver deepagents --model glm-flash \
    --temperature 0.0 --n-starts 50 \
    --recursion-limit 250 --max-wall-seconds 3600
```

At the documented revision, `glm-flash` resolves to `z-ai/glm-5.3-flash`. The model registry also specifies provider routing, so the alias alone is insufficient to identify the service configuration across revisions. Phoenix tracing is enabled for every agent run; without a configured collector endpoint (`PHOENIX_COLLECTOR_ENDPOINT` or `OTEL_EXPORTER_OTLP_ENDPOINT`), the runner starts a local collector. Weights & Biases logging is separately enabled with `-wandb`.

Each invocation recreates the example's `work/` directory from its prepared `problem/` directory, launches a stateful MCP server, runs the selected driver, and writes `work/result.json`. The server uses HTTP transport to keep compiler output separate from protocol messages. Model compilation and server logs are written to `amici_build.log` and `server.log`. Concurrent repetitions of one example require separate example directories because the local working path is reused.

C.3 Preparing empirical problems and reference fits

Published problems are materialised from the Benchmark-Models-PEtab collection at revision `1763dfcd66d25bfd97492cfedf6a730472c383d0`. The setup script obtains this pinned collection and applies each example's declared configuration:

```
python examples/setup.py
python examples/setup.py fujita alkan raimundez
```

The first command prepares the complete empirical suite; the second prepares a subset. This step constructs the agent-facing problem and leaves the committed reference scores unchanged. After materialisation, an empirical problem is run through its own wrapper, for example `examples/fujita/run.py`, with the same runner options as above.

Reference fitting is an explicit additional operation:

```
python examples/setup.py --refit fujita
```

This regenerates the reference record using the published model and the example's declared adjustments. These include, where applicable, freeing parameters on both sides of the comparison, parameter overrides, and charges for fitted values represented by model literals. Reference multistart counts and per-start limits are declared in each example. A regenerated reference should be accompanied by convergence information; increasing the number of starts cannot compensate for truncating nearly every start before convergence. The cluster utilities `slurm/refit_references.sh` and `slurm/verify_refit.py` support reference submission and inspection of per-start outcomes. Reference regeneration is therefore a separate numerical experiment, not a routine consequence of downloading the benchmark.

C.4 Selection criteria and computational budgets

The reported experiments used BIC to guide search and final selection. The standalone server selects it with `-criterion bic`; the example runner inherits `PETAB_MCP_CRITERION=bic`. Other information criteria reported in the results are retrospective comparisons, not separate search arms.

The local runner defaults to 50 multistarts and a recursion limit of 250 graph steps. `-n-starts` sets the default fit budget; an individual model-edit call can request a different count. `-max-wall-seconds` requests a graceful stop at a subsequent step boundary, allowing a committed candidate to be judged. It is not a hard process deadline and does not replace a scheduler allocation.

Numerical budgets have separate controls. `PETAB_MCP_FIT_N_WORKERS` sets the worker limit, and `PETAB_MCP_FIT_PER_START_TIMEOUT` bounds individual optimisation starts, with a default of 180 seconds. Transport and stall timeouts account for the requested starts, parallel worker waves, and compilation allowance. The documented implementation sets AMICI's absolute tolerance to 10^{-10} unless `PETAB_MCP_FIT_AMICI_ATOL` overrides it. These implementation defaults should not be substituted for the settings recorded for historical fits or held-out rescoring.

C.5 Cluster execution and retained records

The supplied Slurm workflow uses cluster-specific partition names and resource allocations that require adaptation to the target system. `slurm/setup_env.sh` builds or updates `envs/petab-mcp/` inside a compute allocation. It installs the development, harness, and explorer extras. A shared Phoenix collector is started using `slurm/phoenix_ctl.sh` with the `start` argument before submitting agent sweeps.

`slurm/make_sweep.py` expands problems, models, temperatures, and repetitions into a uniquely named configuration table and submits one array task per combination. Its resource options include `-cpus-per-task`, `-mem`, `-time`, and `-throttle`. Unlike the local runner, it defaults to 100 multistarts and 500 graph steps. The resolved per-example count is raised to `AGENT_N_STARTS`, or to `REFIT_N_STARTS` when no agent count is declared, if this exceeds the requested count. Per-start limits similarly prefer `AGENT_PER_START_TIMEOUT` and otherwise use the reference limit. Reference and agent budgets can therefore differ, especially for large

Table 3: **Configuration of the scored agent runs**, by figure and arm. *Given*, the support available to the run as in Figure 3; *prompt set*, a hash over the run's complete prompt registry; *commit*, the code revision the run executed. A count after a value gives the number of runs with it.

Arm	Language model	Given	Prompt set	Commit	Runs
Figure 2: 90 runs over 18 benchmarks					
O1	GLM-5.3-Flash	curator + biology + gemot	`637a341`	`604449e` (80), `c1a851f` (5), `59cde21` (5)	90
Figure 3: 30 runs					
F2	GPT-6-Astra	biology + gemot	`971aaa6`	`5969dd5`	5
F3	GPT-6-Astra	biology + bare agent	`637a341`	`5969dd5`	5
O1	GLM-5.3-Flash	curator + biology + gemot	`637a341`	`59cde21`	5
O2	GLM-5.3-Flash	biology + gemot	`971aaa6`	`59cde21`	5
O3	GLM-5.3-Flash	biology + bare agent	`637a341`	`5969dd5`	5
O4	GLM-5.3-Flash	gemot	`9faaf29` (4), `ecfb848` (1)	`7d22712`	5
Figure 4: 20 runs over 4 benchmarks					
O4	GLM-5.3-Flash	gemot	`ecfb848`	`a56d77b`	20

models; the resolved configuration table records these allocations. The default agent deadline leaves 1,200 seconds within the allocation for judging and archiving. The evaluated configurations are listed in Appendix C.6.

Array tasks copy example files into private working directories, align fit parallelism with the allocation unless explicitly overridden, and collect result and log files with the combination identifier and exit status. At normal completion, the harness attempts to archive model versions, parameters, simulations, accepted-change provenance, and rejected-tool-call records under `examples/_runs_archive/`. The result records code and prompt versions, resolved ablations, and the trace identifier linking it to Phoenix. Archives omit large optimisation files such as `pypesto_result.hdf5`; these require separate retention for subsequent optimisation audits. A working-tree digest identifies differences but cannot reconstruct the patch. Consequently, reproducibility requires preserving the actual source changes, environment, prepared problems, reference records, configuration tables, and traces alongside the compact run archive.

C.6 Configurations of the scored runs

Tables 3–5 give the configuration of every agent run that Figures 2–4 score, read from each run's archived record and scheduler log rather than from the implementation defaults above; `paper/generated/run_configurations.csv` lists every run with every field. The harnesses carry the names used in Figure 3. The support a run was given is reported as it ran: the biology-blinded panel records the curator as enabled in its ablation state but seats none. The null-arms experiment of Figure 3 kept no scheduler logs, so its budget is the one its submission manifest records. No run set a reasoning effort, so every call ran at its model's default as the provider serves it: medium effort for GPT-6-Astra and max, the highest of its three levels, for GLM-5.3-Flash, with reasoning on for both (Table 5). The exception is a call whose response spent its whole completion budget on reasoning and returned nothing; the harness re-issued it at low effort.

D Evaluation cohorts, scoring, and numerical protocols

D.1 Evidence sources and version scope

The software description in Appendices A–C is anchored to source revision `e0d6921`. That revision is a description of the available implementation, not a claim that every reported run used the same code. The manuscript combines the run-level training export accompanying Figure 2, the expanded Fujita evaluation in Figure 3, and the additional validation systems in Figure 4. Their candidate selection, numerical settings, and denominators must be interpreted within each cohort.

Table 4: **Fit and wall budgets of the scored agent runs**, per benchmark as each run's scheduler log states them (for the null-arms experiment, which kept no logs, as its submission manifest records them); the Figure 3 arms share one budget. *Starts*, multistarts per fit; *per-start cap*, the optimiser's time limit per start; *wall budget*, the agent's deadline; *fit workers*, parallel optimisation workers.

Benchmark	Starts	Per-start cap (s)	Wall budget (h)	Fit workers
Figure 2				
perelson	100	60	3.67	100
crauste	1000	60	3.67	100
rahman	100	60	3.67	100
brannmark	100	240	3.67	100
boehm	100	60	3.67	100
armistead	100	60	3.67	100
fiedler	100	90	7.33	100
bruno	100	60	3.67	100
oliveira	100	60	7.33	100
sneyd	100	60	7.33	100
fujita	100	90	6.67	32
blasi	100	60	3.67	100
schwen	100	120	7.67	100
bachmann	200	2400	33.3	100
raimundez	100	2400	33.3	100
isensee	100	12000	167	100
alkan	100	2400	33.3	100
lucarelli	100	1200	33.3	100
Figure 3				
all arms	100	90	6.67	32
Figure 4				
perelson	100	60	3.67	100
crauste	1000	60	3.67	100
raimundez	100	2400	33.3	100
lucarelli	100	1200	33.3	100

Table 5: **Settings shared by the scored agent runs.** A count after a value gives the number of runs with it; — marks a setting a run does not record. *Reasoning effort requested*, the setting each run's requests carried; with none, the provider's default applies, as OpenRouter's model metadata listed it on the date shown. *Calls re-issued at low effort*, responses that spent the whole completion budget on reasoning and were retried.

Setting	Value
Sampling temperature	0.0
Agent step limit	500
Output tokens per call	131072
Reasoning effort requested	none (provider default)
Calls re-issued at low effort	0 (128), 1 (9), 2 (1), 3 (1), 4 (1)
Selection criterion	bic (138), — (2)
ΔBIC threshold	10.0 (138), — (2)
OmniPath snapshot	2026-08-11 (105), — (35)
Code-execution timeout (s)	— (130), 1344.0 (10)
Provider, GLM-5.3-Flash	z-ai (served by Z.AI)
Provider, GPT-6-Astra	openai (served by OpenAI)
Default reasoning, GLM-5.3-Flash	on, max effort (of low, high, max)
Default reasoning, GPT-6-Astra	on, medium effort (of low, medium, high, xhigh, max)
Defaults as listed by	OpenRouter model metadata, 2026-09-26

As noted in Appendix A, the use of multiple software revisions reflects time and computational cost constraints. Unless specified otherwise, revision differences did not materially affect model construction, evaluation, or the reported conclusions. We will rerun all experiments using a single software revision upon publication.

Table 6: Evidence provenance and run selection. Infrastructure-failed attempts were excluded when they prevented a usable run or result; interrupted runs with a scored model were retained. Failures to construct a model were retained in the run denominators. Source stamps describe export environments; per-run code revisions describe the executions.

Evidence	Scope	Provenance and selection
Training comparison	90 runs over 18 problems	Latest-started 5 runs per problem from the listed jobs, excluding infrastructure failures. An Isensee rerun replaces the earlier out-of-memory attempt; all selected runs have CPU accounting. Export source `d958571+dirty`, imported at manuscript revision `b4209e6`.
Fujita configurations	30 runs, 28 models	Initial environment-activation failures before agent execution and associated cancellations were excluded; replacement batches entered the analysis. All 30 audited runs remain in the denominator, including two without a model. Final panel head or delivered bare-agent problem; run revisions `59cde21` and `5969dd5`, export stamp `604449e+dirty`.
Additional validation, O1	20 runs over four systems	Separate transfer and fixed-parameter evaluations; compact records in `fujita_validation_deltas.csv`.

D.2 Cross-problem sweep summaries

The sweep summaries use the frozen run-level export `paper/generated/sweep_data.json`, with selection and exclusions described in Table 6. Runs started between 10 and 21 September 2026; the export retains job and archive identifiers, code revisions, fitting budgets, and candidate metrics. We used the recorded candidate summaries without reselecting model versions. The Fujita tasks and selected versions coincide with configuration O1 in Figure 3. The model-source annexes additionally contain the Fujita ablations (Appendix G). The biology-visible validation cohorts of Figure 4 are those systems' Figure 2 runs and are listed in Appendix H; the 20 biology-blinded validation models are not listed.

Per-problem medians use available finite values independently for each metric; these marginal medians need not describe the same candidate. The generated LaTeX macros used within this manuscript and the unrounded `sweep_medians.csv` use the same observations. Training condition counts include conditions represented in the measurement table and can differ from the number of supplied experimental-design rows. Likelihood differences are not pooled across problems with different measurement and noise models. Per-run differences satisfy $\Delta\mathrm{BIC} = 2\Delta\mathrm{NLL} + \Delta k \log N$, where N is the number of training measurements and k is the parameter count used by the scoring implementation.

Recorded multistart budgets (Table 4) are 1,000 for Crauste, 200 for Bachmann, and 100 for the remaining problems. Token and fit counts come from run summaries; token counts include cached inputs. Wall time is full-task elapsed time. CPU time is measured consumption summed over workers in Slurm accounting, including local compilation and fitting but excluding remote LLM inference. All 90 selected runs have performance scores, token usage, and CPU accounting. The smaller/larger usage summaries in Section 2.2 pool all 65 and 25 runs, respectively, rather than problem-level medians. Missing fit counts for two Lucarelli runs are omitted from fit-count summaries only. Export provenance and numerical consistency checks are recorded in `paper/drafted/sweep_provenance.md`.

D.3 Topological comparisons with the published models

We compared the selected candidate from each of the 90 sweep runs with its published reference. The scores in `paper/generated/sweep_data.json` were imported at manuscript revision `b4209e6`, with export stamp `d958571+dirty`. The definitions below follow the method description accompanying the earlier `51bd3d2` package; implementation inspection and the matched source examples refer to producer revision `5375241`. Current cohort selection, including the re-

placement Isensee run, is described in Table 6. These comparisons concern model structure, not the correctness of its biological interpretation.

Weighted graphs of interventions and readouts. The topology analysis reported in Section 2.3 uses weighted graphs to compare connectivity and latent depth. Their nodes represent interventions and readouts, termed intervention and readout anchors. We first extract a directed, unsigned graph of the underlying state-variable influences. A reactant, modifier, or species read by a reaction's rate law points to the species that the reaction changes. Rate rules contribute influences from their right-hand-side dependencies to the integrated variable. Assignment dependencies are resolved except at retained anchors; self-edges and product-only influences are omitted. Signs, parameter values and stoichiometric magnitudes are not compared, and conservation laws are not reduced.

Each shipped observable formula names its readout, so observables that read the same readout form one readout anchor, and the species underlying each readout are aligned between models; the shipped condition table identifies interventions. A declared intervention enters the graph as the species whose initial value it sets, as the assignment-rule variable it feeds, or as the column itself where it appears only in a rate law. In the latter two cases the symbol is retained as a graph node rather than resolved into its own definition, so that its downstream influences are retained. Routes *into* exogenous interventions are not scored. Fifty-nine of the 90 candidates share at least one intervention anchor with their published reference. One further candidate has a declared intervention without a shared anchor; matching its remaining routes does not establish recovery of the intervention response.

Connectivity and latent depth. An edge in the weighted graph records a connection between an ordered pair of anchors whose underlying route does not pass through a third anchor's species. Its weight is the shortest such route's hop count, which quantifies latent depth between the anchors. Let $d_c(A, B)$ and $d_r(A, B)$ be the candidate and reference hop counts where a route exists, and let $\mathcal{R}_c$ and $\mathcal{R}_r$ denote the sets of reachable ordered pairs.

Pairs present in only one model are grouped by their initial route segment to reduce repeated charges for a shared branch reaching multiple readouts. Write $\Gamma_m(\mathcal{P})$ for the partition of a pair set $\mathcal{P}$ induced by the source anchor and the first node of model m's selected shortest route; directly adjacent endpoint pairs form their own groups. These groups summarise structural differences, not independent biological hypotheses. The discrepancy is

$$D = \sum_{(A,B)\in\mathcal{R}_c\cap\mathcal{R}_r} |d_c(A,B) - d_r(A,B)| + \sum_{\pi\in\Gamma_c(\mathcal{R}_c\setminus\mathcal{R}_r)} \Big(1 + \min_{(A,B)\in\pi} d_c(A,B)\Big) + \sum_{\pi\in\Gamma_r(\mathcal{R}_r\setminus\mathcal{R}_c)} \Big(1 + \min_{(A,B)\in\pi} d_r(A,B)\Big),$$

normalised by $\sum_{\pi\in\Gamma_r(\mathcal{R}_r)}\big(1 + \min_{(A,B)\in\pi} d_r(A,B)\big)$. This is an exact comparison of the aligned anchor-route summaries, not an exact edit distance between the full reaction networks. Shared species can give zero-hop routes; the leading unit ensures that adding or removing such a route still has a cost.

We separate the score into added connections, increased latent depth on shared routes, deleted connections, and reduced latent depth on shared routes. Depth denotes the shortest route's hop count, rather than a count of uniquely added or removed latent species: an intermediate can serve multiple routes, and readout mappings affect which species delimit them. The sum of these four normalised components equals the reported total, up to rounding.

A normalised comparison is available for 75 of the 90 candidates, across 15 problems. Rahman has only one readout anchor and no intervention anchors. Oliveira has two readout anchors, cumulative cases and deaths, but both are terminal readouts with no directed route between them and no intervention anchors. Their reference route summaries therefore have zero normalising denominators; these comparisons are undefined and excluded rather than counted as matches.

Perelson also has a single readout, viral load, and its published reference encodes treatment implicitly rather than through an explicit drug-input variable. The current export has no intervention anchor for Perelson, so all five comparisons have zero normalising denominators and are undefined.

Of the 75 defined comparisons, 49 differ in connectivity and seven only in latent depth; 19 match both summaries. Added connections occur in 38 candidates and deleted connections in 41, with overlap. All 25 candidates for the larger problems add connections, although 23 use fewer states, one the same number, and one more than its reference. In contrast, the nonzero Fiedler and Sneyd comparisons change only latent depth. These distinctions are concealed by the total. Figure 2 and Table 7 therefore report the components separately. The normalised total is not bounded by one: six candidates exceed one (maximum 1.36), and longer shared routes alone can produce such a score without adding any connection.

Table 7: Connectivity and latent depth in the training sweep. Each problem has five selected candidates. Δs is the median candidate-minus-reference state count; "Changed" counts candidates whose connectivity or latent depth differs from the reference among defined comparisons. The last four columns give median weighted contributions from added connections, increased latent depth, deleted connections and reduced latent depth, using the reference-summary denominator defined above. They are not fractions of recovered biological interactions. Separate component medians need not sum to the median total. A dash denotes an undefined normalised comparison.

			Added		Removed	
Problem	Δs	Changed	Connections	Depth	Connections	Depth
Perelson	0	–	–	–	–	–
Crauste	1	4/5	0	0	0	0
Rahman	0	–	–	–	–	–
Brannmark	-4	5/5	0	0	0.38	0.12
Boehm	-1	3/5	0	0	0	0
Armistead	0	5/5	0.07	0	0.6	0
Fiedler	1	4/5	0	0.11	0	0
Bruno	0	0/5	0	0	0	0
Oliveira	2	–	–	–	–	–
Sneyd	-1	2/5	0	0	0	0
Fujita	2	5/5	0	0.15	0.46	0.08
Blasi	0	2/5	0	0	0	0
Schwen	-3	1/5	0	0	0	0
Bachmann	-10	5/5	0.13	0.01	0.3	0.35
Raimundez	-10	5/5	0.45	0.06	0.36	0.09
Isensee	-15	5/5	0.15	0.15	0	0.15
Alkan	-19	5/5	0.3	0.04	0.57	0
Lucarelli	-6	5/5	0.01	0.02	0.07	0.01

Further structural examples. Matching connectivity and latent depth does not imply that the underlying models are identical. For example, Schwen candidate 3 (Listing 86) uses eight states instead of the reference's eleven while retaining both connectivity and latent depth, with training $\Delta\mathrm{BIC} = -131.78$. Four Schwen candidates match both connectivity and latent depth; this includes candidate 5 (Listing 88), whose declared insulin-dose intervention enters through `E = init_Ins` and binding reactions `bind1` and `bind2`, although `init_Ins` has no shared anchor in the exported comparison.

Raimundez candidate 1 (Listing 94) adds MAPK-dependent inhibition of EGFR activation. Reaction `egfr_actL` converts `E_b` to `E_p` at rate `a_E*E_b/(1 + f_eEb*M_p)`, with `M_p` identified by `readout_pMAPK := M_p`. The coefficient `f_eEb` is unset in the listing but equals 39.91 in the saved parameter table, `paper/model_sources/benchmark/raimundez_1.parameters.tsv`. This positive coefficient makes the activation rate decrease with MAPK; the reference influence graph has no MAPK-to-EGFR route. This candidate has four fewer states and training $\Delta\mathrm{BIC} = -696.68$. The signed interpretation comes from the rate law and fitted coefficient, not from the unsigned topology score.

Relation to Fujita topology recovery. The recovery counts in Figure 3C (Appendix D.5) test whether each model contains the three forward connections EGF→pEGFR→pAkt→pS6, with each route excluding other intervention or readout anchors. Additional forward connections and different latent depths are permitted; backward connections are not assessed. The sweep instead scores added

and removed connections and changes in latent depth across all ordered anchor pairs, including feedback between readouts but excluding routes into interventions. Both analyses use EGF as the intervention anchor for the five shared Fujita candidates.

The numerical path extraction also differs from the sweep and the equation-based SBGN maps in Appendix F. The Fujita extractor treats products as sources of influence on other reaction participants even when the rate law does not read them. For example, reference reaction `v3_reaction_3`, `pEGFR_Akt`→pEGFR+pAkt, creates a pAkt→pEGFR link solely because the two species are co-products. The sweep omits this link unless supported by a reactant, modifier or rate-law dependency. This rule can alter connectivity and shortest path lengths. Consequently, recovery in Figure 3C does not imply zero discrepancy in Figure 2; neither establishes biological causality or dynamical equivalence.

Mechanistic interpretation of added connections and latent intermediates. We inspected the equations and saved parameter tables for six systems, including the examples discussed in Section 2.3. Table 8 identifies representative candidates with added connections or increased latent depth. These examples illustrate different interpretations of the scores, rather than estimating how often agents recover supported mechanisms. Signs below follow the fitted equations, not the unsigned graphs. The literature assessment concerns each specific proposal, not validation of the whole candidate. Candidate reaction and assignment-rule names below identify the corresponding entries in the linked listings. Sources are listed in Appendices H and G; the Fujita example belongs to the shared training-sweep cohort, without using its held-out performance for selection.

Table 8: Inspected examples from the training sweep. Added and Depth are the normalised contributions from added connections and increased latent depth, rounded to three decimals; other components may also be nonzero. All selected examples improve training BIC. Listings identify the archived models and retain their task and version provenance.

Model	Listing	ΔBIC	Added	Depth
Crauste 2	35	−23.08	0.250	0
Brannmark 2	45	−21.93	0.250	0
Armistead 5	58	−28.86	0.067	0
Sneyd 5	78	−131.14	0	0.250
Fiedler 5	63	−34.66	0	0.111
Fujita O1d	4	−320.72	0.231	0.154

Crauste (reference: Figure 34): pathogen-dependent memory differentiation. Candidate 2 (Listing 35) makes late-effector-to-memory conversion pathogen-dependent through reaction `e2_mem`, with flux $v_{L\to M} = c_{\text{fate}}(1-a_P)L = c_{\text{fate}}\alpha_{\text{act}}L/(\alpha_{\text{act}}+P)$ instead of the reference's constant multiple of L. Assignment rule `a_P` defines $a_P = P/(\alpha_{\text{act}}+P)$, with $P =$ `Pathogen`, $L =$ `LateEffector`, $\alpha_{\text{act}} =$ `alpha_act` and $c_{\text{fate}} =$ `c_fate`. The saved fit gives $\alpha_{\text{act}} = 1.2332$ and $c_{\text{fate}} = 0.09673$. Early effectors clear pathogen through `p_clear`, so this creates an early-effector–pathogen–memory route that avoids the measured late-effector anchor. The hypothesis is release of suppression as pathogen declines, not a direct early-effector-to-memory lineage. Candidate 3 (Listing 36) uses the same qualitative gate in `l_to_m`, but also sends 9.6% of modelled naive-cell activation directly to late effectors through `n_to_l`, alongside the early-effector route `n_act`; that shortcut has less support from the reference study's predominantly linear ontogeny (Crauste et al., 2017). Vaccinia experiments in Sarkar et al. (2008) separated cognate antigen from infection-associated stimuli and found that shortening late antigenic stimulation favoured memory-associated phenotype and function. This supports the gate's sign, not its exact kinetics or a larger final memory population. Moreover, antigen can persist after infectious-virus clearance (Tamburini et al., 2014); the single pathogen variable is an approximation to antigenic stimulation.

Brannmark (reference: Figure 39): a shared response approximation. Candidate 2 (Listing 45) assigns $b_{\text{IR}} = R_p$ and $b_{\text{IRS1}} = 1 + K_d R_p$ in `readout_IR_P` and `readout_IRS1_P`, with fitted $K_d = 16.0742$, before assay-specific observation scaling. Candidate 4 (Listing 47) uses the same readout names, with R_p and $R_p + r_0$ via `IRS1p_lin` (R_p is `Rp` and `IRp`, respectively). Both therefore force the two biological signals to share a temporal shape, rather than supplying separate IRS1 dynamics. Insulin reaches their shared state directly through `receptor_activation` and `ir_activation`, respectively, creating an additional insulin–IRS1 anchor route; in the reference,

that route passes through the IR readout anchor. This is a testable reduction of the readout mapping, not receptor-independent IRS1 activation. The primary adipocyte study supports receptor trafficking together with downstream feedback (Brännmark et al., 2010), but does not establish affine equivalence of the two signals. Candidate 2 retains trafficking (`receptor_internalization` and `receptor_recycling`) while omitting downstream feedback; candidate 4 retains a delayed inhibitory variable (`x2_prod`/`x2_deg`, acting through `ir_dephos`) but no explicit internalisation. Such differences make the published trafficking interventions relevant discriminating evidence.

Armistead (reference: Figure 35): genotype-dependent ceramide supply or clearance. All five candidates introduce a direct genotype effect on a ceramide rate. The reference already connects genotype to ceramide through measured sphinganine; the added anchor route bypasses this intermediate. In candidate 5 (Listing 58), the reaction named `s_salvage` adds $k_{cs}(1 + g_{\text{on}}S_{\text{on}})$ independently of sphinganine, with $k_{cs} = 0.60754$ and $g_{\text{on}} = -0.48799$ (parameter `g`). Because S_{on} is one in wild type and zero in the Hai1a-deletion mutant, this source is about 49% lower in wild type. Candidate 1 (Listing 54) instead changes clearance through `cer_exit`; together with the source `csl_to_cer`, this gives $\dot{C} = 0.03145 - 0.22143 S_{\text{on}} C$ for $C =$ `Cer`. These are distinct kinetic hypotheses despite sharing the added unsigned connection. In the original zebrafish study, serine palmitoyltransferase inhibition and ceramide-synthase knockdown supported de novo synthesis, whereas sphingomyelinase inhibition did not alleviate apoptosis (Armistead et al., 2024). This evidence favours de novo supply and does not validate a sphinganine-independent source or altered clearance. The unassigned source may instead absorb omitted processes in the restricted training data.

Sneyd (reference: Figure 36): priming before pore opening. Candidate 5 (Listing 78) contains the route $\text{IP}_3 \rightarrow g_{\text{primed}} \rightarrow g_{\text{open}}$ through `J_prime` and `J_gate`, with states `gprimed` and `gopen`. Their fluxes are $k_{pr} f_{\text{IP3}}(1 - g_{\text{primed}} - g_{\text{open}})$ and $k_{pr} g_{\text{primed}}$, respectively, with fitted $k_{pr} = 23.87$ (`kpr`) and $f_{\text{IP3}} = \text{IP}_3^{h_I}/(K^{h_I} + \text{IP}_3^{h_I})$ (assignment rule `fip3`). The primed state is absent from `readout_open_probability`, giving an explicit nonconducting intermediate. The reference input directly influences a readout-contributing gating state, so this adds one hop while using fewer declared states overall. Partial-agonist and mutational experiments support conformational coupling between ligand binding and pore opening (Rossi et al., 2009), although principally in $\text{IP}_3\text{R1}$ rather than the benchmark's type-2 receptor. They do not identify this fitted gate. Ligand-step experiments also constrain strict binding-order interpretations (Mak et al., 2007). The reference already describes dynamic activation and eliminates fast intermediates (Sneyd & Dufour, 2002); added depth resolves these dynamics differently, rather than establishing a previously unknown receptor state.

Fiedler (reference: Figure 37): feedback with a finite response time. Candidate 5 (Listing 63) replaces direct ERK inhibition of RAF activation by an inhibitory state w: `fb_charge` and `fb_decay` yield $\dot{w} = k_a(\text{pERK}^2 - w)$, while `raf_act` makes RAF activation proportional to $1/w$. The fitted $k_a = 0.32935$ (`k_adapt`) gives a relaxation time of approximately 3.04 model-time units. Together with `mek_phos`, the return route becomes pERK$\rightarrow w \rightarrow$pRAF$\rightarrow$pMEK, one hop longer than the reference. The hypothesis is that RAF sensitivity depends on recent ERK activity, rather than only its instantaneous abundance. ERK-dependent inhibitory RAF phosphorylation and subsequent recovery provide an experimental precedent (Dougherty et al., 2005). They do not identify w as a distinct protein, validate reciprocal inhibition, or establish its timescale in the benchmark's HeLa experiments (Fiedler et al., 2016). Positive ERK–RAF feedback has also been reported under different cellular and stimulation conditions (Balan et al., 2006), further limiting identification of w with a specific RAF phosphoform. Candidate 4 (Listing 62) also introduces a feedback mediator `Xinh` through `feedback_recruitment` and `feedback_decay`, with inhibition encoded by assignment rule `Ffb`. Part of its additional depth comes from moving Sorafenib's action from reference MEK phosphorylation (`v3_v_2`) to `raf_activation` through assignment rule `I_sor`, which does not enter `mek_phosphorylation`, showing how pharmacological abstraction can affect the score.

Fujita (reference: Figure 43): resolving mTORC1 and S6K. Candidate O1d (Listing 4) inserts the three-hop route pAkt$\rightarrow M \rightarrow$pS6K$\rightarrow$pS6 through `mtorc1_act`, `s6k_act` and `s6_phos`; the first reaction identifies M as mTORC1 activity. The first two activation rates are $V_M \text{pAkt}(1 - M)/(K_M + \text{pAkt})$ and $k_M M(1 - \text{pS6K})$; the saved coefficients are positive ($V_M = 1.17 \times 10^{-7}$, $K_M = 0.04119$ and $k_M = 1.3212$; `Vmax_M`, `Km_M` and `k_M`). Akt regulation of TSC2 and mTOR-

dependent S6K activation support this pathway ordering (Inoki et al., 2002; Brown et al., 1995), although the candidate still lumps upstream regulation and does not establish these kinetics in EGF-stimulated PC12 cells. The reference Akt readout includes the Akt–S6 complex, which reaches pS6 in one hop through `v6_reaction_6`: the two-hop increment therefore reflects both an explicit cascade and a changed readout mapping. O1c (Listing 3) goes further, adding serial `S6K2`/`S6K3` stages through `s6k2_act`/`s6k3_act`, without established molecular identities. Such stages can implement filtering consistent with the original study's functional observation (Fujita et al., 2010), but their names and depth do not demonstrate recovery of additional biological actors.

D.4 Training fit per model version

Figures 5 and 6 follow every scored run through its model versions. At each version a run contributes the best training ΔBIC it had fitted so far, carried forward once it stops; a panel shows the median over its runs as a line and the minimum–maximum range as a band. These curves include all fitted versions, without applying the final noise-plausibility check. The judge instead scans versions in BIC order and can skip those whose fitted noise is flagged as implausibly inflated or collapsed (Appendix B.10). It therefore can select a higher-BIC model; if every version is flagged, it falls back to the lowest-BIC version. The delivered model differs from the raw minimum in 9 of the 90 Figure 2 runs (four of the five on Alkan). The trajectory export does not retain individual selection reasons, so it does not establish that this filter explains all nine discrepancies. Each run's delivered ΔBIC, the value its figure plots, is therefore marked where the run ended. The two bare-agent arms of Figure 3 (O3, F3) archive only the model they delivered, so they are shown as their delivered end state.

D.5 Fujita configurations and fixed-parameter prediction

Figure 3 combines three audited Fujita experiments, comprising 30 runs across 6 configurations. The first experiment used revision `59cde21`; the later bare-agent and Astra experiment is associated with `5969dd5` in its submission manifest, and the biology-blinded re-run used `7d22712`, which appends the stimulus protocol to the blinded task (described below). We verified that the other differences between these three run-code revisions had no impact on agent behaviour. These run revisions differ from the export stamp `604449e+dirty`, preserved in manuscript commit `16a2651`. Construction used 144 measurements from six constant-dose conditions; fixed-parameter evaluation used 276 withheld measurements from six pulse and four ramp conditions (Fujita et al., 2010). The published dynamical model was not supplied to the agents. The experiment records specify 32 CPUs, 100 fitting starts, and a 90-second per-start limit.

Table 9: Fujita configurations and denominators. All configurations have five audited runs. "Topology" counts models containing every required forward readout connection among delivered models; "Improved" counts complete held-out scores below the stored reference.

Code	Configuration	Models	Complete	Improved	Topology
O1	GLM panel, curator on	5	5	4	5/5
O2	GLM panel, curator off	5	4	3	5/5
O3	GLM bare agent	3	3	2	3/3
O4	GLM panel, biology blinded	5	4	3	1/5
F2	Astra panel, curator off	5	5	5	5/5
F3	Astra bare agent	5	5	5	5/5

We ran these experiments with `z-ai/glm-5.3-flash` and `openai/gpt-6-astra`. The figure uses C and F for these models, followed by a configuration number and a run letter. Display names differ from archive labels: O1a–O1e map to B1–B5, O2a–O2e to O1–C5, O3a–O3e to G1–G5, O4a–O4e to X1–X5, F2a–F2e to P1–P5, and F3a–F3e to N1–N5. O3a and O3e delivered no fittable model. Panel runs contribute their recorded final `head_version`; bare-agent runs contribute the delivered PEtab problem, represented by the sentinel version -1 in the CSV. Neither selection uses held-out performance.

O1 includes the biocurator and its external knowledge tools. O2 and F2 remove that role, its tools, and the lead's curator guidance, while retaining biological identifiers, assay descriptions, and the

three modelling analysts. O3 and F3 use a stock agent with file tools and a sandboxed shell to author and fit a PEtab problem directly. They retain curated biological knowledge tools and scenario information, but lack the panel's MCP modelling operations, version store, specialist analysts, and finish gate. The comparison therefore changes a collection of modelling and coordination facilities rather than isolating knowledge retrieval. The configurations are not a complete factorial design: the Astra curator-on (F1) and biology-blinded (F4) arms were not run because of cost.

Biology-visible configurations receive the time-dependent EGF rule and guidance to use EGF in receptor binding. Biology blinding anonymises identifiers, removes descriptive metadata and replaces the biological task with numerical instructions. The current implementation (revision `7d22712`) preserves the experimental input protocol: it reads the rules from `inputs.tsv`, translates stimulus and condition-input identifiers to anonymous aliases, and appends them to the blinded task. For Fujita, the rule specifies

$$u(t) = \begin{cases} u_0 + \rho t, & t \le t_{\text{off}}, \\ 0, & t > t_{\text{off}}, \end{cases}$$

where u_0, ρ and t_{off} are fixed condition inputs. The prompt requires the agent to implement the assignment rule exactly and use $u(t)$ wherever the stimulus enters the model, rather than substituting its parameter columns or estimating them. This supplies the prescribed step, pulse and ramp waveform definitions without revealing the stimulus's biological identity or receptor-binding role. Rules may reference only their declared condition-input columns, time and mathematical functions; both preparation and prompt assembly validate the allowed symbols. The driver constructs the task from this validated input, discarding caller-supplied scenario prose. A private identifier mapping supports evaluation. Blinding does not remove pretrained biological knowledge, and enabling external tools does not establish which sources a particular run consulted.

We designed both pipelines to avoid an explicit incentive to optimise held-out performance. The supplied objectives concern training fit and BIC (Listings 153, 120 and 124); held-out evaluation is neither announced as an objective nor returned as construction feedback. The panel tool allowlist excludes the held-out evaluator. The bare agent sees a staged problem in a sandbox, with reference files withheld; its scoring server is started only after delivery. In both pipelines the judge evaluates held-out data after construction. We cannot exclude that an agent inferred later evaluation from the supplied experimental design.

Scoring and numerical availability. The archived parameters, including observation scales, remain fixed for Fujita evaluation (Appendix E.1). We rescored all delivered Fujita models with a script that varied solver tolerances and integration step sizes; the results were consistent across these settings. O2c and O4b lack a complete score over the ten conditions. Together with the two runs that delivered no model, they leave 26 complete held-out scores among 30 audited runs. An incomplete integration receives no joint likelihood over a reduced subset of measurements. It counts as predicting worse than the reference: it ranks worst in the held-out medians and ranges, and counts against its configuration in Figure 3D. A model without a held-out score still contributes its training ΔBIC to the training summary, whereas a no-model run contributes to neither.

All differences are candidate minus reference. The reference offsets for $\Delta\mathrm{NLL}$ are -66.5807726637496 for training and 4156.3460971713175 for held-out evaluation. Training $\Delta\mathrm{BIC}$ uses the committed reference with $k = 20$ and $N = 144$; the export corrects only its reference offset, with every recorded adjustment equal to zero. The held-out reference uses a stored score.

Readout topology and structural features. For each forward intervention/readout pair, the topology audit searches for a directed influence-graph path that does not pass through the species of another intervention or readout. The reference contains three adjacent connections under this rule. Inclusion requires all three, permitting additional connections and different latent-path lengths. Of 28 models, 24 satisfy this criterion; the two runs without a model are reported separately. This unsigned comparison does not test edge signs, reaction identities, rate laws, backward feedback, or the complete latent mechanism, and does not reduce conservation laws. Panel B illustrates the reference and two candidates selected for having the lowest held-out $\Delta\mathrm{NLL}$. That illustrative selection is retrospective. Its arcs all use one activity-flow terminator and are not sign-tested; they are separate from the unsigned recovery criterion used in panel C.

Depths count shortest directed influence-graph hops between implemented intervention/readout anchors. Assignment dependencies are expanded and readout rules add no hop. The blinded intervention anchor is the anonymous ligand/dose species mapped from `EGF_0`; other models use their implemented EGF stimulus. These are propagation proxies, not exclusively counts of unobserved intermediates. Reaction count includes all SBML reactions. State count includes dynamical species and rate-rule variables before conservation reduction, excluding constant, boundary, and assignment-defined species. Charged parameter count includes estimated observation scales and charged hardcoded literals.

Predictive success and rank association. Panel D counts models with held-out $\Delta\mathrm{NLL} < 0$ among those with complete scores, irrespective of topology. The same complete-score population underlies the Spearman correlations between training $\Delta\mathrm{BIC}$ and held-out $\Delta\mathrm{NLL}$ reported in Section 2.4: 16 GLM models across O1–O4 and ten Astra models across F2–F3. O3a and O3e produced no model; O2c and O4b lack complete held-out scores. We correlate the ranks of the two quantities within each language-model sample, using average ranks for ties, without covariate adjustment or topology filtering. The resulting coefficients are $\rho = -0.14$ for GLM and $\rho = 0.30$ for Astra. These pooled-configuration associations are descriptive and do not establish causal effects of model complexity.

Cost accounting. Panel E divides each configuration's total expenditure estimate across all five audited runs by its number of fittable delivered models. Runs with no model are charged, and models with incomplete held-out integration remain in the denominator. Both models use their recorded usage amount, so the language-model component is the provider's invoiced spend with no list-price correction applied.

The CPU component values Slurm `TotalCPU`, measured in core-seconds, at the exported `c7i.8xlarge` Linux on-demand rate in `us-east-1`: $1.428 per 32-vCPU hour, quoted 12 September 2026. It is computed as

$$C_{\mathrm{CPU}} = \frac{\mathrm{TotalCPU}}{3600 \times 32}\, 1.428\ \mathrm{USD}. \tag{11}$$

This prorates actual CPU use; it does not price an allocated instance for the full elapsed time or measure a cloud invoice. Storage, data transfer, and reservation or spot discounts are excluded. Under this convention the 30 Fujita runs total $130. The separate experiments in Figure 4 are outside this cost total.

Held-out predictions per arm. Figures 7–12 draw every model of Figure 3 on the two held-out protocols, the one-minute EGF pulse at six doses and the EGF ramp at four rates, at its stored parameters and the `stricter` solver setting. The first column of each figure is the published model at the collection's nominal parameters; every other column carries the training $\Delta\mathrm{BIC}$ and held-out $\Delta\mathrm{NLL}$ that Figure 3 plots for that model. O4b integrates on no held-out condition, and O4e, which reads the condition inputs directly instead of implementing the supplied stimulus rule, predicts no response to the ramps; the other three biology-blinded models respond to both protocols.

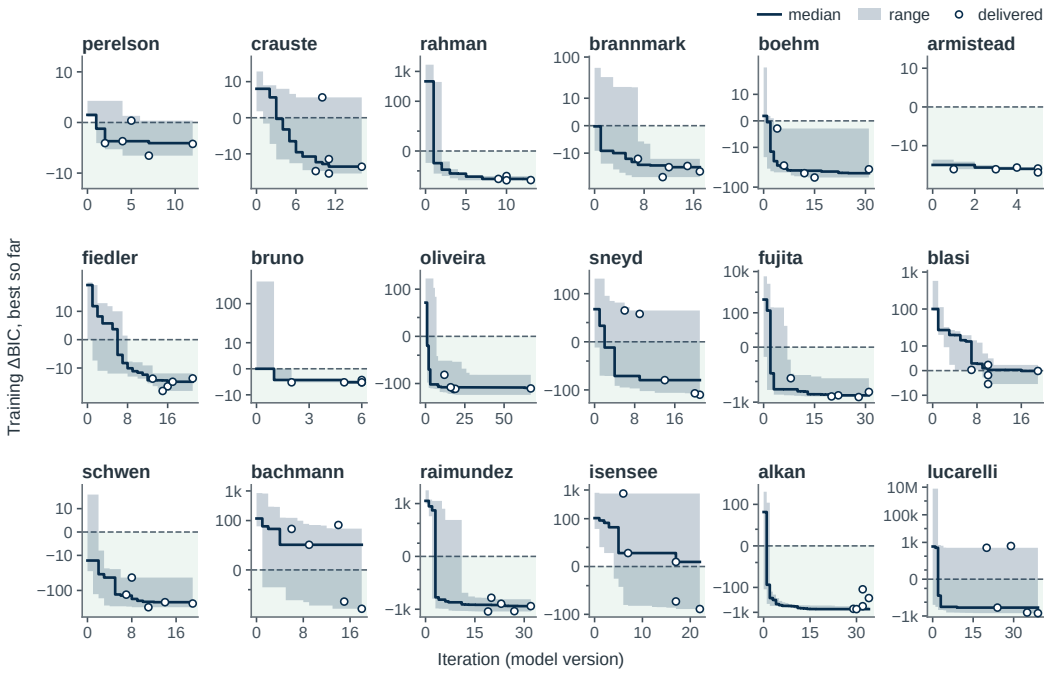

Figure 5: **Training ΔBIC per model version, Figure 2 runs.** One panel per benchmark, five runs each, on its own scale. Line, median best-so-far training ΔBIC against the committed reference; band, range over runs; open marks, each run's delivered model at the version it ended on. Symmetric-log axis, linear within ± 10; the shaded region is better than the reference.

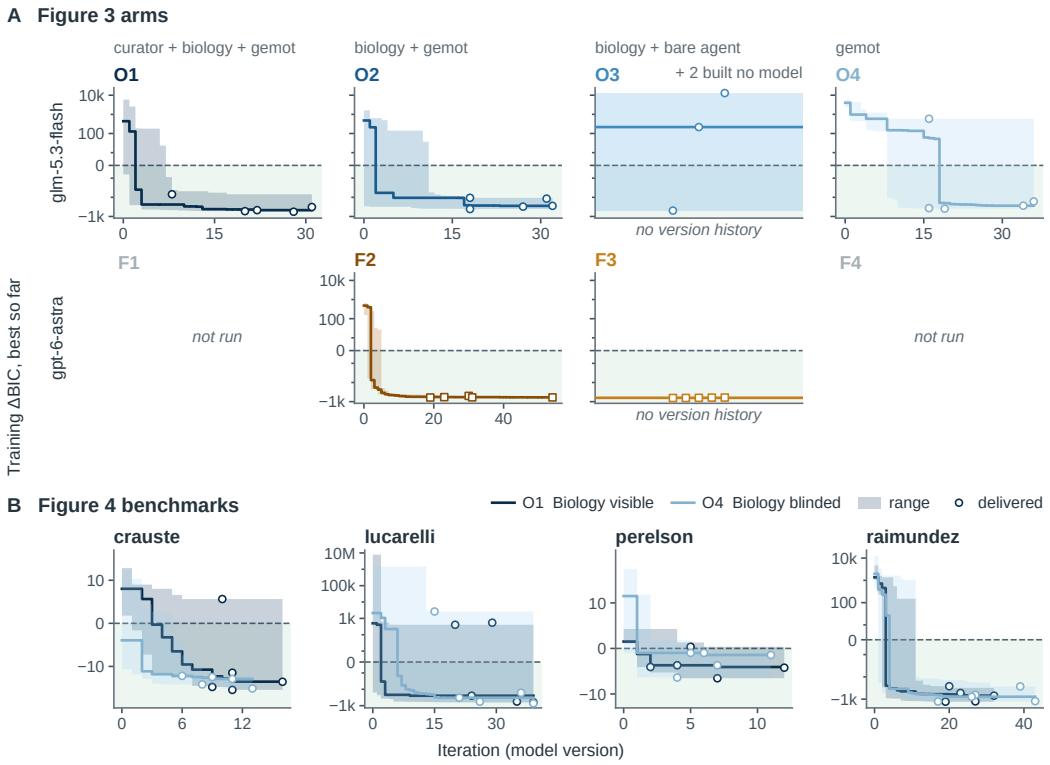

Figure 6: **Training ΔBIC per model version, Figures 3 and 4.** (A) The Fujita arms by language model (rows) and support rung (columns, in Figure 3's colours); O3 and F3 keep no version history and show their delivered models only; F1 and F4 were not run. (B) The four Figure 4 systems, biology-visible (O1) and biology-blinded (O4) runs overlaid. Encoding as in Figure 5.

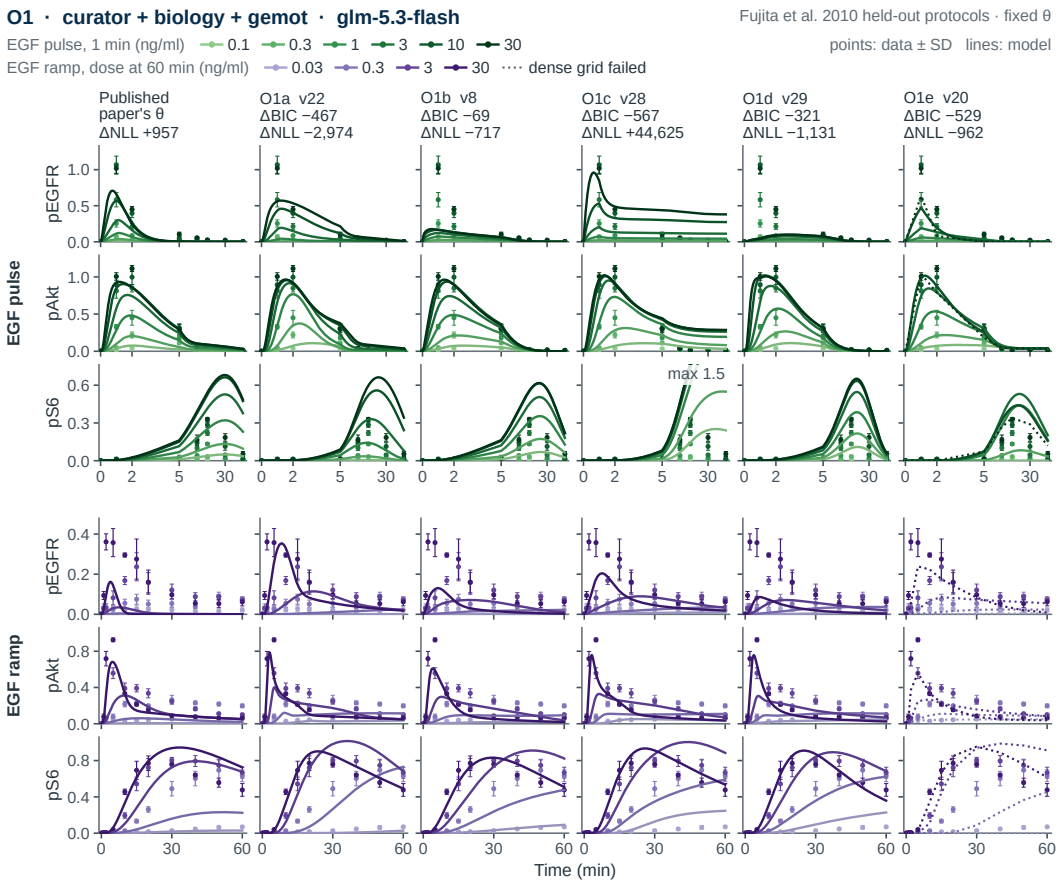

Figure 7: **Held-out Fujita predictions, O1** (curator, biology and gemot; GLM-5.3-Flash). Rows, the pulse block then the ramp block, three observables each; columns, the published model at its nominal parameters, then O1a–O1e. Points, data ± SD; lines, each model at its stored parameters; one sequential scale per protocol, lightest for the weakest stimulus. Rows share a scale; a prediction leaving it is clipped with its maximum printed. Dotted, a condition that integrated at the measured times but not on the drawing grid.

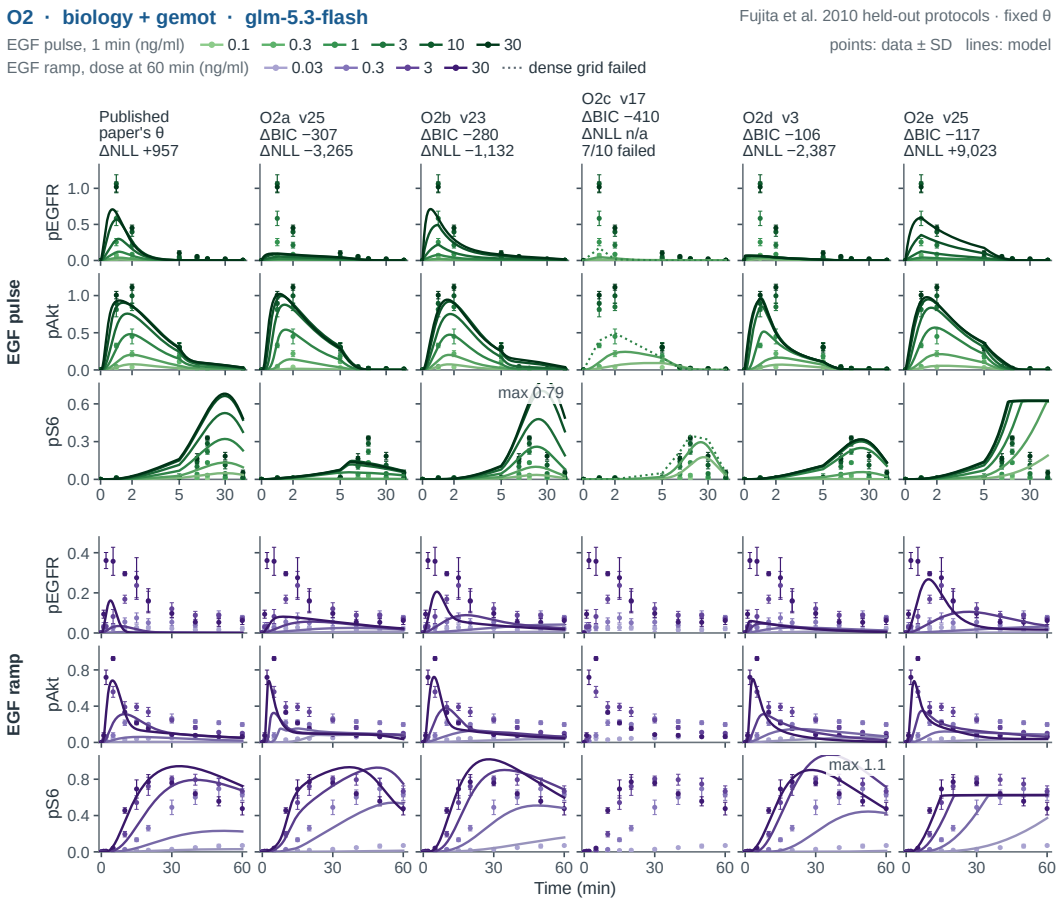

Figure 8: **Held-out Fujita predictions, O2** (biology and gemot; GLM-5.3-Flash). As Figure 7; columns, the published model, then O2a–O2e.

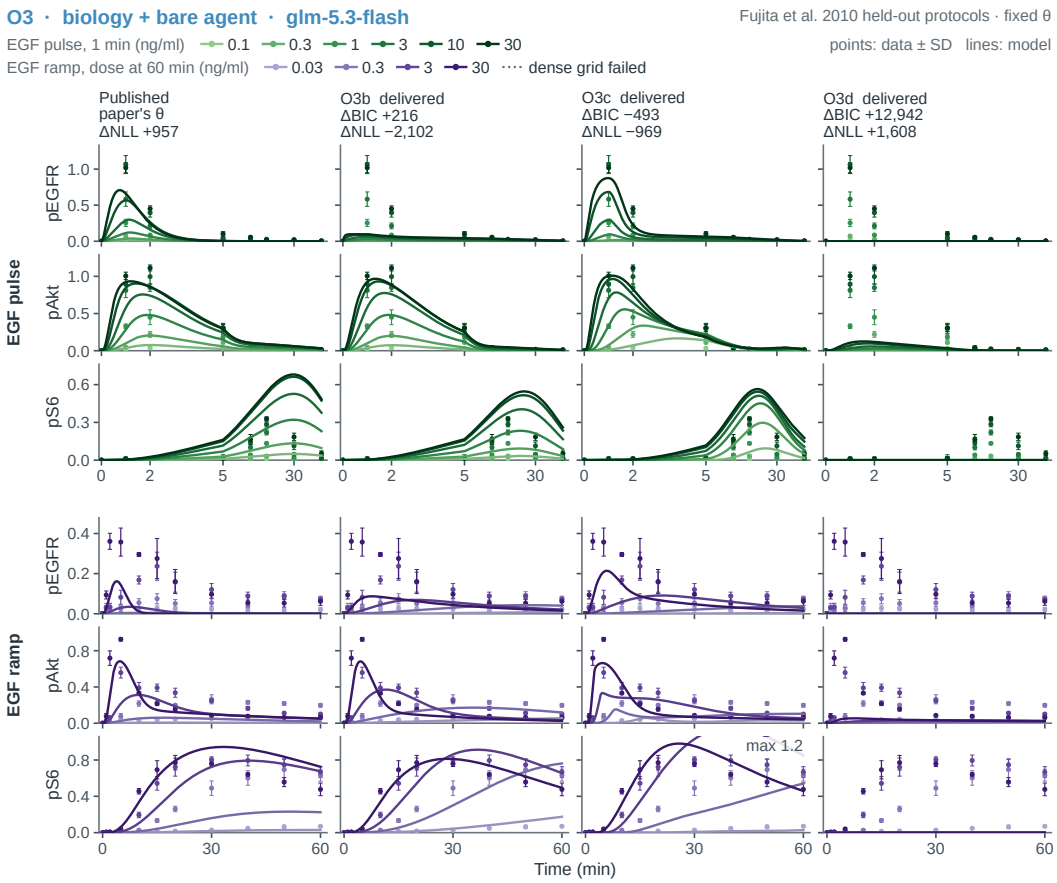

Figure 9: **Held-out Fujita predictions, O3** (biology and bare agent; GLM-5.3-Flash). As Figure 7; columns, the published model, then O3b–O3d; two O3 runs delivered no model.

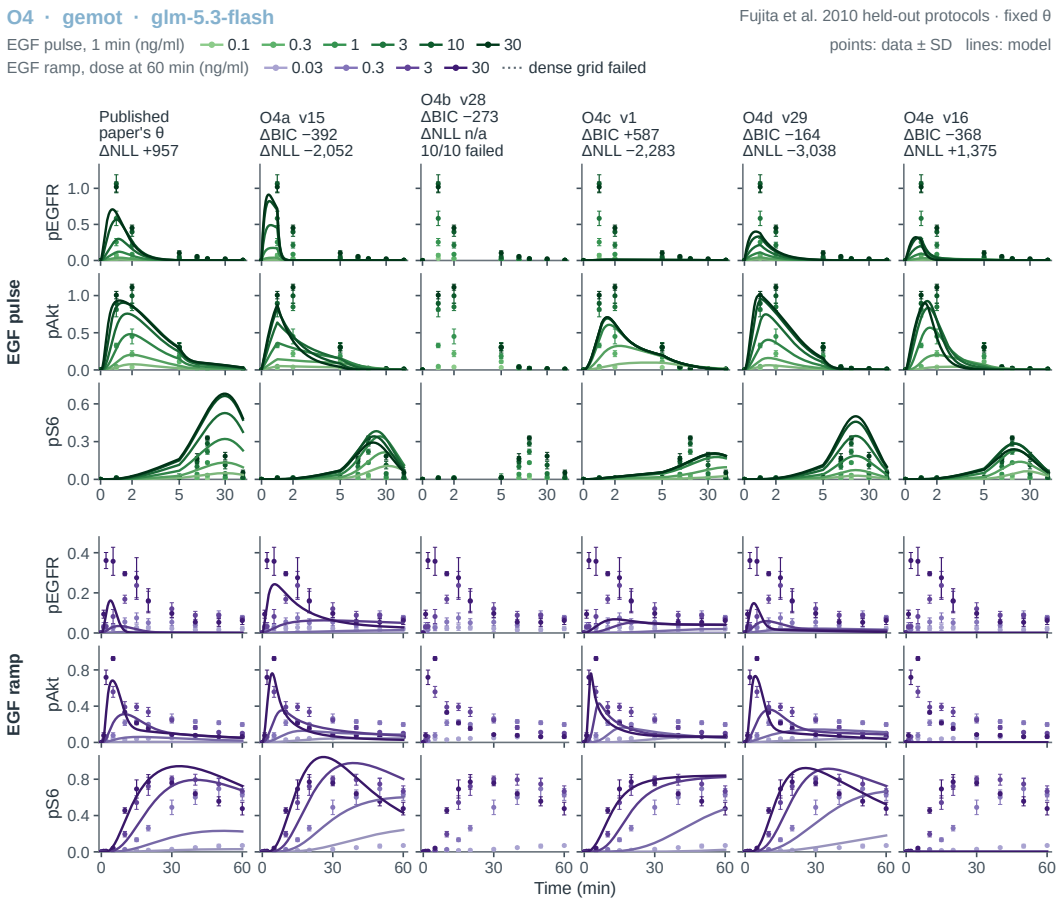

Figure 10: **Held-out Fujita predictions, O4** (gemot, biology blinded; GLM-5.3-Flash). As Figure 7; columns, the published model, then O4a–O4e.

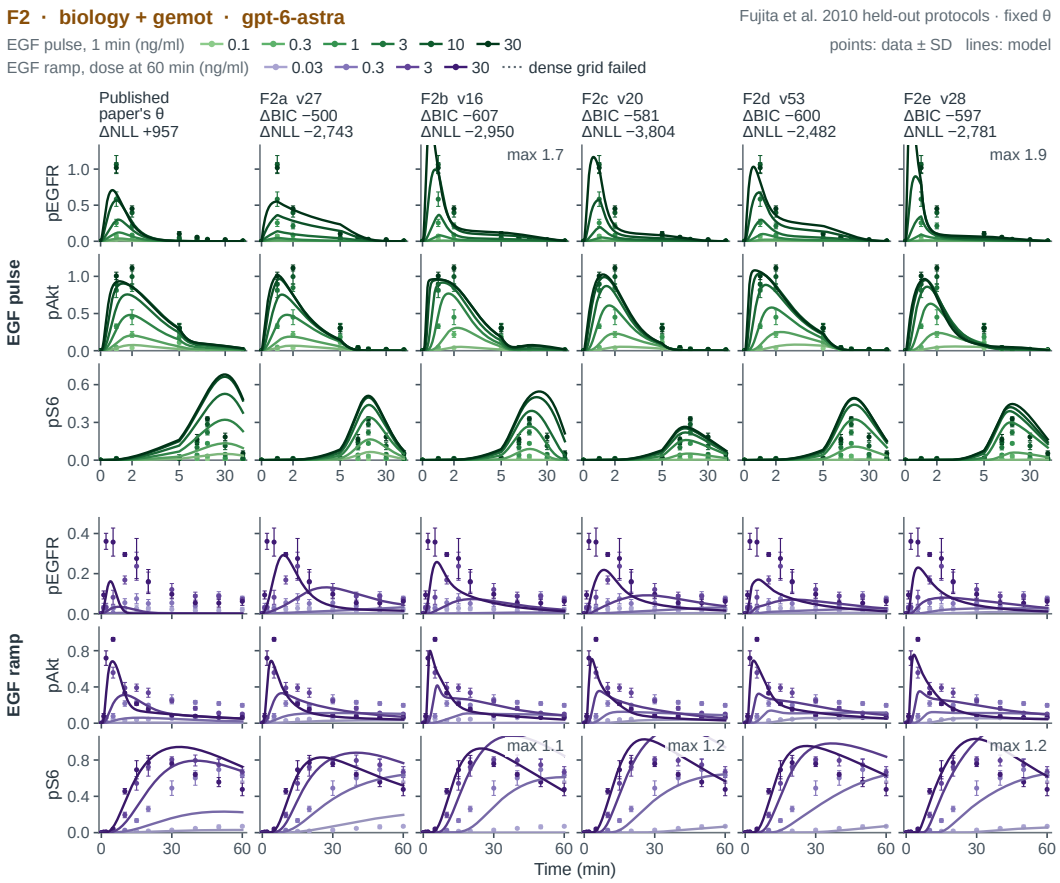

Figure 11: **Held-out Fujita predictions, F2** (biology and gemot; GPT-6-Astra). As Figure 7; columns, the published model, then F2a–F2e.

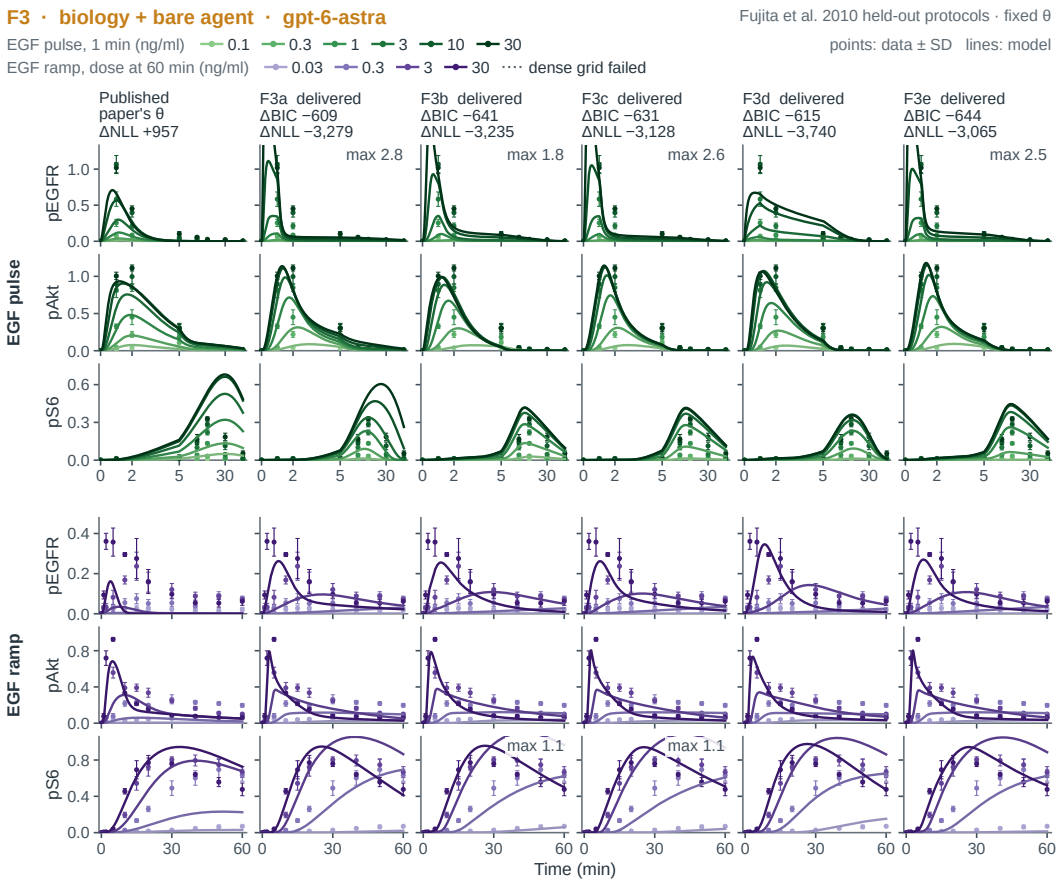

Figure 12: **Held-out Fujita predictions, F3** (biology and bare agent; GPT-6-Astra). As Figure 7; columns, the published model, then F3a–F3e.

D.6 Additional validation systems in Figure 4

The compact score export covers 20 biology-visible O1 GLM models in Figure 4: five each for Crauste, Lucarelli, Perelson and Raimundez. It does not cover the biology-blinded or neural-model comparisons in that figure.

The evaluation protocols only partly reproduce the original studies' validation designs. Raimundez keeps dynamical parameters fixed, using published nominal values for the reference, and analytically fits new per-blot observation scales for the subset of validation experiments in the original Figures 5D and 6E (Raimúndez et al., 2020). Crauste refits each structure to the tumour and vaccinia responses in original Figures 2D and 2B; the former was a published transfer test, whereas the original early-to-late prediction tests in Figures 6 and S6 are not reproduced here (Crauste et al., 2017). Perelson uses four additional patients withheld from model construction, retaining patient-specific refitting as in the original study (Perelson et al., 1996). Lucarelli transfers to three further cell types, pooling genes from original Figure 6B (used for fitting) and Figure S6 (originally withheld), and refits all estimated parameters for candidate and reference rather than only signalling parameters as in the original study (Lucarelli et al., 2018). This broader refit treats candidates and reference symmetrically because the harness cannot reliably separate signalling from gene-regulatory parameters in arbitrary agent-built models. It tests whether model structures can be refitted across cell types, without testing whether the learned Smad-complex-to-gene-expression mapping transfers unchanged.

Median held-out $\Delta\mathrm{NLL}$ values were $+7.4$ for Crauste, $+29.8$ for Lucarelli, -8.2 for Perelson, and $+83.5$ for Raimundez; all favour the references except Perelson, whose median favours the candidates. Four of five Perelson candidates improved on the reference; no Crauste, Lucarelli or Raimundez candidate did.

Held-out and transfer predictions. Figures 13–26 draw the runs of Figure 4, biology visible (O1) and biology blinded (O4), on their held-out data; Figures 27 and 28 draw its distilled neural-ODE models the same way. Raimundez is predicted at fixed dynamical parameters with each held-out blot's observation scale solved at the fitted point; Crauste, Lucarelli, and Perelson are drawn at the optimum each model reached when refitted to each held-out unit from its training optimum. Column titles carry the training ΔBIC and held-out ΔNLL plotted in Figure 4, except that a distilled model's title gives the held-out ΔNLL of its drawn refit: these are refitted with the neural judge's 50 starts, as Figure 4 scored them, and need not settle in the optimum the judge's refit found. The Raimundez figures and the biology-visible Perelson runs are drawn at a tighter tolerance than their scores were computed at (10^{-16} absolute, 10^{-14} relative); one blinded Raimundez run (7fd274ec), whose held-out objective is not finite there, is drawn instead at the fit tolerance its score used, where its drawn likelihood reproduces the score. The other transfer figures are drawn at the fit tolerance their scores used (10^{-10} absolute, AMICI's default relative tolerance). Four of the five blinded Raimundez models do not respond to the EGFR knockdown, where every biology-visible model does: the knockdown input is constant in the training data and, under blinding, was not wired into EGFR synthesis.

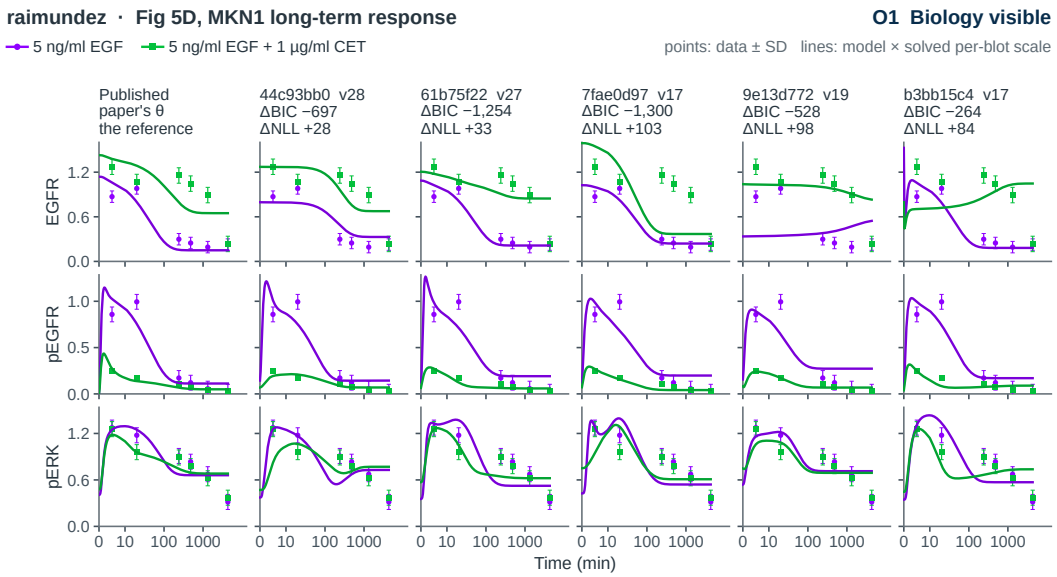

Figure 13: **Held-out Raimundez predictions, Fig. 5D, biology visible (O1)**. MKN1 long-term response to 5 ng/ml EGF (purple) and EGF with 1 μg/ml cetuximab (green). Columns, the published model at its nominal parameters (the reference), then the five Figure 2 runs. Points, data $\pm$ SD; lines, each model's readout times the per-blot observation scale solved at the fitted point, the quantities the held-out likelihood scores.

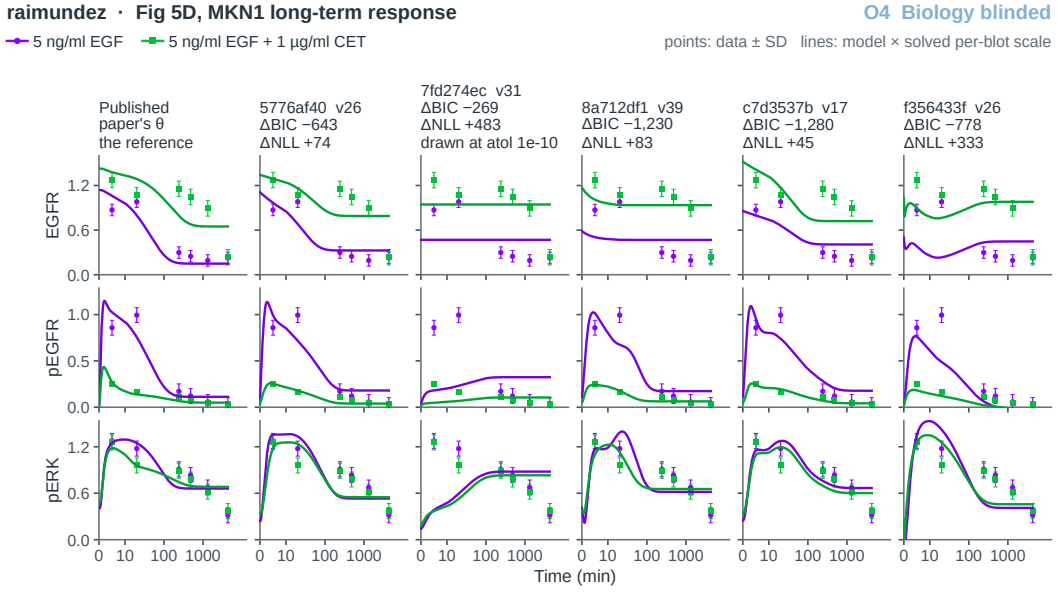

Figure 14: **Held-out Raimundez predictions, Fig. 5D, biology blinded (O4)**. As Figure 13, for the five biology-blinded runs.

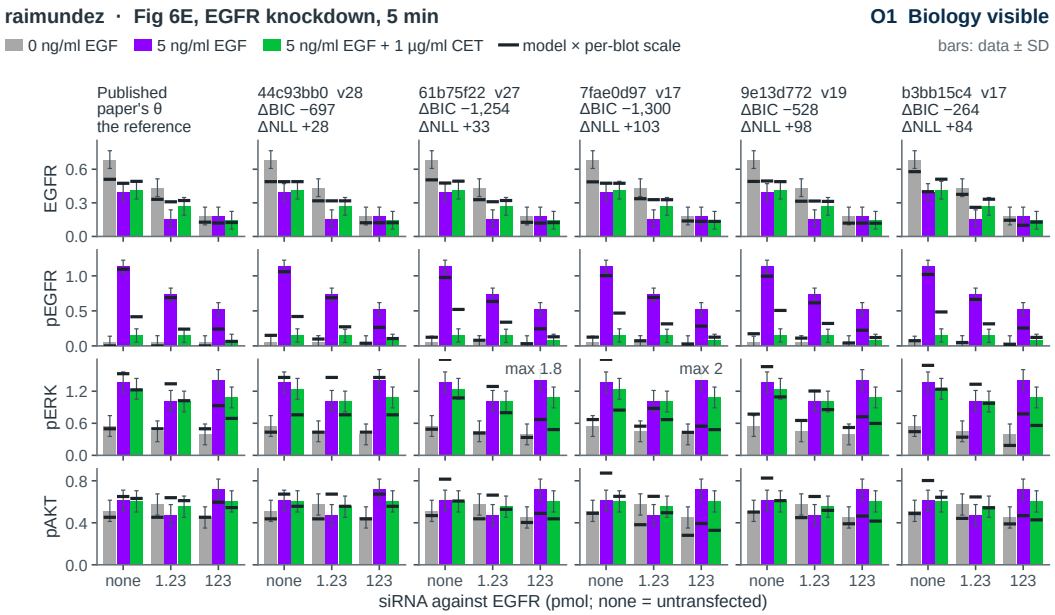

Figure 15: **Held-out Raimundez predictions, Fig. 6E, biology visible (O1)**. EGFR knockdown, read 5 min after no stimulus (grey), EGF (purple), or EGF with cetuximab (green). Bars, data ± SD; dark levels, each model's scaled prediction.

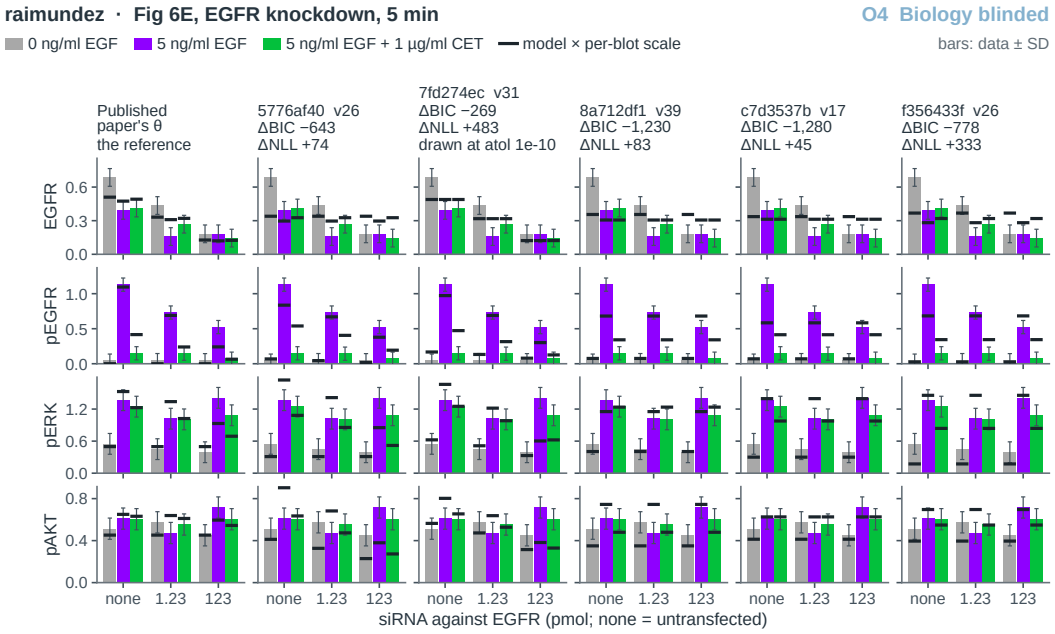

Figure 16: **Held-out Raimundez predictions, Fig. 6E, biology blinded (O4)**. As Figure 15, for the five biology-blinded runs.

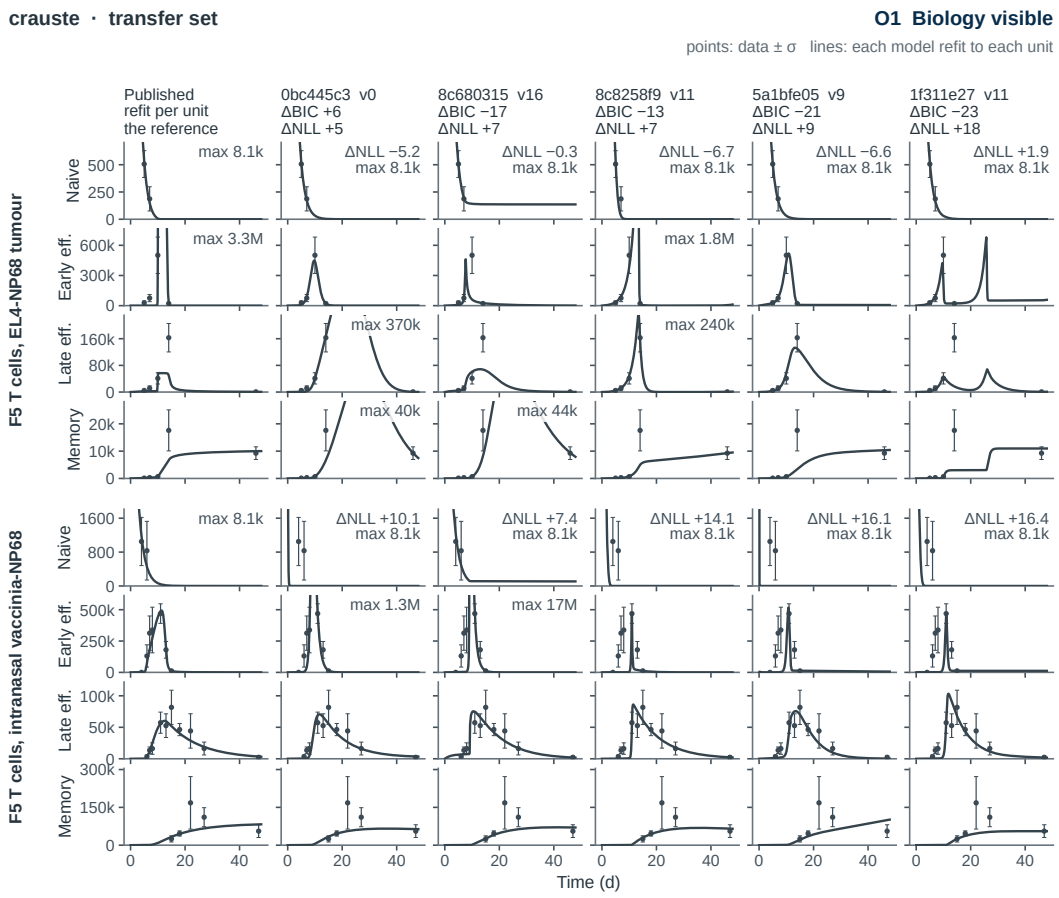

Figure 17: **Crauste transfer refits, biology visible (O1)**. Two F5 T-cell experiments as units, four populations each. Columns, the published structure refitted per unit (the reference), then the five Figure 2 runs, each refitted per unit. Points, data $\pm$ σ (the refit's own where estimated); lines, the refitted model; the first panel of each unit gives that unit's ΔNLL.

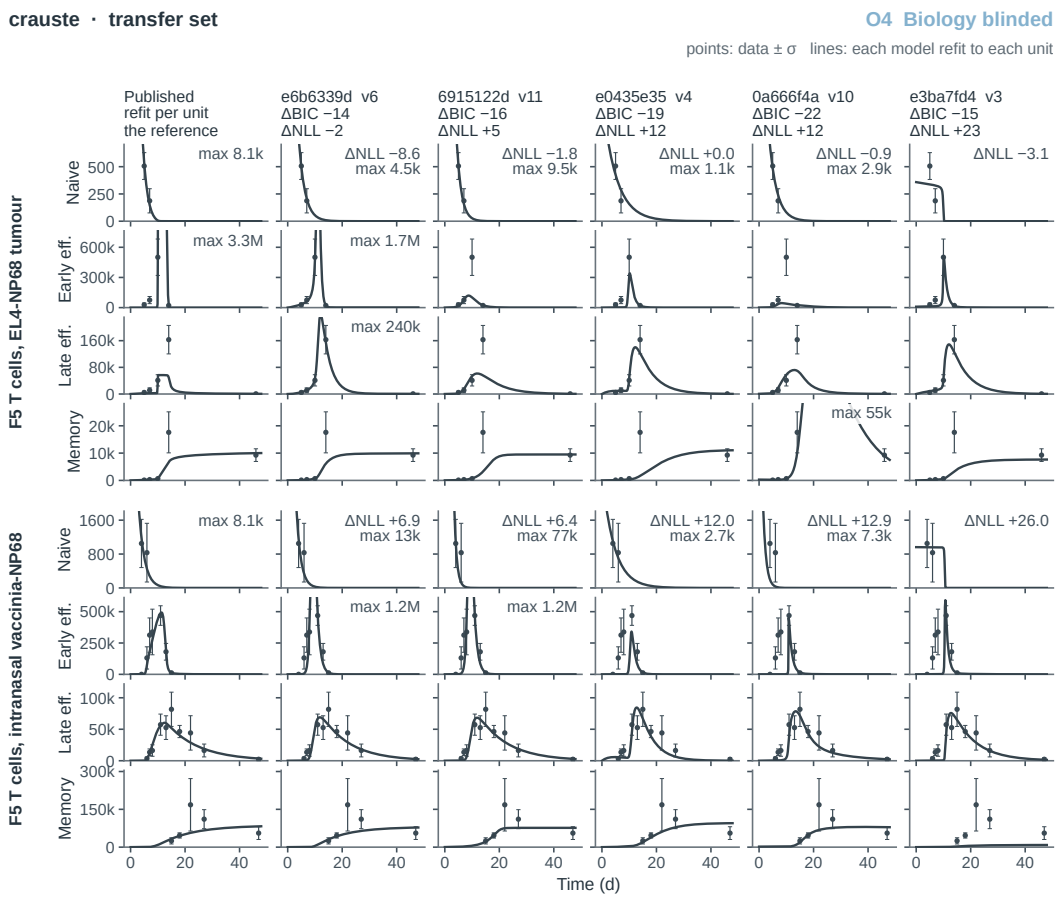

Figure 18: **Crauste transfer refits, biology blinded (O4)**. As Figure 17, for the five biology-blinded runs.

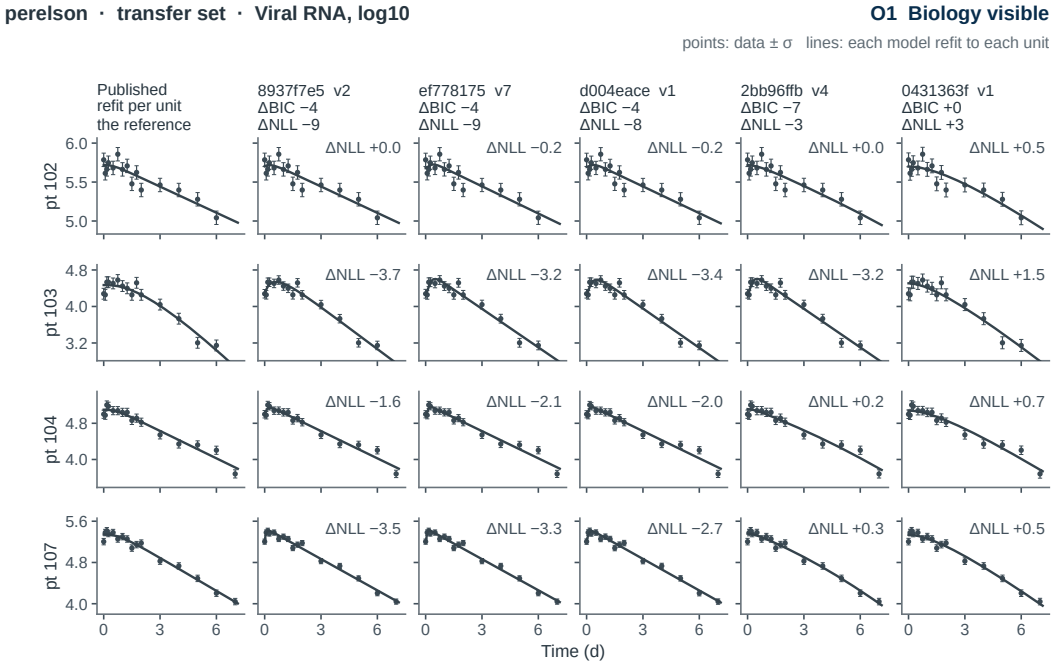

Figure 19: **Perelson transfer refits, biology visible (O1)**. Viral RNA ($\log_{10}$) for four patients as units. Encoding as in Figure 17.

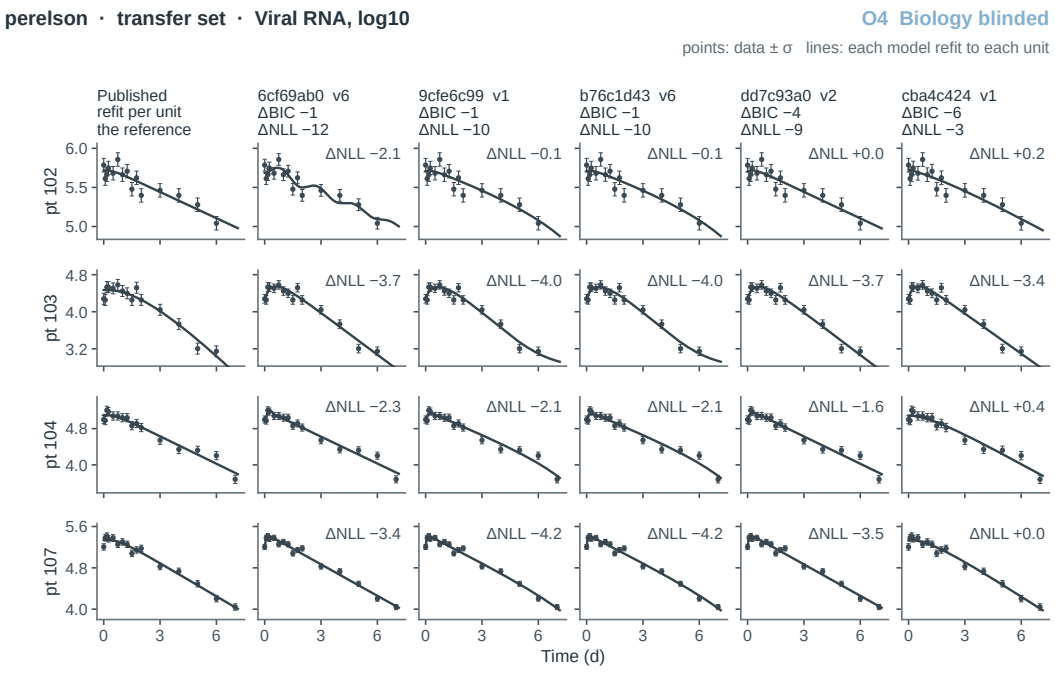

Figure 20: **Perelson transfer refits, biology blinded (O4)**. As Figure 19, for the five biology-blinded runs.

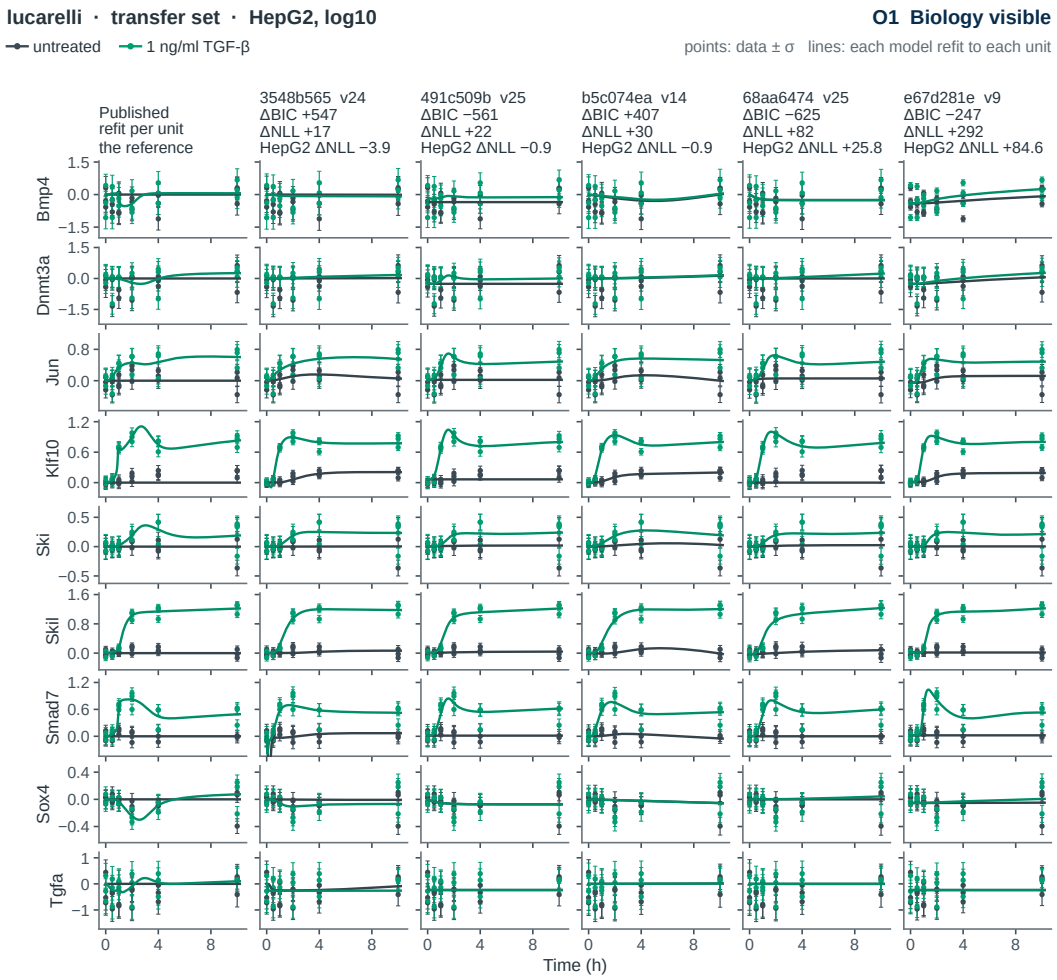

Figure 21: **Lucarelli transfer refits, HepG2, biology visible (O1)**. Gene expression ($\log_{10}$) in HepG2 cells, untreated (slate) and with 1 ng/ml TGF-β (green). Encoding as in Figure 17; the fourth title line gives the cell type's share of each run's ΔNLL.

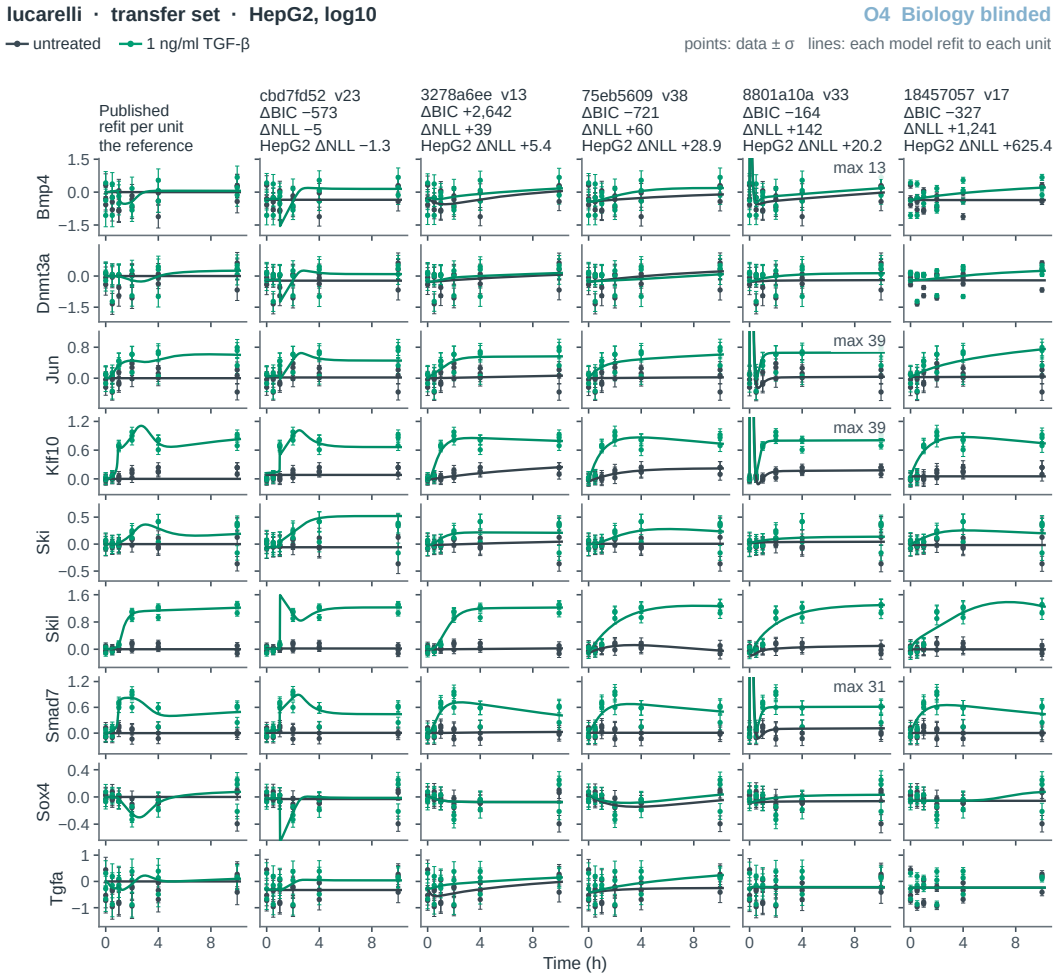

Figure 22: **Lucarelli transfer refits, HepG2, biology blinded (O4)**. As Figure 21, for the five biology-blinded runs.

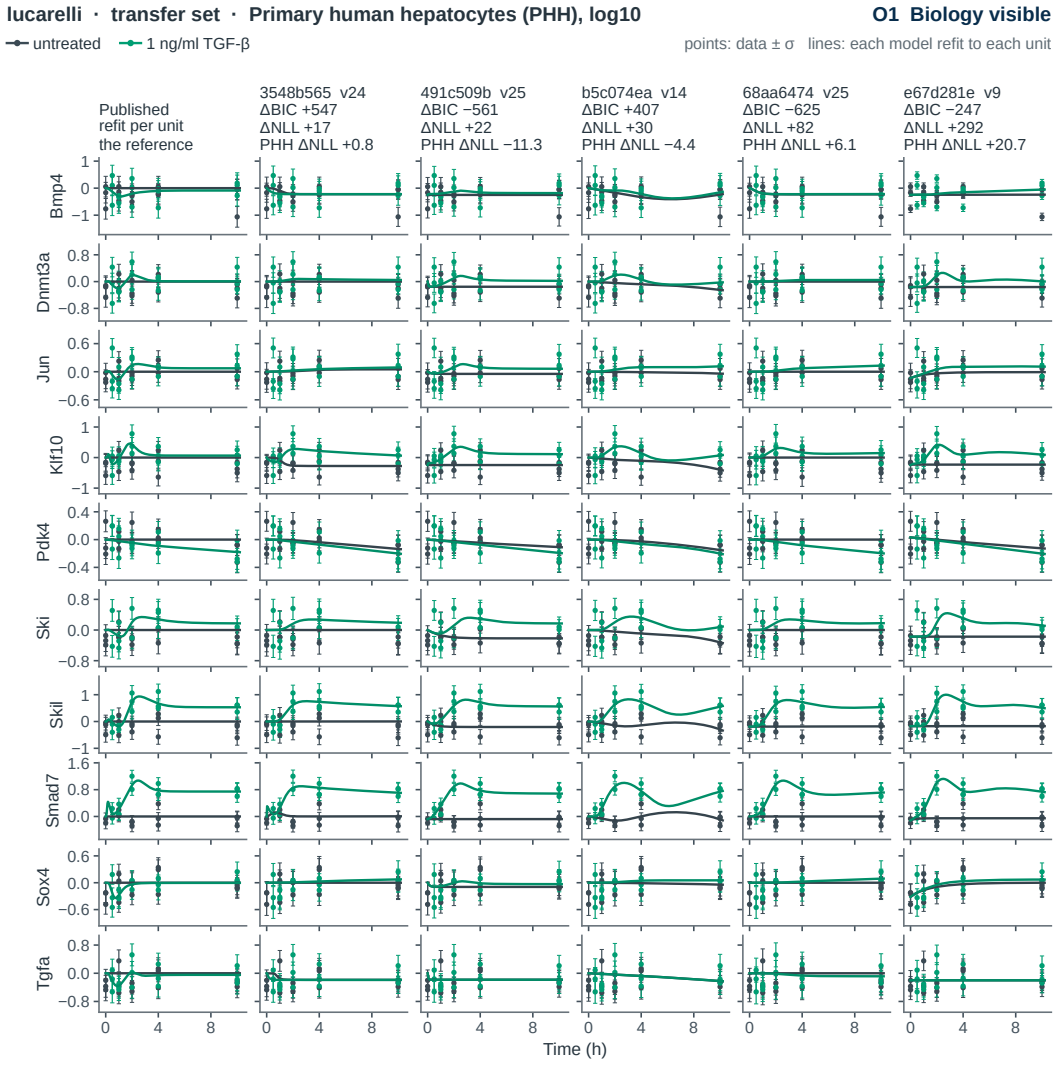

Figure 23: **Lucarelli transfer refits, PHH, biology visible (O1)**. Gene expression ($\log_{10}$) in primary human hepatocytes, untreated (slate) and with 1 ng/ml TGF-β (green). Encoding as in Figure 17; the fourth title line gives the cell type's share of each run's ΔNLL.

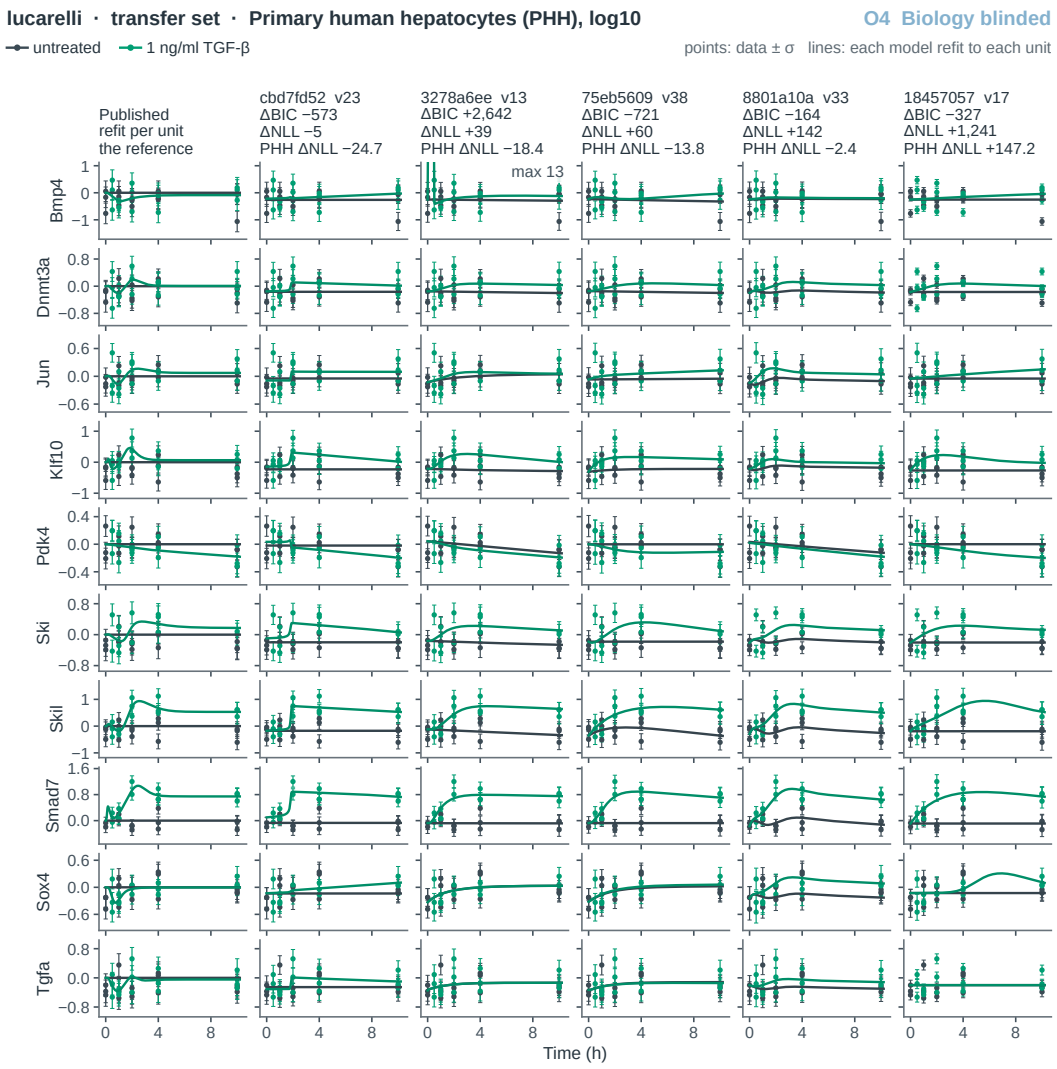

Figure 24: **Lucarelli transfer refits, PHH, biology blinded (O4)**. As Figure 23, for the five biology-blinded runs.

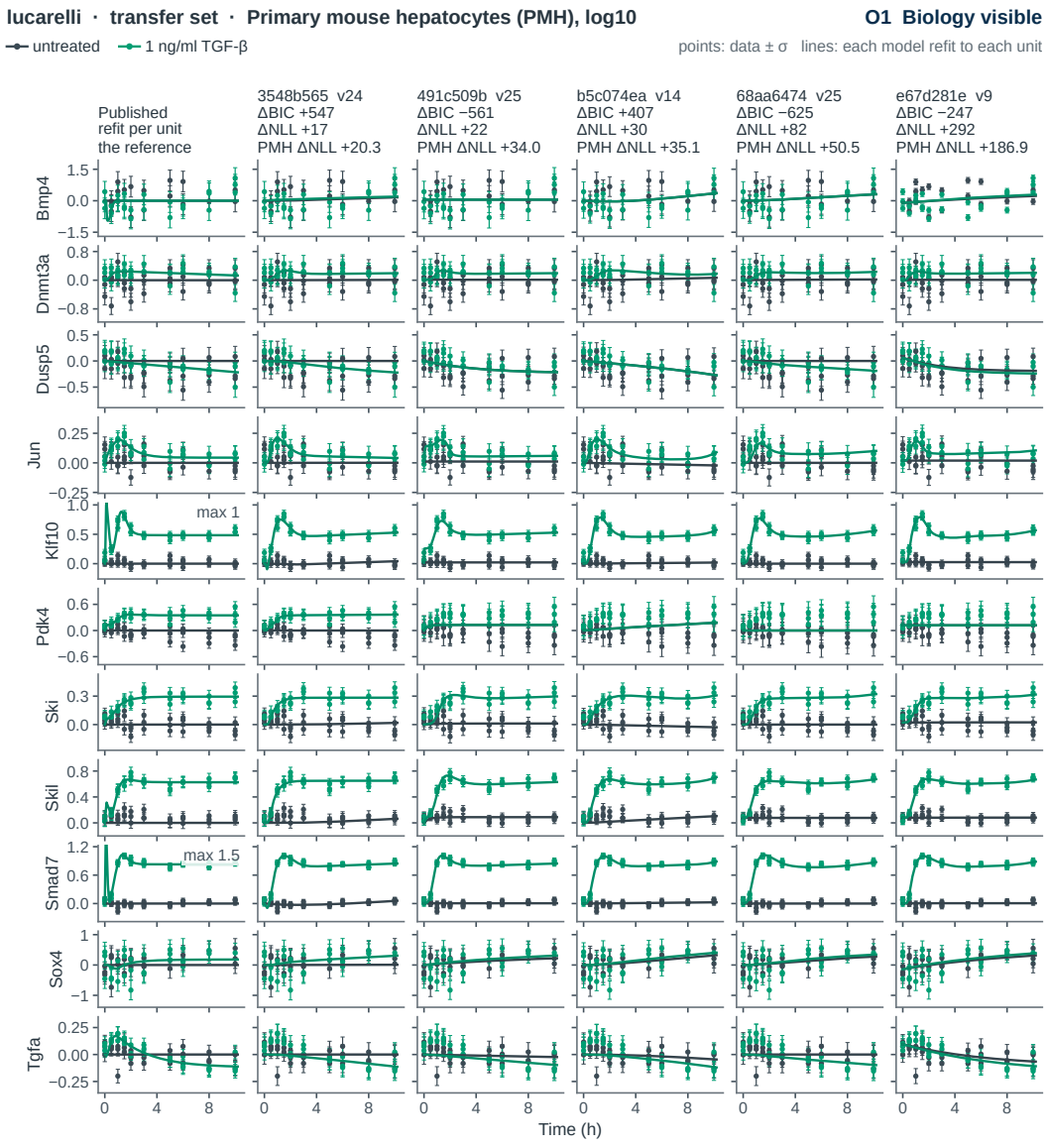

Figure 25: **Lucarelli transfer refits, PMH, biology visible (O1)**. Gene expression ($\log_{10}$) in primary mouse hepatocytes, untreated (slate) and with 1 ng/ml TGF-β (green). Encoding as in Figure 17; the fourth title line gives the cell type's share of each run's ΔNLL.

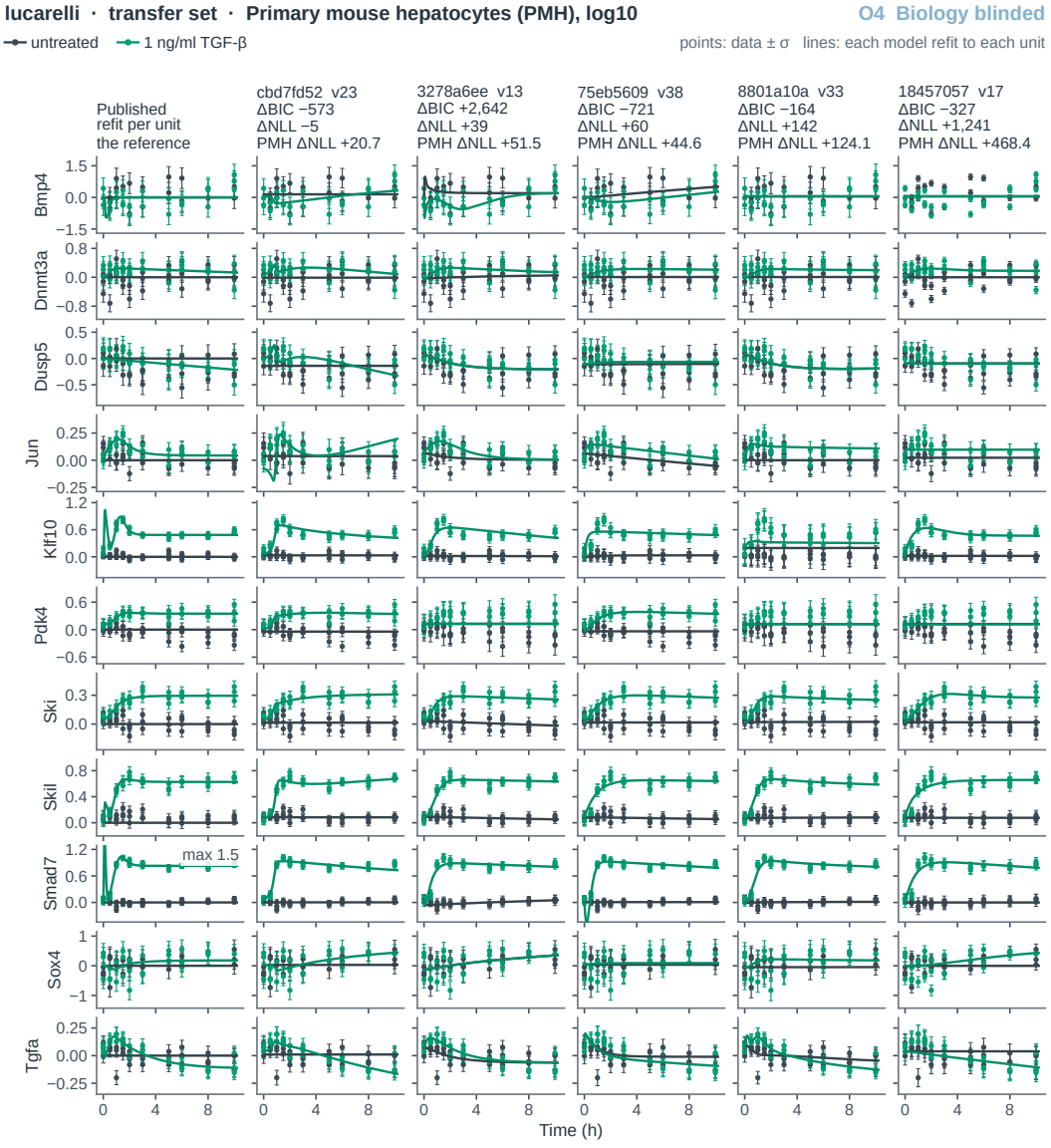

Figure 26: **Lucarelli transfer refits, PMH, biology blinded (O4)**. As Figure 25, for the five biology-blinded runs.

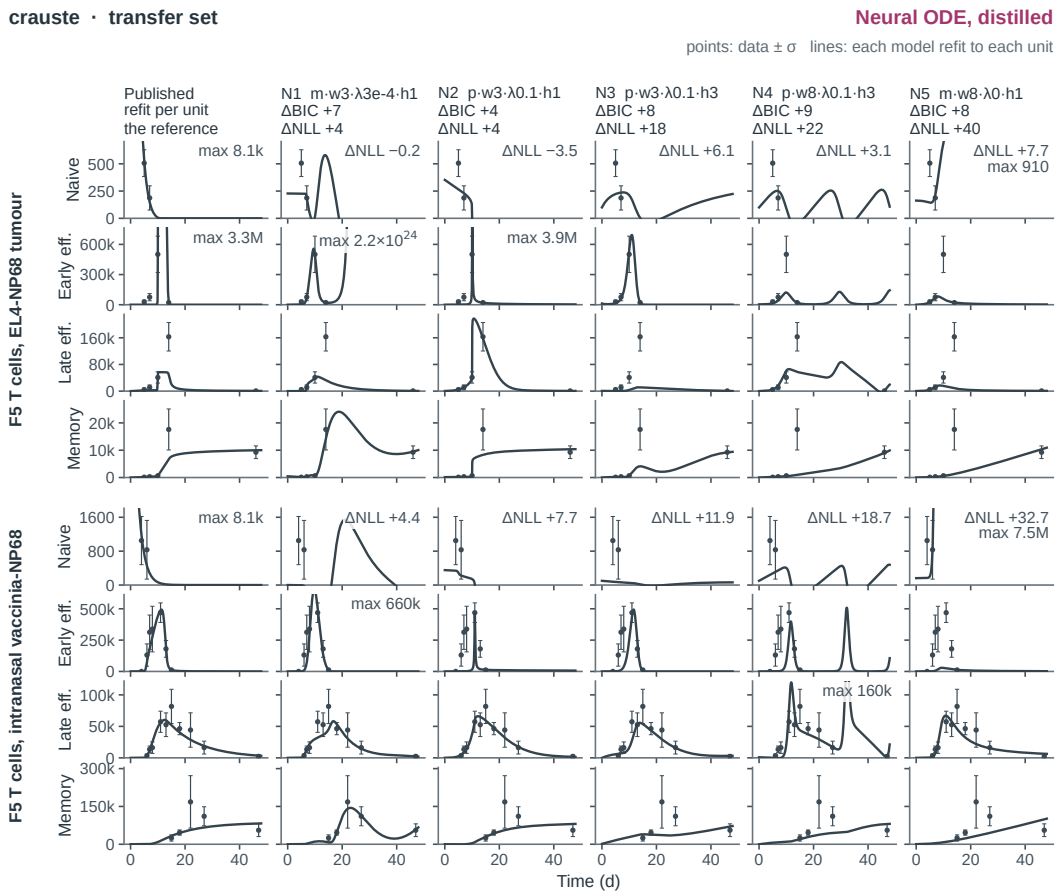

Figure 27: **Crauste transfer refits, distilled neural ODEs**. As Figure 17, for the five distilled neural-ODE models of Figure 4 (Appendix D.7), each refitted per unit with the neural judge's 50 starts. Columns N1–N5 are titled by grid cell: plain (p) or multiplicative (m) field, hidden width w, $\ell_{2,1}$ weight λ, and horizon stages h. The held-out ΔNLL in each title is that of the drawn refit.

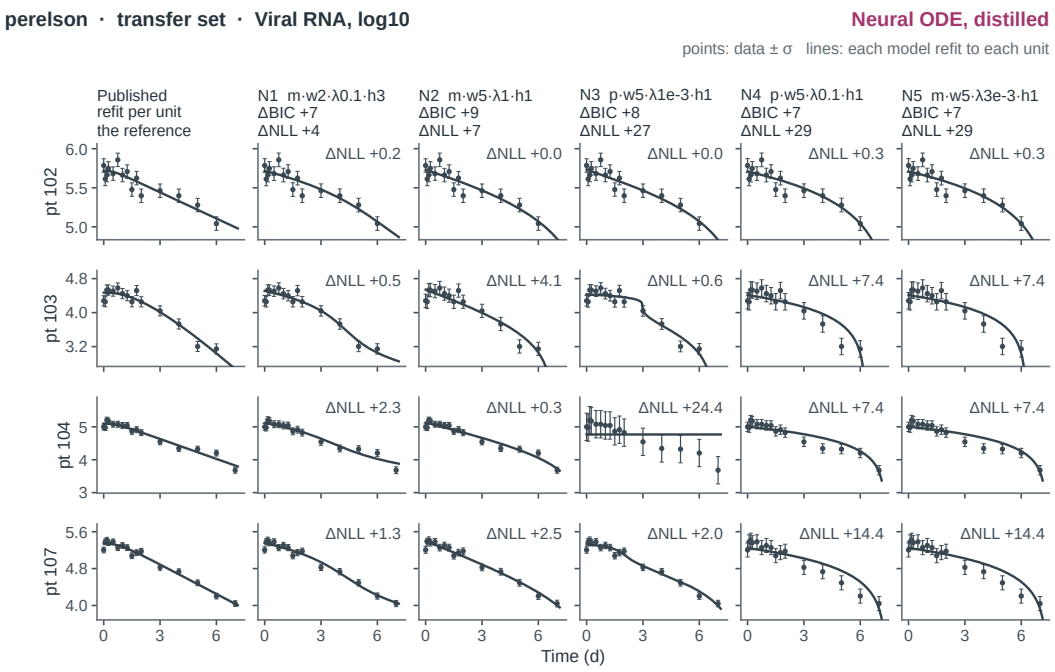

Figure 28: **Perelson transfer refits, distilled neural ODEs**. As Figure 19, for the five distilled neural-ODE models of Figure 4, each refitted per unit with the neural judge's 50 starts; columns as in Figure 27. N4 and N5 come from different grid cells but distil to the same equation, viral RNA falling at a constant rate under therapy, and differ only in fitted constants.

D.7 NEURAL ODE DISTILLATION PROTOCOL

The neural baseline constructs models without an LLM: a neural ODE (Chen et al., 2018) is fitted to training measurements, then its vector field is distilled into symbolic rate laws using PySR (Cranmer, 2023). The symbolic model is subsequently refitted through `gemot`'s numerical evaluation layer (Appendix B.7). Figure 4 reports these distilled models; their parameter penalties count the fitted symbolic model's parameters, including estimated observation parameters, after the neural network has been removed.

Data interface and neural training. The implementation reads the supplied measurement, observable, condition and parameter tables, together with time-dependent input specifications where present. It does not read the reference SBML model. The `readout_*` symbols of the observable formulas and explicitly dosed species define the initial state representation, which can be augmented with latent states. The supplied observable and noise formulas, experimental inputs and preequilibration conditions are retained. Known initial amounts remain fixed; unspecified initial amounts are estimated. During neural training, observation scales are fixed at their supplied nominal values, allowing neural states to carry the corresponding amplitudes.

A fully connected network with tanh hidden activations predicts the state derivatives from the states and experimental inputs. States and inputs are normalised using data-derived scales, and time by the last finite training measurement time. Neural training problems are represented using PEtab SciML with PEtab version 2 (Pathirana et al., 2026; Persson et al., 2026) and fitted with AMICI's JAX backend in double precision, with adaptive Kvaerno5 integration and gradient-based optimisation. Positive scalar parameters, including estimated initial amounts and noise scales, are optimised in log space. The reported grid used multistart Adam, with one or three horizon stages and either no sparsity penalty or an $\ell_{2,1}$ penalty (Equation 13; Table 10). Failed integrations are recorded rather than treated as fitted models.

Multiplicative field. Let $\xi \in \mathbb{R}^d$ collect the normalised states, including any latent states, and experimental inputs. The plain field feeds ξ directly to the network. The multiplicative field augments it with all unordered pairwise products, including squares:

$$\frac{d\tilde{x}}{d\tau} = f_{W,b}(\phi(\xi)), \qquad \phi(\xi) = (\xi, (\xi_i \xi_j)_{1 \le i \le j \le d}),$$

where $\tilde{x}$ and τ denote normalised states and time, and $f_{W,b}$ is the neural network. This expands the first-layer input dimension from d to $d + d(d+1)/2$, making bilinear interactions, such as mass-action terms, and quadratic dependence directly available to the network. The products are algebraic features; the network outputs the state derivatives. Symbolic distillation uses the original variables as regressors, so PySR must recover any products in the symbolic rate laws.

Sparsity penalties. Let $W^{(\ell)}$, $\ell = 1, \ldots, L$, denote the neural field's weight matrices, b its biases, and ϑ the fitted scalar parameters. For training data D, the regularised objective is

$$\mathcal{J}(W, b, \vartheta) = -\log p(D \mid W, b, \vartheta) + \lambda R(W), \qquad \lambda \ge 0. \tag{12}$$

The grid uses $\lambda = 0$ for unpenalised training and otherwise applies

$$R(W) = R_{\ell_{2,1}}(W) = \sum_v \sqrt{\|W^{(1)}_{:,G_v}\|_F^2 + \varepsilon}, \qquad \varepsilon = 10^{-12}. \tag{13}$$

Here G_v indexes first-layer columns carrying input variable v. For the plain field, each group is one column; in the multiplicative field, each product column belongs to the groups of its constituent variables, so groups can overlap. Biases and fitted scalar parameters are not penalised. The $\ell_{2,1}$ penalty targets input-variable dependence. Penalised training additionally applies group-wise soft thresholding after each step with strength $\eta\lambda$, where η is the learning rate. After penalised training it attempts an unpenalised refit, initialised at the selected solution and holding zero first-layer weights fixed. This penalty guides neural-field construction; the reported BIC is computed from the subsequently fitted symbolic model.

Symbolic distillation and refitting. The trained field is sampled along simulated training trajectories and around the settled states reached during preequilibration, using the corresponding experimental or basal inputs and local state perturbations. Each preequilibrated experiment contributes 25 basal-state rows to the sampling pool. PySR regresses each state derivative against the sampled states and inputs. These regression targets are neural-field evaluations, rather than finite differences of the experimental measurements. Candidate expressions are retained along an accuracy–complexity frontier and combined into complete ODE systems. The equations are converted back to data units and exported as Antimony rate rules; reaction stoichiometry is not inferred by this step. Non-integer coefficients become named parameters initialised at their regressed values. Each proposed system is then reconciled, compiled and fitted to the experimental training data through the same MCP evaluation path used for agent-built models. Agreement with the neural field is therefore a proposal criterion, while the reported fit and BIC come from the refitted symbolic model.

Selection and transfer evaluation. The neural points in Figure 4 were selected from the `grid3-20260922` campaign. The export collects judged candidates with recorded training BIC and held-out scores, sorts them by training BIC, removes duplicate BIC values after rounding to six decimal places, and retains the first five per problem. Thus, these points summarise selected candidates from a hyperparameter search; their spread does not represent five independent repetitions of the full procedure. Deduplication by BIC is a numerical heuristic, not a test of symbolic model identity. For Crauste and Perelson, the symbolic structures undergo the transfer refitting described in Appendix D.6. Their held-out ΔNLL values use the same refitted references as the O1 and O4 comparisons.

Search grid and implementation. The search comprised 648 configurations, with 216 each for Crauste, Fujita and Perelson: three hidden widths, two augmented-state counts, two multiplicative-field settings, two horizon curricula and nine sparsity weights (Table 10). The archived specification, `paper/neural_distillation/neural_ode_grid.json`, records these axes, shared training, solver and distillation settings, evaluation budgets and execution revision `4d7da49`. It was extracted from per-cell configurations, recorded trainer and distiller settings, and launch metadata; `paper/generated/neural_ode_grid.csv` enumerates every configuration. All ten plotted neural candidates map to cells in this grid; their selection procedure and scores are archived alongside the specification. The construction workflow above was documented from `src/petab_harness/neural_ode/` at revision `249d7c5`.

Table 10: **The neural-ODE search grid.** The axes above the rule define 216 configurations per benchmark; shared recorded settings appear below it. Each cell was allocated two CPUs, 16 GB of memory and a 12 h scheduler limit. These are search configurations, not counts of successful fits.

Setting	Values
Hidden width	crauste 3, 5, 8; fujita 4, 7, 12; perelson 2, 3, 5
Augmented states	0, 1
Multiplicative field	no, yes
Horizon stages	1, 3
$L_{2,1}$ weight (0 = none)	0, 3×10^{-4}, 1×10^{-3}, 3×10^{-3}, 0.01, 0.03, 0.1, 0.3, 1
Network depth	2
Optimiser	Adam
Learning rate	3×10^{-3}
Steps	5000
Restarts	4
Patience (steps)	300
Solver tolerances (rel., abs.)	1×10^{-6}, 1×10^{-8}
Solver step limit	8192
Shooting windows	1
Seed	0
Wall time per cell (s)	36000
Symbolic regression iterations	40
Expression size limit	20
Population size	33
Parsimony	3.2×10^{-3}
Operators	$+, -, \times, \div$
Samples of the field	2000
Proposals per cell	11 (complexity budgets 1–19, and the distiller's own selection)
Judge fit starts	50

E FUJITA MODEL DIAGRAMS AND PARAMETER UNCERTAINTY

The expanded Figure 3 compares 28 delivered models from 30 audited runs. Its display names identify the language model, configuration, and run; Appendix D.5 maps them to the original archive labels. The source annex (Appendix G) contains Antimony listings and parameter tables for all 28 delivered models across the six configurations. Twenty listings preserve the archived Antimony; eight bare-agent listings were converted from archived SBML. The manifest `paper/generated/fujita_models.csv` connects each exported source to its task, version, and hashes. Source initializations need not equal the separately saved fitted parameter values.

E.1 PARAMETER UNCERTAINTY AROUND THE PLOTTED OPTIMA

Figure 3A plots one parameter vector per model. To test whether that choice drives the held-out ranking, the vectors from a 1000-start multistart that the training data cannot distinguish from the best optimum found were re-scored on the held-out protocols in Appendix D.5 (Figure 30), every band member of every model. 342 of them, 192 in O1e, 95 in O2c, 52 in O4b, and 3 in O4e, have no held-out value because their held-out simulation did not integrate. Band membership was decided at the tolerance the held-out score is computed with (atol 10^{-16}, rtol 10^{-14}): at the fit tolerance, apparent spreads of optima were integration error rather than non-identifiability. Tightening the tolerance moves an archived optimum's training NLL by at most 8.5×10^{-4} (median 2.3×10^{-6}; more than 10^{-5} in 4 of 27 models) and its held-out NLL by at most 3.7×10^{-8}. All 2704 band members whose simulated values exceed ten times the largest held-out measurement fall outside the band at the tight tolerance.

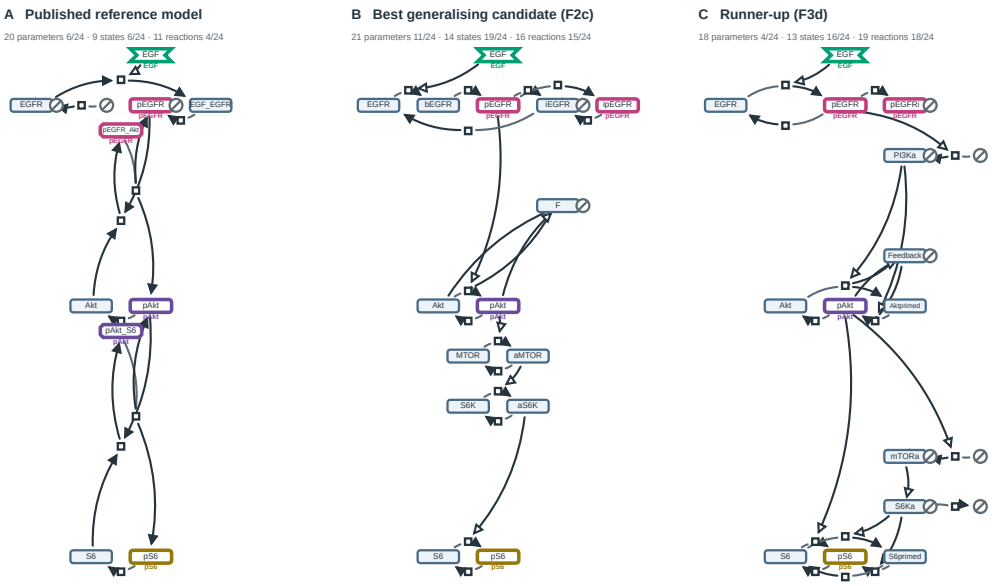

Figure 29: **The published reference model and the two best generalising candidates.** SBGN process description maps of **A,** the published reference model, **B,** the candidate with the lowest held-out NLL (F2c, held-out NLL 352.09, ΔNLL -3804.26) and **C,** the candidate with the next lowest (F3d, held-out NLL 416.59, ΔNLL -3739.75), drawn from one portable export with one deterministic layout, so no model is loaded or simulated to produce the figure. Glyphs: rounded, macromolecule; cut corners, complex; concave hexagon, perturbation; small square, process; struck circle, source or sink. Arcs: plain line, consumption; filled arrowhead, production; open arrowhead, stimulation; bar, inhibition. Entity pools claim the rows and processes the layers between them. A row is SHARED between the maps wherever something in them is the same: an entity summed into the same readout, or an entity carrying the same name. Those rows sit at identical heights in every map, in the order Figure 3 stacks the readouts, so the mechanisms can be read against one another row by row. Everything else is placed between the shared rows in proportion to how many arcs separate it from each, and every gap is made exactly as wide as the busiest map needs there, so pinning can stretch a map but never squeeze two glyphs together. A pool is outlined and named in the colour of the readout it is summed into (the EGF input likewise), the colours of Figure 3B. Entity classes follow a naming heuristic recorded with the export; modulation arcs are signed from the rate laws. Each heading gives that model's size and its rank, smallest first, among the 24 candidates that recovered the reference connections; a model that missed one is small or large for reasons unrelated to parsimony. Rows are grouped by conserved moiety, one entity in all the states any map gives it, found from the arcs: a process with one reactant and one product is a state change. Each moiety takes ONE row, the unmodified form on the left and each further state one step to the right, and every row is centred on the same axis. A complex joins neither moiety, so it sits on a half-row under the earlier of its parts. A process sits 30% of the way between the rows it joins, or 30% of a row off a row its pools share, towards the earlier row when it carries material rightwards and the later one when leftwards.

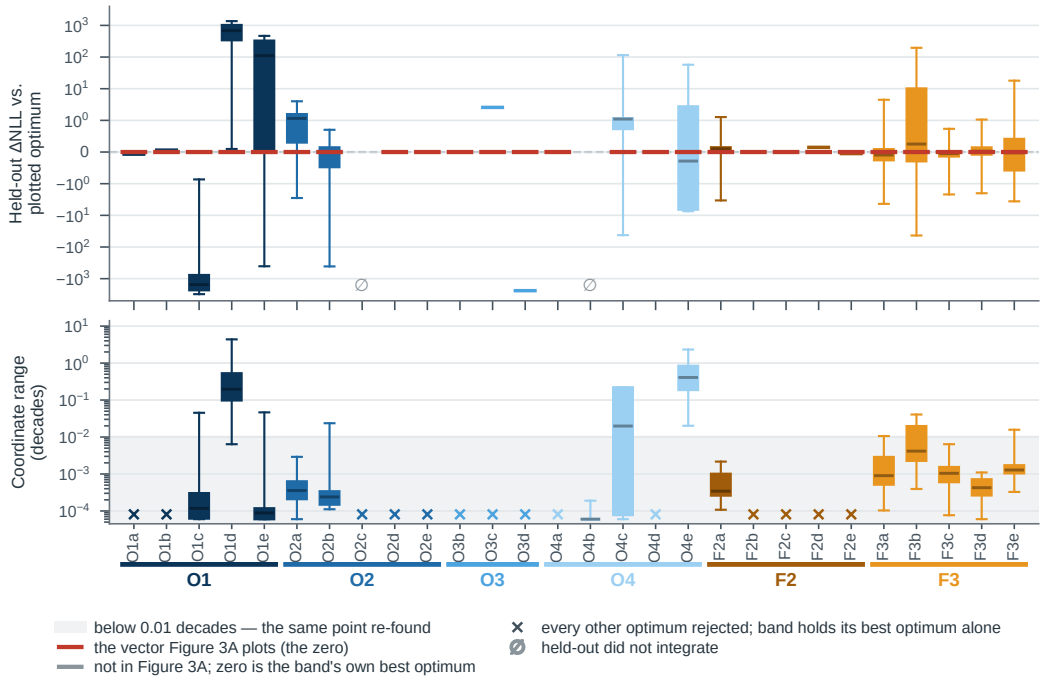

Figure 30: **Held-out score across each Fujita model's likelihood-ratio band.** Band members are the multistart optima whose training likelihood, re-evaluated at atol 10^{-16} / rtol 10^{-14}, lies within the 99% χ^2 threshold (df $= k$) of the best among them; 1000 starts per model, 154–1000 converged, every band member re-scored. **Top,** held-out NLL of each band member relative to the vector Figure 3A plots (red), on a symmetric log scale. **Bottom,** per-coordinate range across the band in decades; the shaded region (below 0.01 decades) is the same point re-found, and a cross marks a band holding only its best optimum.

F REFERENCE MODELS IN SBGN ACTIVITY FLOW NOTATION

The following diagrams cover all 18 published reference models, using the biological-activity and unknown-influence glyphs of SBGN Activity Flow (Mi et al., 2015). They depict structural dependencies between model variables: each rectangle represents a state or time-dependent input, labelled with its original SBML identifier. Blue rectangles contribute to at least one observable formula; this does not imply that these states are individually measured. A diamond at the target end denotes a dependency whose sign is not assigned.

Arcs are extracted from kinetic laws and assignment or rate rules, with algebraic definitions expanded where necessary. Reaction arcs connect quantities read by a rate law to states with nonzero net stoichiometric changes. Self-loops are retained: they can represent depletion or turnover, rather than biological feedback. These equation-level abstractions omit parameter values, stoichiometric magnitudes, initial conditions and observable formulas; they do not establish biological causality or replace the SBML models. They are distinct from the collapsed anchor-route graphs used in the topology analysis.

Sources are the SBML models and observable tables in Benchmark-Models-PEtab at revision `1763dfc`, the collection pin used for this benchmark suite. The manuscript sources include vector diagrams, the extracted dependencies and file hashes in `paper/reference_activity_flow/`, together with the extraction and rendering scripts.

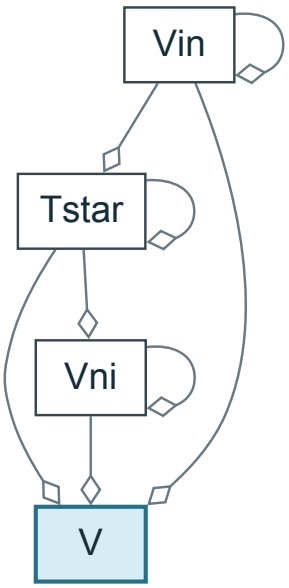

Figure 31: **Perelson reference model.** SBGN AF abstraction of `Perelson_Science1996`: 4 nodes and 8 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

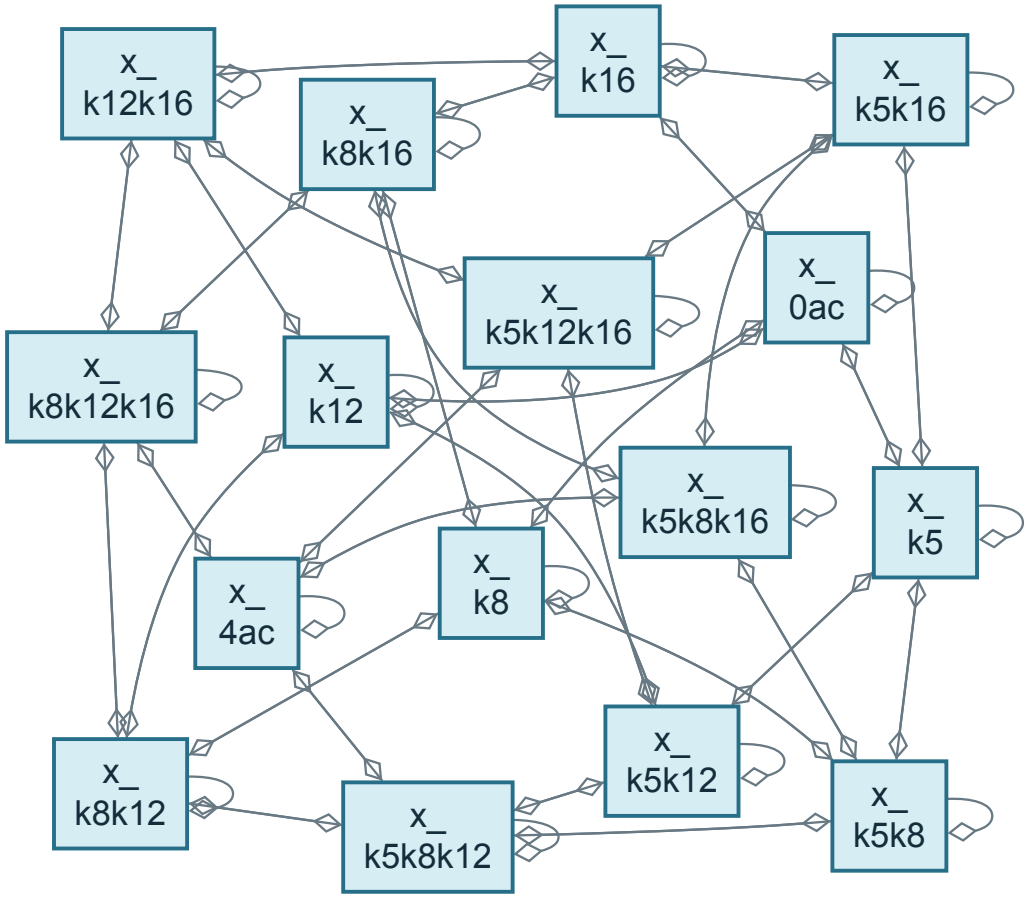

Figure 32: **Blasi reference model.** SBGN AF abstraction of `Blasi_CellSystems2016`: 16 nodes and 80 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

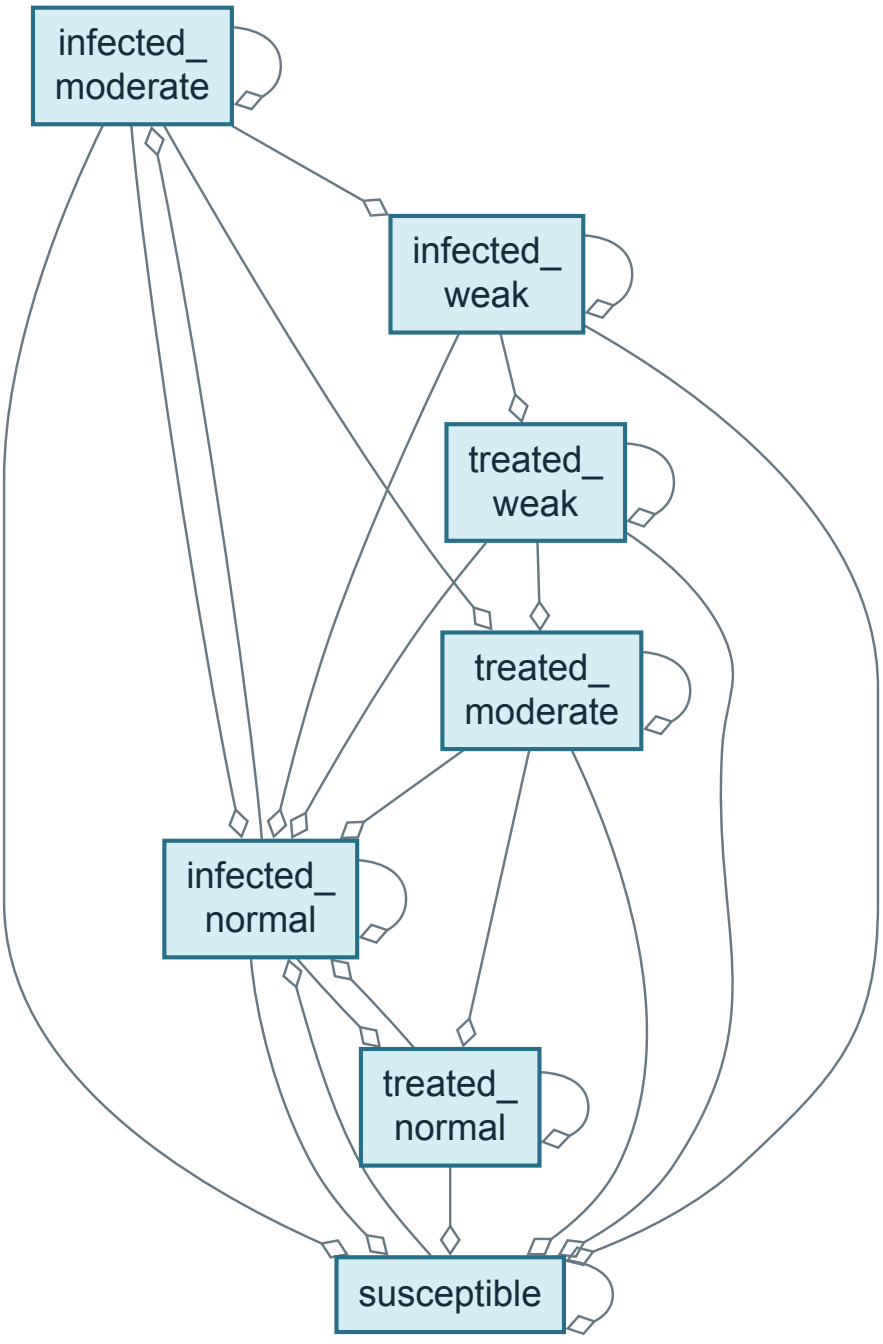

Figure 33: **Rahman reference model.** SBGN AF abstraction of `Rahman_MBS2016`: 7 nodes and 26 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

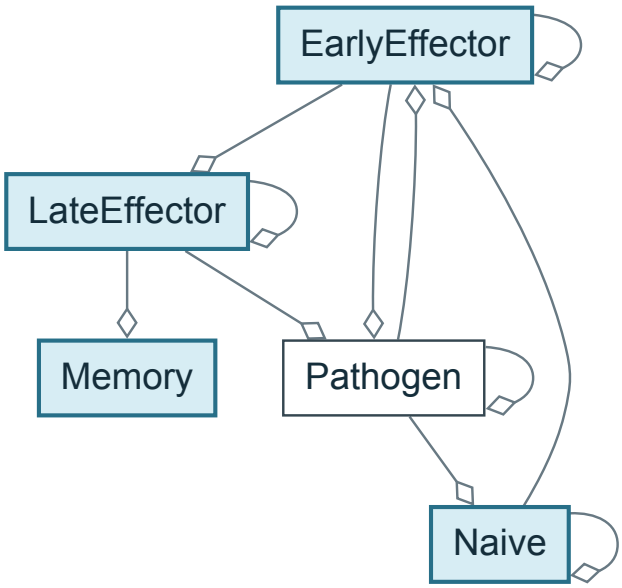

Figure 34: **Crauste reference model.** SBGN AF abstraction of `Crauste_CellSystems2017`: 5 nodes and 11 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

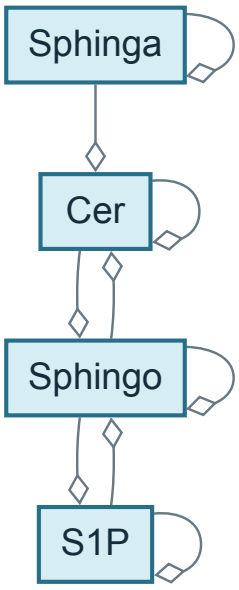

Figure 35: **Armistead reference model.** SBGN AF abstraction of `Armistead_CellDeathDis2024`: 4 nodes and 9 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

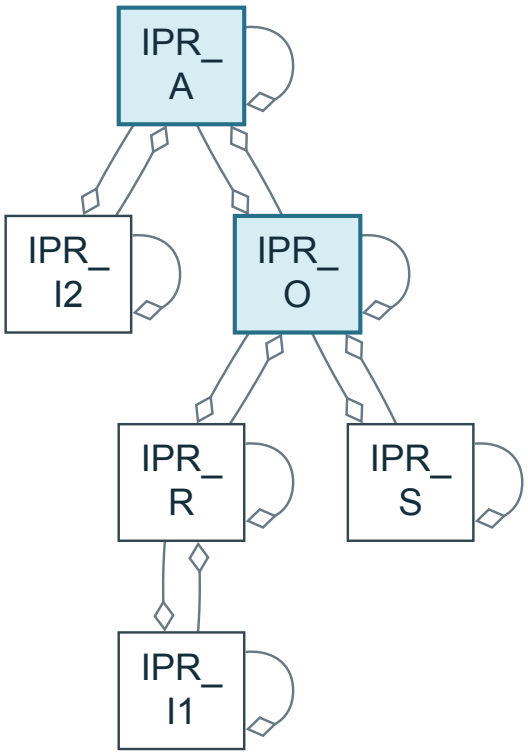

Figure 36: **Sneyd reference model.** SBGN AF abstraction of `Sneyd_PNAS2002`: 6 nodes and 16 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

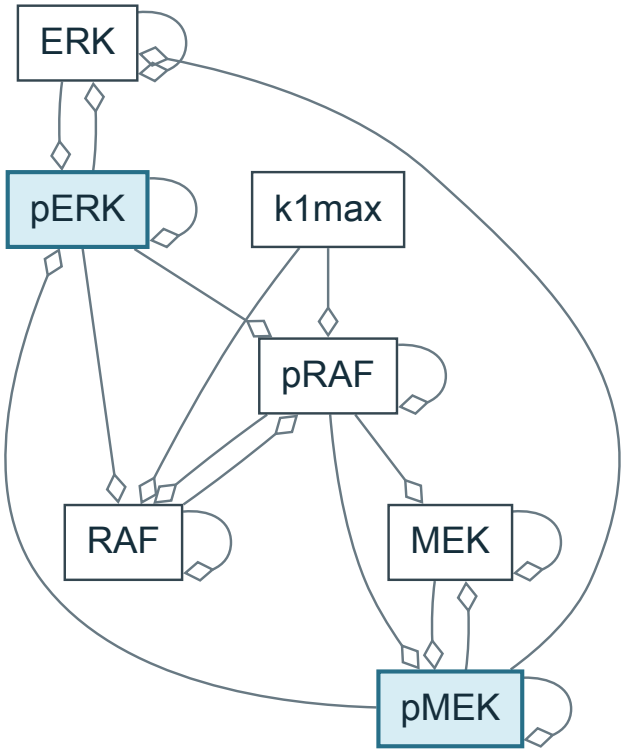

Figure 37: **Fiedler reference model.** SBGN AF abstraction of `Fiedler_BMCSystBiol2016`: 7 nodes and 20 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

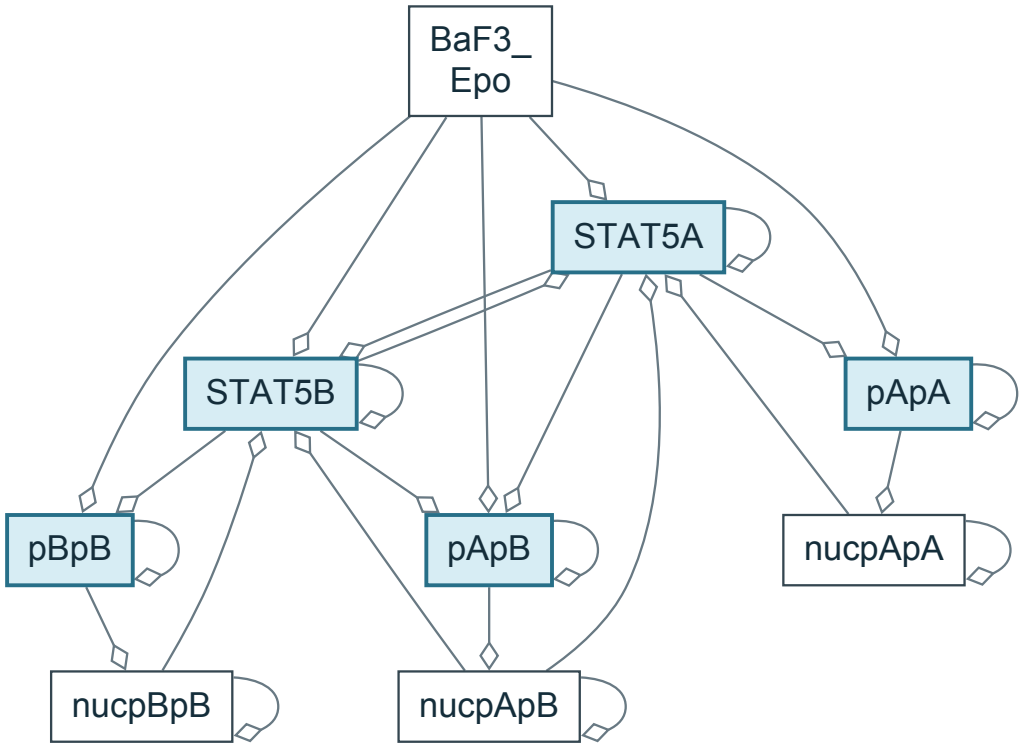

Figure 38: **Boehm reference model.** SBGN AF abstraction of `Boehm_JProteomeRes2014`: 9 nodes and 26 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

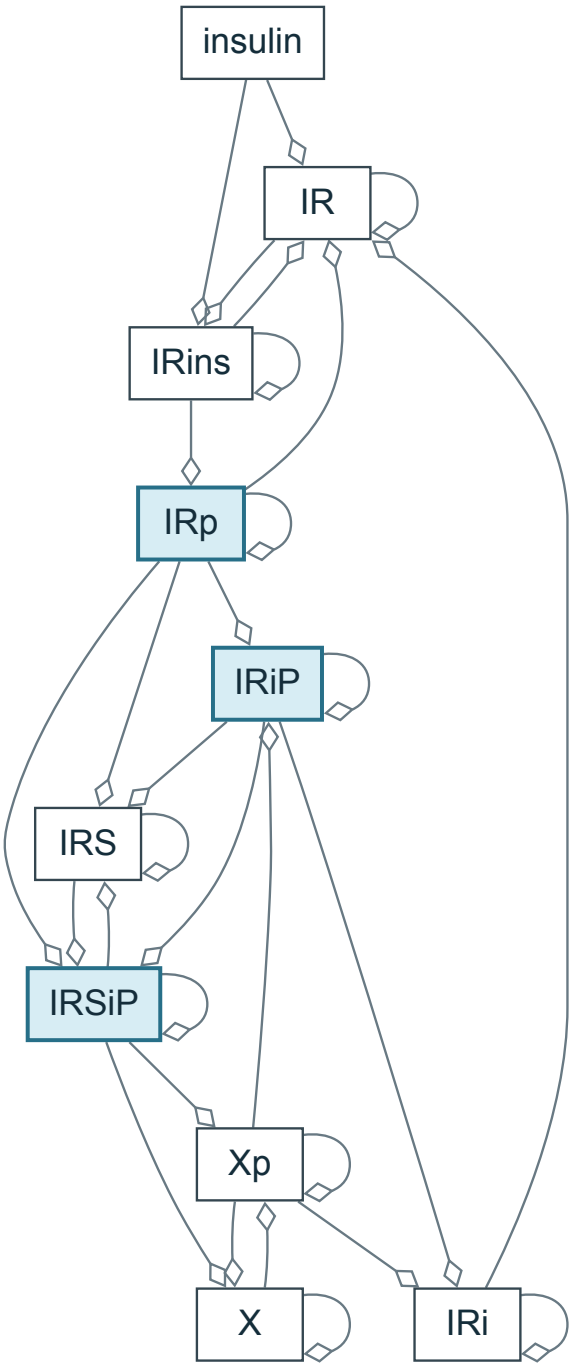

Figure 39: **Brännmark reference model.** SBGN AF abstraction of `Brannmark_JBC2010`: 10 nodes and 30 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

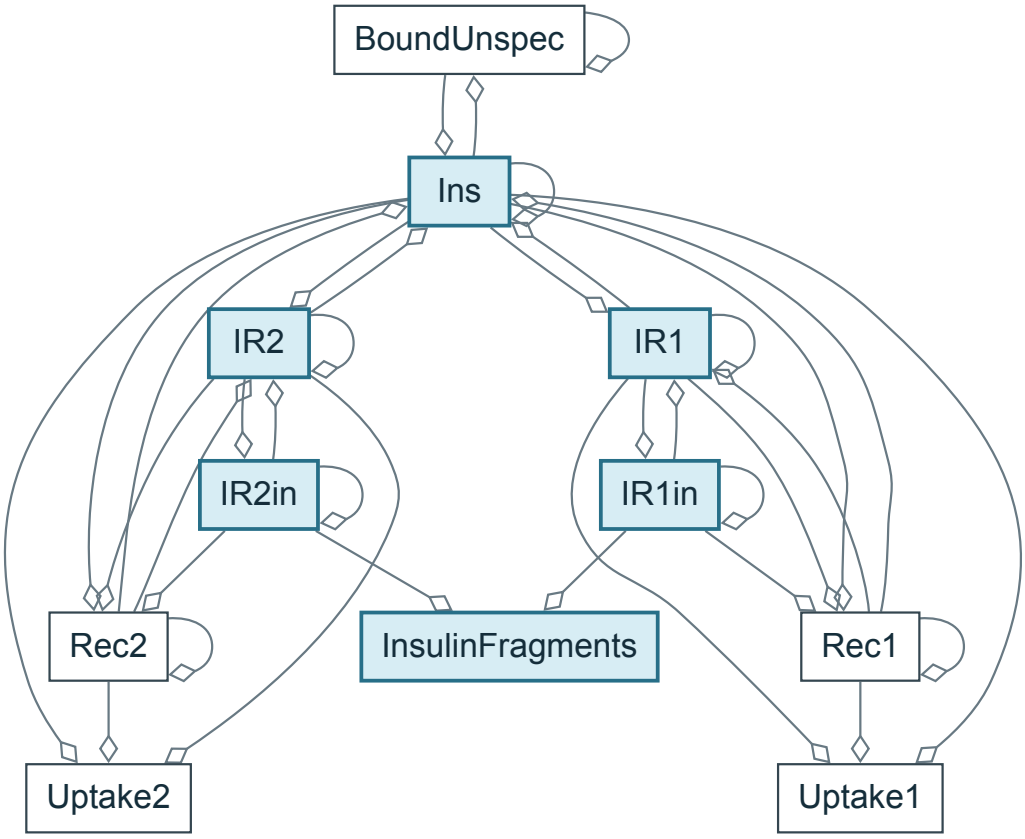

Figure 40: **Schwen reference model.** SBGN AF abstraction of `Schwen_PONE2014`: 11 nodes and 36 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

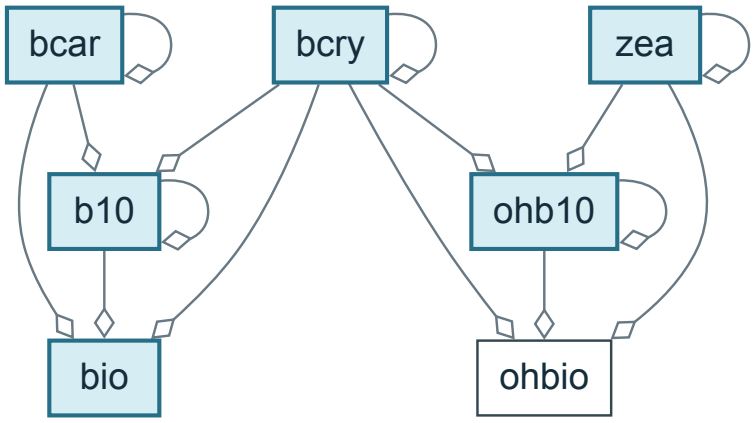

Figure 41: **Bruno reference model.** SBGN AF abstraction of `Bruno_JExpBot2016`: 7 nodes and 15 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

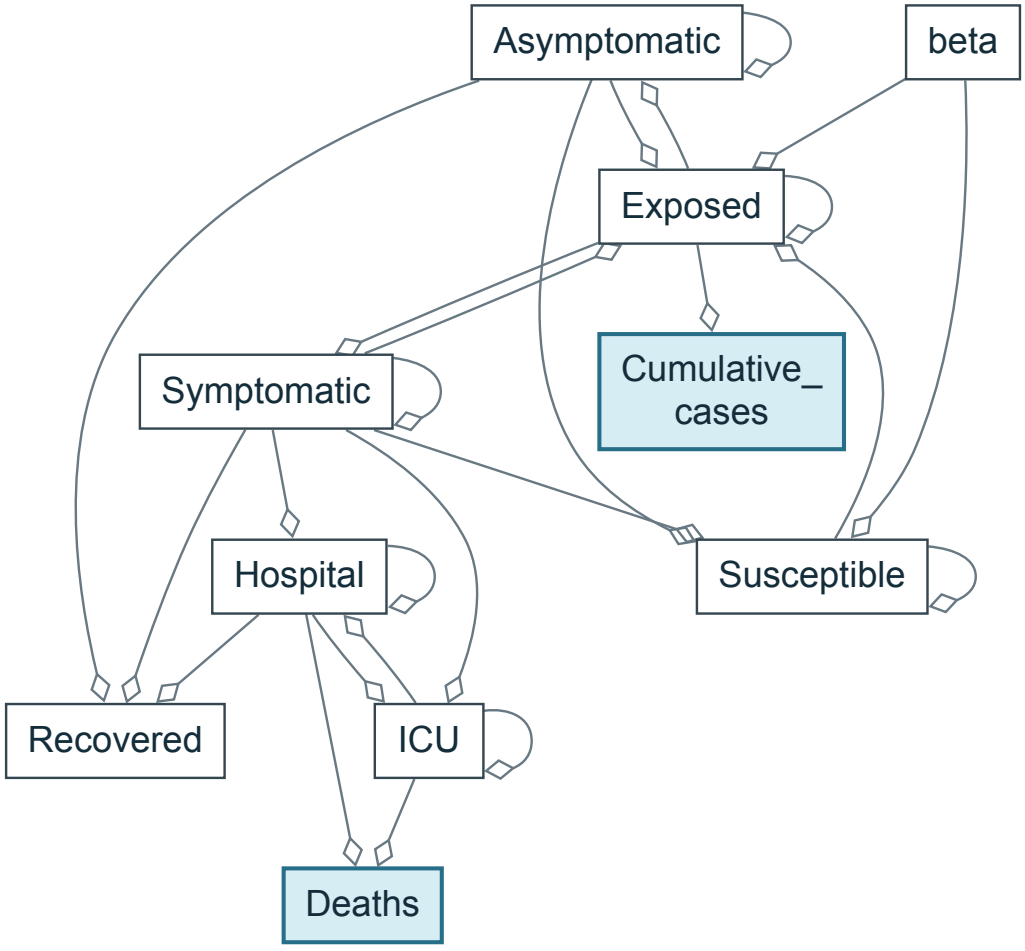

Figure 42: **Oliveira reference model.** SBGN AF abstraction of `Oliveira_NatCommun2021`: 10 nodes and 25 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

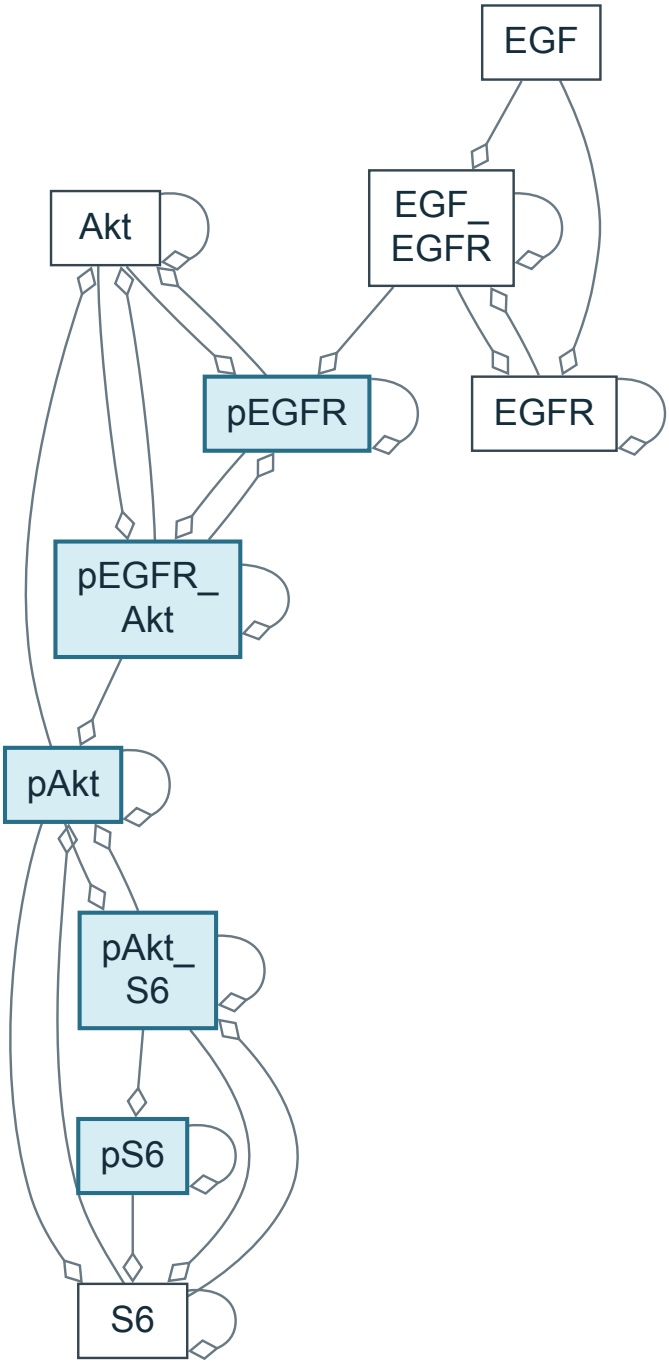

Figure 43: **Fujita reference model.** SBGN AF abstraction of `Fujita_SciSignal2010`: 10 nodes and 30 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

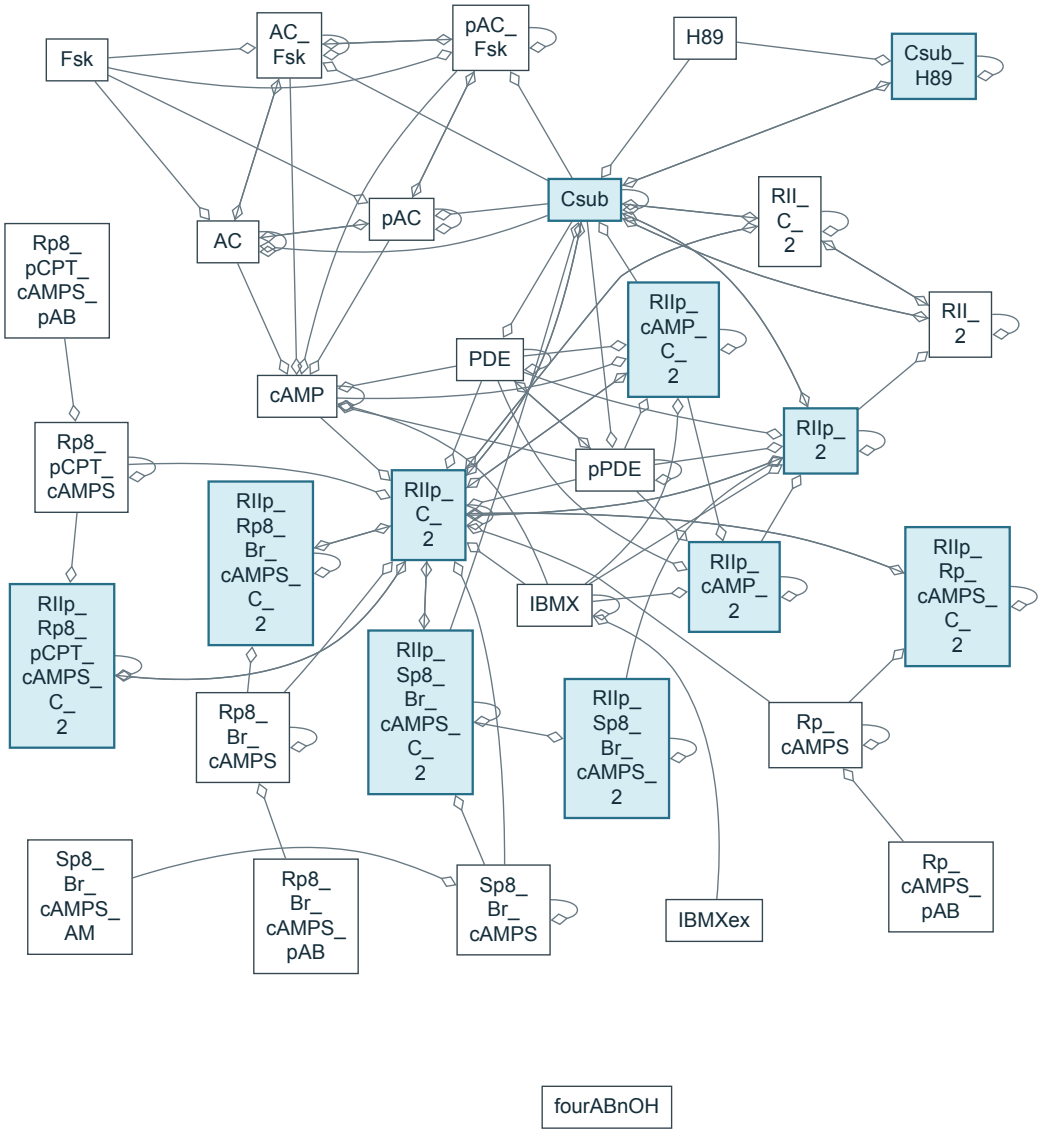

Figure 44: **Isensee reference model.** SBGN AF abstraction of `Isensee_JCB2018`: 33 nodes and 114 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

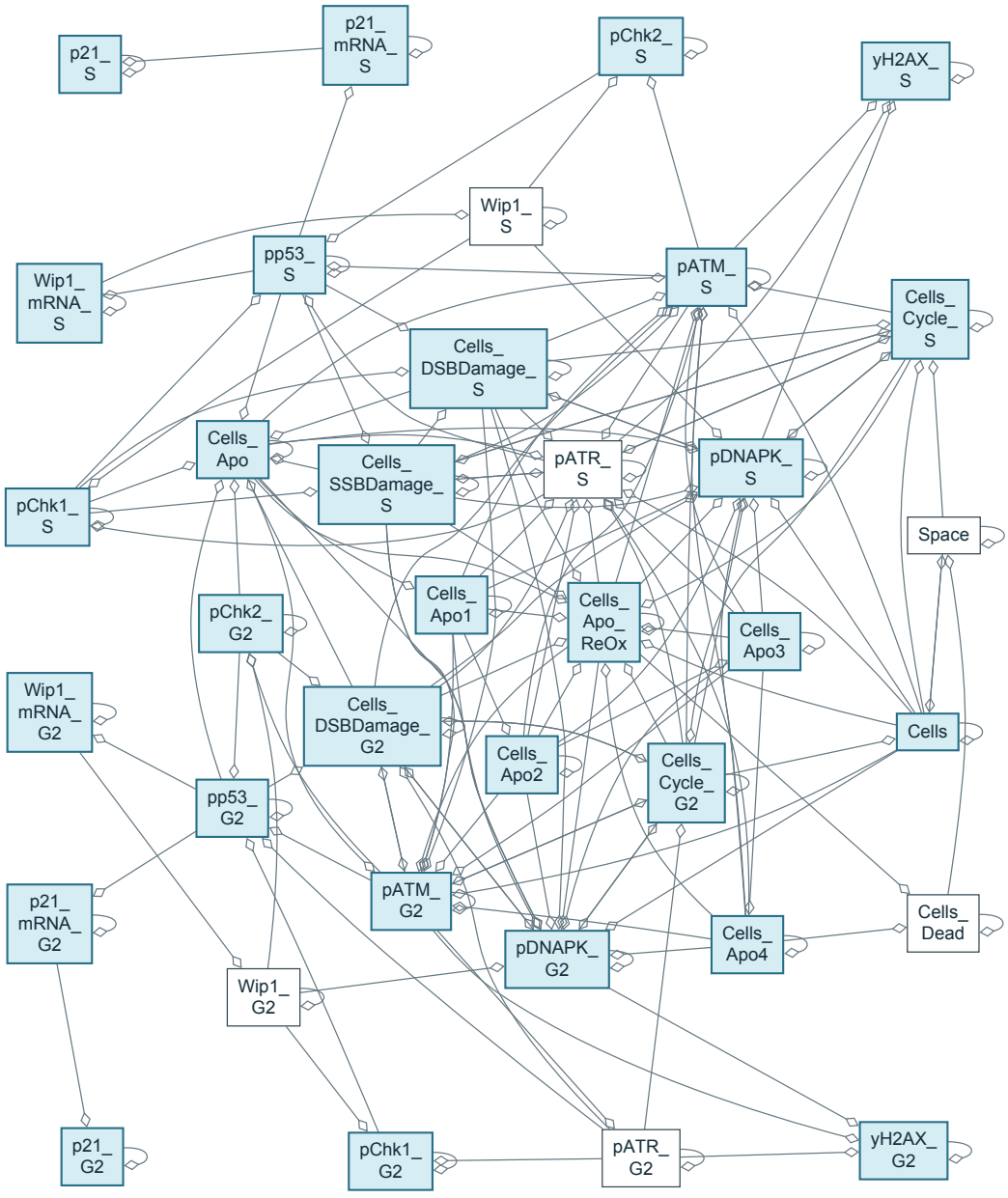

Figure 45: **Alkan reference model.** SBGN AF abstraction of `Alkan_SciSignal2018`: 36 nodes and 181 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

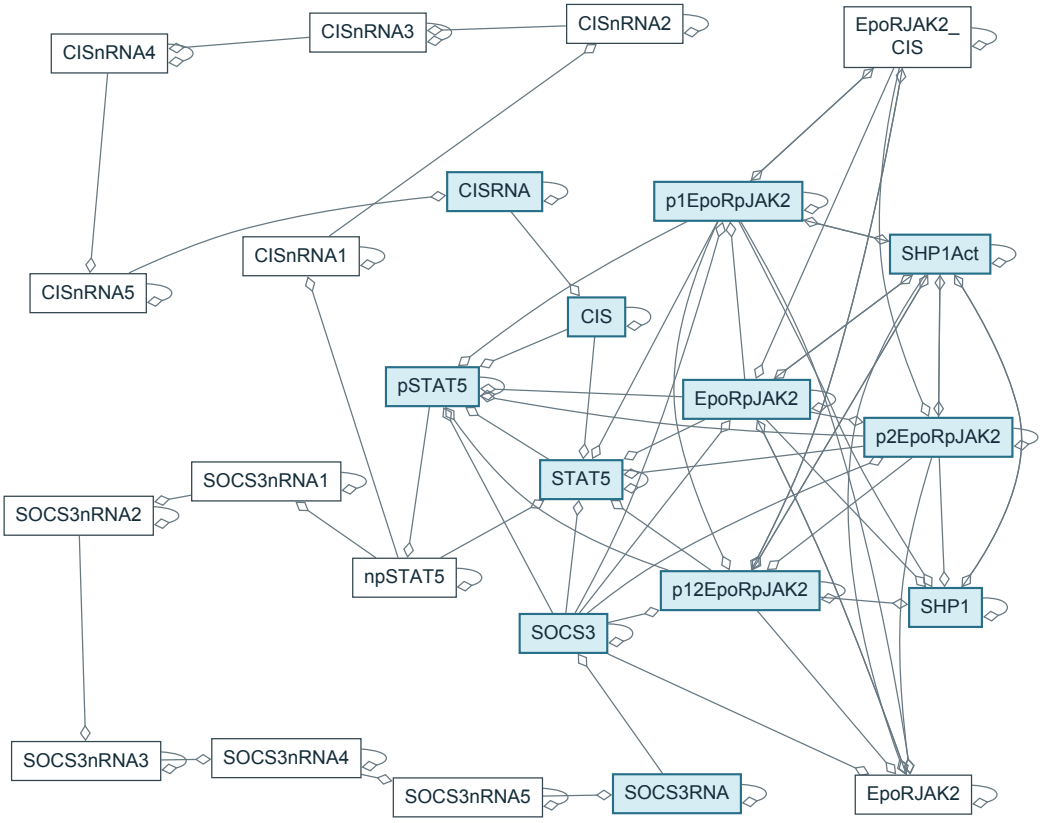

Figure 46: **Bachmann reference model.** SBGN AF abstraction of `Bachmann_MSB2011`: 25 nodes and 89 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

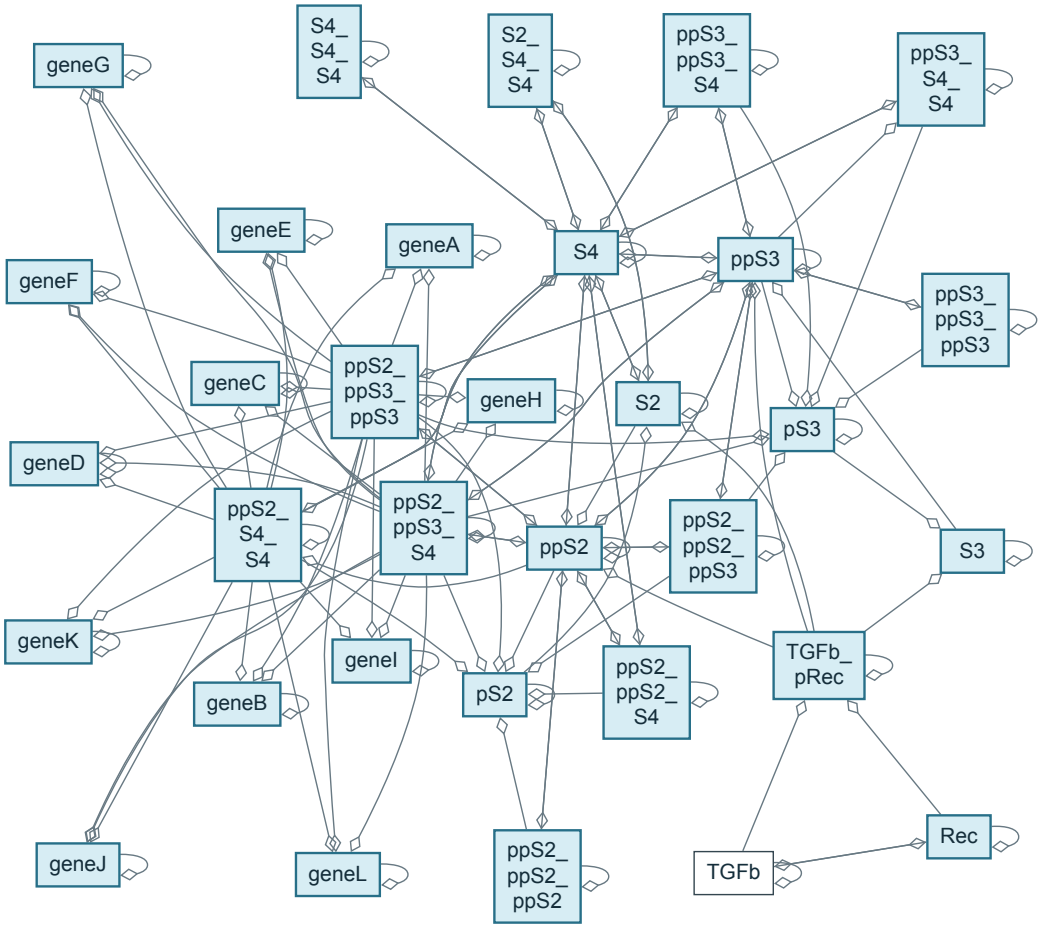

Figure 47: **Lucarelli reference model.** SBGN AF abstraction of `Lucarelli_CellSystems2018`: 33 nodes and 141 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

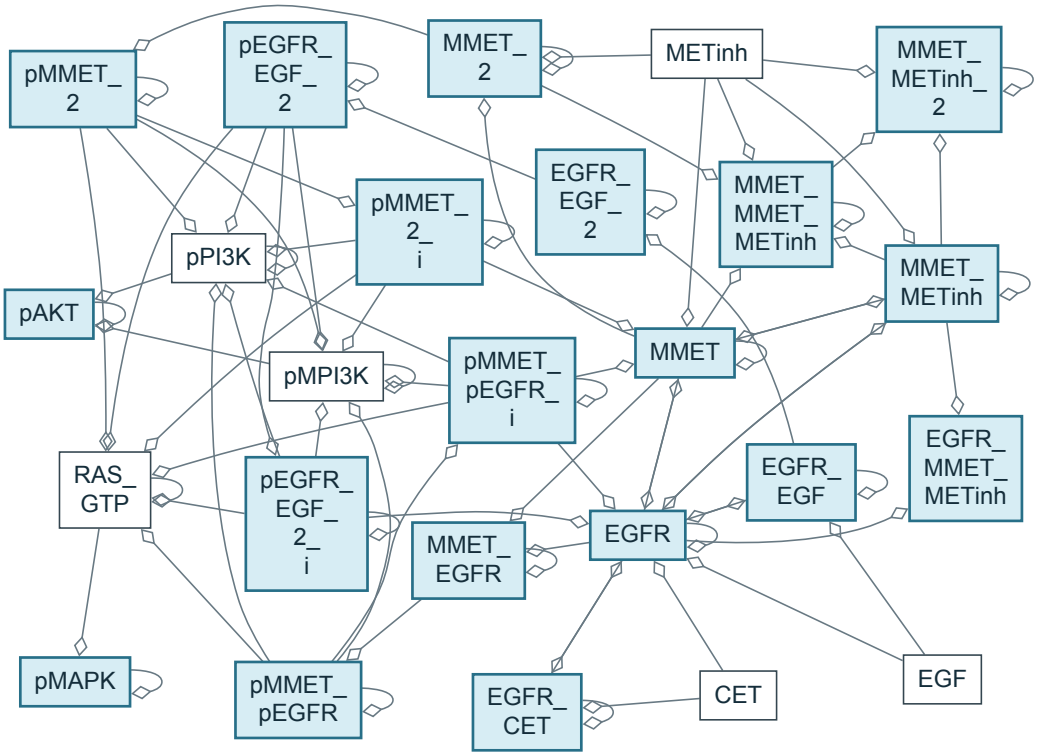

Figure 48: **Raimundez reference model.** SBGN AF abstraction of `Raimundez_PCB2020`: 25 nodes and 82 dependency arcs. Blue nodes contribute to observable formulas; diamonds leave influence signs unspecified.

G AGENT-GENERATED FUJITA MODEL SOURCES

The following listings contain the archived Antimony for the 28 models of Figure 3 whose sources were exported with their hashes. The label, arm, array task and selected version identify each source. Listings marked *converted from the archived SBML* are libantimony renderings of the `model.xml` those runs delivered: the bare-agent arms wrote no Antimony, so none exists to reproduce. The hash recorded for such a row is their `model.xml`, because verifying a derived listing against itself would prove nothing; that SBML is not shipped, so the annex is Antimony throughout. Every other listing is the archived file itself, preserved exactly, including anonymous identifiers and unset initializations. Numeric assignments in these source listings are not necessarily the fitted values. The matching parameter tables (including observation scales and fixed entries) and SHA-256 hashes are exported alongside them; see `fujita_models.csv`. These sources and tables document the selected structures and parameters; reproducing simulations also requires the original PEtab conditions, observables and initialization semantics. No parameters were refitted for this annex.

Listing 1: Archived Antimony for O1a: Panel, glm-5.3-flash; task `56476116_0`; selected version 22

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6_pathway()

  // Compartments and Species:
  compartment cell;
  species R in cell, C in cell, pR in cell, PIP3 in cell, Akt in cell,
    pAkt in cell;
  species S6K in cell, pS6K in cell, S6 in cell, pS6 in cell, pRi;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := pR + pRi;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  reaction_1: R -> C; k_on*EGF*R;
  r_aut: C -> pR; k_aut*(C/R_init)^n_a/(K_a^n_a + (C/R_init)^n_a);
  r_deR: pR -> C; k_deR*pR;
  r_int: C -> ; k_int*C;
  r_pip3:  -> PIP3; (pR + f_pRi*pRi)/(K_p3k + pR + f_pRi*pRi);
  r_pA: Akt -> pAkt; k_pA*PIP3*Akt;
  r_dA: pAkt -> Akt; k_dA*pAkt;
  r_pSK: S6K -> pS6K; (k_pSK*pAkt^n_SK/(K_Akt^n_SK + pAkt^n_SK))*S6K;
  r_dSK: pS6K -> S6K; k_dSK*pS6K;
  r_pS6: S6 -> pS6; (k_pS6*pS6K^n_S6/(1 + pS6K^n_S6))*S6;
  r_dS6: pS6 -> S6; k_dS6*pS6;
  r_deRi: pRi -> ; k_deRi*pRi;
  r_intP: pR -> pRi; k_intP*pR;
  r_sink: PIP3 -> ; k_dep*pS6K^n_dep*PIP3;

  // Species initializations:
  R = R_init;
  C = 0;
  pR = 0;
  PIP3 = 0;
  Akt = Akt_init;
  pAkt = 0;
  S6K = S6K_init;
  pS6K = 0;
  S6 = S6_init;
  pS6 = 0;
  pRi = ;

  // Compartment initializations:
  cell = 1;
```

```
  // Variable initializations:
  R_init = ;
  Akt_init = 0.03;
  S6K_init = 1;
  S6_init = 2e-05;
  EGF_0 = 0.1;
  EGF_rate = 0;
  EGF_end = 3600;
  k_on = 0.01;
  k_aut = 0.1;
  k_deR = 0.001;
  k_int = 0.0007;
  K_p3k = 1000000;
  k_pA = 0.001;
  k_dA = 0.0006;
  k_pSK = 0.1;
  k_dSK = 0.0005;
  k_pS6 = 0.005;
  k_dS6 = 0.0003;
  n_a = ;
  K_a = ;
  f_pRi = ;
  n_SK = ;
  K_Akt = ;
  n_S6 = ;
  k_deRi = ;
  k_intP = ;
  k_dep = ;
  n_dep = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, R_init, Akt_init, S6K_init, S6_init, EGF_0, EGF_rate,
    EGF_end;
  const k_on, k_aut, k_deR, k_int, K_p3k, k_pA, k_dA, k_pSK, k_dSK, k_pS6
    ;
  const k_dS6, n_a, K_a, f_pRi, n_SK, K_Akt, n_S6, k_deRi, k_intP, k_dep;
  const n_dep;
end
```

Listing 2: Archived Antimony for O1b: Panel, glm-5.3-flash; task `56476116_1`; selected version 8

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cell;
species EGFR in cell, pEGFR in cell, pAkt in cell, S6Kp in cell, pS6 in
    cell;
species p_B;

// Assignment Rules:
EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
readout_pEGFR_tot := R_pool*(pEGFR + p_B);
readout_pAkt_tot := pAkt;
readout_pS6_tot := pS6;

// Reactions:
egfr_activation: EGFR -> pEGFR; k_ri*(EGF/(Kd_EGF + EGF))^n_act*EGFR;
egfr_dephos: pEGFR -> EGFR; k2*pEGFR;
akt_activation:  -> pAkt; k_akt*(pEGFR + p_B)*(1 - pAkt)*(1 - inh_fbk*
    S6Kp/(Ki_fbk + S6Kp));
akt_dephos: pAkt -> ; k_deakt*pAkt;
s6k_activation:  -> S6Kp; k_s6k*pAkt*(1 - S6Kp);
s6k_deact: S6Kp -> ; k_ds6k*S6Kp;
```

```
s6_activation:  -> pS6; k_s6*S6Kp;
s6_dephos: pS6 -> ; k_ds6*pS6;
egfrA_internalize: pEGFR => ; k_ri*(EGF/(Kd_EGF + EGF))^n_int*pEGFR;
egfrB_activation:  => p_B; k1B*(EGF/(Kd_EGF + EGF))^n_act*(fB*exp(-
    k_const*time) - p_B);
egfrB_dephos: p_B => ; k2*p_B;

// Species initializations:
EGFR = EGFR_init;
pEGFR = 0;
pAkt = 0;
S6Kp = 0;
pS6 = 0;
p_B = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
EGFR_init = 1;
EGF_0 = 0.1;
EGF_rate = 0;
EGF_end = 3600;
Kd_EGF = 2;
n_act = 2;
k2 = 0.001;
k_akt = 0.1;
Ki_fbk = 0.1;
k_deakt = 0.003;
k_s6k = 0.01;
k_ds6k = 0.003;
k_s6 = 0.002;
k_ds6 = 0.0003;
R_pool = ;
k_ri = ;
inh_fbk = ;
n_int = ;
k1B = ;
fB = ;
k_const = ;

// Other declarations:
var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
const cell, EGFR_init, EGF_0, EGF_rate, EGF_end, Kd_EGF, n_act, k2, k_akt
    ;
const Ki_fbk, k_deakt, k_s6k, k_ds6k, k_s6, k_ds6, R_pool, k_ri, inh_fbk;
const n_int, k1B, fB, k_const;
```

Listing 3: Archived Antimony for O1c: Panel, glm-5.3-flash; task `56476116_2`; selected version 28

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6_cascade()

  // Compartments and Species:
  compartment cell;
  species R_s in cell, pE_s in cell, pE_i in cell, R_i in cell, pAkt in
    cell;
  species mTORC1 in cell, S6K in cell, pS6 in cell, pIRS1 in cell, X,
    S6K3;
  species S6K2, pE_p;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  pE_tot := pE_s + pE_i;
```

```
readout_pEGFR_tot := pE_s + pE_i + pE_p;
readout_pAkt_tot := pAkt;
readout_pS6_tot := pS6;

// Reactions:
egfr_act: R_s -> pE_s; k1*EGF*R_s;
egfr_int: pE_s -> pE_i; (kint*EGF/(Kint + EGF))*pE_s;
egfr_idephos: pE_i -> R_i; kint*pE_i;
akt_act:  -> pAkt; k4*X*(1 - pAkt)/(1 + alpha*pIRS1);
akt_dephos: pAkt -> ; kda*pAkt;
mtor_act:  -> mTORC1; k5*pAkt*(1 - mTORC1);
mtor_deact: mTORC1 -> ; k6*mTORC1;
s6k_act:  -> S6K; k7*mTORC1*(1 - S6K);
s6k_deact: S6K -> ; k6*S6K;
s6_phos:  -> pS6; k7*S6K3;
s6_dephos: pS6 -> ; k8*pS6;
irs1_phos:  -> pIRS1; k9*mTORC1*(1 - pIRS1);
egfr_recycle: R_i -> R_s; krec*R_i;
r_deg: R_i -> ; kdeg*R_i;
s6k2_act:  -> S6K2; k7*S6K*(1 - S6K2);
s6k2_deact: S6K2 -> ; k6*S6K2;
pE_prot_fill: pE_i -> pE_p; kf*pE_i;
pE_prot_out: pE_p -> R_i; kout*pE_p;
pip3_prod:  -> X; (kPI3K*pE_tot^mA/(KAa^mA + pE_tot^mA))*(1 - X);
pip3_loss: X -> ; kPI3Koff*X;
s6k3_act:  -> S6K3; k7*S6K2*(1 - S6K3);
s6k3_deact: S6K3 -> ; k6*S6K3;

// Species initializations:
R_s = R_s_init;
pE_s = 0;
pE_i = 0;
R_i = 0;
pAkt = 0;
mTORC1 = 0;
S6K = 0;
pS6 = 0;
pIRS1 = 0;
X = ;
S6K3 = ;
S6K2 = ;
pE_p = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
R_s_init = 30000000;
EGF_0 = 1;
EGF_rate = 0;
EGF_end = 3600;
k1 = 0.0001;
kint = 0.002;
kdeg = 0.001;
k4 = 1;
mA = 1.7;
KAa = 400000;
kda = 0.012;
alpha = 385;
k5 = 0.007;
k6 = 0.0037;
k7 = 1.5e-06;
k8 = 0.0012;
k9 = 0.0008;
Kint = ;
```

```
  krec = ;
  kf = ;
  kout = ;
  kPI3K = ;
  kPI3Koff = ;

  // Other declarations:
  var EGF, pE_tot, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, R_s_init, EGF_0, EGF_rate, EGF_end, k1, kint, kdeg, k4, mA;
  const KAa, kda, alpha, k5, k6, k7, k8, k9, Kint, krec, kf, kout, kPI3K;
  const kPI3Koff;
end
```

Listing 4: Archived Antimony for O1d: Panel, glm-5.3-flash; task `56476116_3`; selected version 29

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species Rec in cell, pEGFR in cell, D in cell, Akt in cell, pAkt in
    cell;
  species S6 in cell, pS6 in cell, pS6K in cell, IRS1i, M, RE;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  Rec_tot := Rec + pEGFR + D;
  S6_tot := S6 + pS6;
  readout_pEGFR_tot := pEGFR;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  egfr_dephos: pEGFR => Rec; k_de*pEGFR;
  akt_phos: Akt => pAkt; Vmax_ak*(pEGFR/Rec_tot)*(1 - IRS1i)*Akt;
  akt_dephos: pAkt => Akt; kd_ak*pAkt;
  s6k_act:  -> pS6K; k_M*M*(1 - pS6K);
  s6k_deact: pS6K -> ; kd_s6*pS6K;
  s6_phos: S6 => pS6; ka_s6*pS6K*S6;
  s6_dephos: pS6 => S6; kd_s6*pS6;
  egfr_int: Rec => D; (k_int*EGF/(Km_int + EGF))*Rec;
  d_deg: D => ; k_deg*D;
  egfr_intp: pEGFR => D; (k_intp*EGF/(Km_int + EGF))*pEGFR;
  mtorc1_act:  -> M; (Vmax_M*pAkt/(Km_M + pAkt))*(1 - M);
  mtorc1_deact: M => ; kd_M*M;
  irs1_inhib:  -> IRS1i; k_i*pS6K*(1 - IRS1i);
  irs1_recover: IRS1i => ; k_r*IRS1i;
  egfr_bind: Rec => RE; k_on*EGF*Rec;
  egfr_unbind: RE => Rec; k_off*RE;
  egfr_autophos: RE => pEGFR; k_p*RE;

  // Species initializations:
  Rec = Rec_init;
  pEGFR = 0;
  D = 0;
  Akt = Akt_init;
  pAkt = 0;
  S6 = S6_init;
  pS6 = 0;
  pS6K = 0;
  IRS1i = ;
  M = ;
  RE = ;

```

```
  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  Rec_init = 100000;
  Akt_init = 1;
  S6_init = 1;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 3600;
  k_on = 0.0075;
  k_de = 0.0021;
  k_int = 0.0005;
  Vmax_ak = 0.02;
  kd_ak = 0.02;
  ka_s6 = 0.02;
  kd_s6 = 0.0009;
  k_M = ;
  Km_int = ;
  k_deg = ;
  k_intp = ;
  Vmax_M = ;
  Km_M = ;
  kd_M = ;
  k_i = ;
  k_r = ;
  k_off = ;
  k_p = ;

  // Other declarations:
  var EGF, Rec_tot, S6_tot, readout_pEGFR_tot, readout_pAkt_tot,
    readout_pS6_tot;
  const cell, Rec_init, Akt_init, S6_init, EGF_0, EGF_rate, EGF_end, k_on
    ;
  const k_de, k_int, Vmax_ak, kd_ak, ka_s6, kd_s6, k_M, Km_int, k_deg,
    k_intp;
  const Vmax_M, Km_M, kd_M, k_i, k_r, k_off, k_p;
end
```

Listing 5: Archived Antimony for O1e: Panel, glm-5.3-flash; task `56476116_4`; selected version 20

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species R in cell, Rp in cell, D in cell, A in cell, SK in cell, S6 in
    cell;
  species F in cell, U in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  hillR := Rp^n_h/(k_traf^n_h + Rp^n_h);
  hillA := A^n_A/(k_on^n_A + A^n_A);
  readout_pEGFR_tot := (Rp + D)*1000000;
  readout_pAkt_tot := A;
  readout_pS6_tot := S6;

  // Reactions:
  egfr_act: R -> Rp; (k_SK*EGF^n_h/(K_E^n_h + EGF^n_h))*R;
  egfr_dephos: Rp -> R; k_deg*Rp;
  egfr_int: Rp -> D; k_traf*U*Rp;
  egfr_rec: D -> R; k_traf*D;
  egfr_degD: D -> ; k_traf*D;
```

```
  akt_phos:  -> A; k_A*hillR*(1 - inh*F)*(1 - A);
  akt_basal:  -> A; k_bas*(1 - A);
  akt_dephos: A -> ; k_pA*A;
  sk_act:  -> SK; k_SK*hillA*(1 - SK);
  sk_dephos: SK -> ; k_bas*SK;
  s6_phos:  -> S6; k_S*SK^n_SK/(K_SK^n_SK + SK^n_SK);
  s6_dephos: S6 -> ; k_deg*S6;
  fb_act:  -> F; k_on*A*(1 - F);
  u_on:  -> U; k_u*EGF;
  u_off: U -> ; k_bas*U;

  // Species initializations:
  R = R_init;
  Rp = 0;
  D = 0;
  A = 0;
  SK = 0;
  S6 = 0;
  F = 0;
  U = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 1;
  EGF_0 = 0.1;
  EGF_rate = 0;
  EGF_end = 3600;
  n_h = 1.15;
  k_traf = 0.03;
  n_A = 2.5;
  k_on = 0.073;
  k_SK = 0.0041;
  K_E = 11.2;
  k_deg = 0.0026;
  k_A = 0.0258;
  k_pA = 0.0086;
  k_S = 1.48e-06;
  K_SK = 0.72;
  n_SK = 5.65;
  k_u = 0.0009;
  inh = 0.9;
  k_bas = 0.00013;

  // Other declarations:
  var EGF, hillR, hillA, readout_pEGFR_tot, readout_pAkt_tot,
    readout_pS6_tot;
  const cell, R_init, EGF_0, EGF_rate, EGF_end, n_h, k_traf, n_A, k_on,
    k_SK;
  const K_E, k_deg, k_A, k_pA, k_S, K_SK, n_SK, k_u, inh, k_bas;
end
```

Listing 6: Archived Antimony for O2a: Biocurator off; task `56476121_0`; selected version 25

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species R in cell, pEGFR in cell, pEGFRi in cell, Akt in cell, pAkt in
    cell;
  species S6K in cell, S6 in cell, pS6 in cell, FB in cell, F;

  // Assignment Rules:
```

```
EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
readout_pEGFR_tot := (pEGFR + pEGFRi + F)*10000000;
readout_pAkt_tot := pAkt;
readout_pS6_tot := pS6;

// Reactions:
reaction_1: R -> pEGFR; k1*EGF^nH*R;
egfr_dephos: pEGFR -> ; k2*pEGFR/(KD2 + pEGFR);
egfr_internalize: pEGFR -> pEGFRi; k3*pEGFR;
akt_phos: Akt -> pAkt; (k5p*(pEGFR + pEGFRi)^nA/(KAA^nA + (pEGFR +
  pEGFRi)^nA))*Akt/(1 + FB);
akt_dephos: pAkt -> Akt; k6*pAkt;
s6k_act:  -> S6K; (kk2*pAkt^nS/(KSA^nS + pAkt^nS))*(1 - S6K);
s6k_deact: S6K -> ; kk2*S6K;
s6_phos: S6 -> pS6; VS6*S6K*S6;
s6_dephos: pS6 -> S6; k6*pS6/(KD8 + pS6);
fb_syn:  -> FB; kfdeg*GFB*S6K;
fb_deg: FB -> ; kfdeg*FB;
egfr_fast_act: R -> F; k1f*(EGF^nF/(KDF^nF + EGF^nF))*R;
egfr_fast_dephos: F -> R; kf*F;

// Species initializations:
R = R_init;
pEGFR = 0;
pEGFRi = 0;
Akt = Akt_init;
pAkt = 0;
S6K = 0;
S6 = S6_init;
pS6 = 0;
FB = 0;
F = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
R_init = 2.5;
Akt_init = 1;
S6_init = 1;
k1 = 0.001;
nH = 1.2;
k2 = 0.015;
k3 = 0.0013;
k6 = 0.0025;
KSA = 0.3;
kk2 = 0.004;
kf = 0.05;
kfdeg = 0.005;
EGF_0 = 1;
EGF_rate = 0;
EGF_end = 3600;
KD2 = ;
k5p = ;
nA = ;
KAA = ;
nS = ;
VS6 = ;
KD8 = ;
GFB = ;
k1f = ;
nF = ;
KDF = ;

// Other declarations:
```

```
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, R_init, Akt_init, S6_init, k1, nH, k2, k3, k6, KSA, kk2, kf
    ;
  const kfdeg, EGF_0, EGF_rate, EGF_end, KD2, k5p, nA, KAA, nS, VS6, KD8;
  const GFB, k1f, nF, KDF;
end
```

Listing 7: Archived Antimony for O2b: Biocurator off; task `56476121_1`; selected version 23

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cell;
species R in cell, pR in cell, A in cell, pA in cell, I in cell, pI in
    cell;
species S in cell, pS in cell, pRi, pA1, Ds, pI1;

// Assignment Rules:
EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
readout_pEGFR_tot := (pR + w_pRi*pRi)*10000;
readout_pAkt_tot := pA;
readout_pS6_tot := pS*0.01;

// Reactions:
reaction_1: R => pR; k1*EGF^n_EGF*R;
akt_dephos: pA => A; k_deA*pA;
s6k_dephos: pI => I; k_deI*pI;
s6_activation: S => pS; k_actS*pI*S;
s6_dephos: pS => S; k_deI*pS;
egfr_internalization: pR => pRi; k_int*pR;
egfr_endo_deg: pRi => ; k_deg*pRi;
akt_activation: A => pA1; k_actA*A*exp(-k_des*Ds)*pR/(KM_pR + pR);
akt_maturation: pA1 => pA; k_deA*pA1;
akt_desensitization:  => Ds; pRi;
s6k_activation: I => pI1; pA*I;
s6k_maturation2: pI1 => pI; k_pI1*pI1;
s6k_primed_dephos: pI1 => I; k_deI*pI1;
akt_reactivation:  => pA1; k_res*pRi*A;

// Species initializations:
R = R_init;
pR = 0;
A = A_init;
pA = 0;
I = I_init;
pI = 0;
S = S_init;
pS = 0;
pRi = ;
pA1 = ;
Ds = ;
pI1 = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
R_init = 1;
A_init = 1;
I_init = 1;
S_init = 1;
EGF_0 = 0.1;
EGF_rate = 0;
EGF_end = 3600;
k1 = 0.002;
k_int = 0.004;
```

```
k_actA = 0.05;
k_deA = 0.002;
k_deI = 0.002;
k_actS = 0.0015;
w_pRi = ;
n_EGF = ;
k_deg = ;
k_des = ;
KM_pR = ;
k_pI1 = ;
k_res = ;

// Other declarations:
var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
const cell, R_init, A_init, I_init, S_init, EGF_0, EGF_rate, EGF_end, k1;
const k_int, k_actA, k_deA, k_deI, k_actS, w_pRi, n_EGF, k_deg, k_des,
    KM_pR;
const k_pI1, k_res;
```

Listing 8: Archived Antimony for O2c: Biocurator off; task `56476121_2`; selected version 17

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cell;
species R in cell, pR in cell, D in cell, A in cell, pA in cell, K1 in
    cell;
species pK1 in cell, K2 in cell, pK2 in cell, S in cell, pS in cell, I in
     cell;

// Assignment Rules:
EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
fE := pR/(R + pR);
fA := pA/(A + pA);
fK1 := pK1/(K1 + pK1);
fK2 := pK2/(K2 + pK2);
readout_pEGFR_tot := pR;
readout_pAkt_tot := pA;
readout_pS6_tot := pS;

// Reactions:
egfr_act: R => pR; k1*EGF^hE*R;
egfr_fb_dephos: pR => R; k3fb*I*pR;
egfr_int: pR => D; k3*EGF*pR;
adapter_act:  => I; pR/(R + pR);
akt_act: A => pA; (k2a*fE^hA/(KmR^hA + fE^hA))*A + (k2s*I^hI/(KI^hI + I^
    hI))*A;
akt_dephos: pA => A; k3a*pA;
akt_fb_dephos: pA => A; k4fb*I*pA;
s6k1_act: K1 => pK1; k5a*fA*K1;
s6k1_dephos: pK1 => K1; k6m*pK1;
s6k2_act: K2 => pK2; k5b*fK1*K2;
s6k2_dephos: pK2 => K2; k6m*pK2;
s6_act: S => pS; k7a*fK2*S;
s6_dephos: pS => S; k8a*pS;

// Species initializations:
R = R_init;
pR = 0;
D = 0;
A = A_init;
pA = 0;
K1 = K1_init;
pK1 = 0;
K2 = K2_init;
pK2 = 0;
```

```
S = S_init;
pS = 0;
I = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
R_init = 20000000;
A_init = 1;
K1_init = 1;
K2_init = 1;
S_init = 0.001;
k1 = 0.001;
hE = 2;
k3fb = 0.001;
k3 = 0.0001;
k2a = 0.3;
KmR = 0.05;
k3a = 0.003;
k4fb = 0.003;
k5a = 0.002;
k5b = 0.002;
k7a = 0.005;
k8a = 0.00025;
EGF_0 = 1;
EGF_rate = 0;
EGF_end = 3600;
hA = ;
k2s = ;
hI = ;
KI = ;
k6m = ;

// Other declarations:
var EGF, fE, fA, fK1, fK2, readout_pEGFR_tot, readout_pAkt_tot,
    readout_pS6_tot;
const cell, R_init, A_init, K1_init, K2_init, S_init, k1, hE, k3fb, k3,
    k2a;
const KmR, k3a, k4fb, k5a, k5b, k7a, k8a, EGF_0, EGF_rate, EGF_end, hA,
    k2s;
const hI, KI, k6m;
```

Listing 9: Archived Antimony for O2d: Biocurator off; task `56476121_3`; selected version 3

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species R in cell, Rp in cell, pAkt in cell, H in cell, pS6 in cell,
    Rpi;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := Rp + Rpi;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  egf_binding: R => Rp; k_bind*EGF*R + k_auto*EGF*R*Rp;
  rec_dephos: Rp => R; k_deP*Rp;
  akt_act:  => pAkt; k_actR*Rp^n_A/(Km_R^n_A + Rp^n_A) + k_actI*Rpi;
  akt_dephos: pAkt => ; k_deA*pAkt;
  s6k_act:  => H; k_actH*pAkt^n_H/(Km_H^n_H + pAkt^n_H);
```

```
  s6k_deact: H => ; k_deH*H;
  s6_act:  => pS6; k_actS*H;
  s6_dephos: pS6 => ; k_deS*pS6;
  rec_int: Rp => Rpi; k_int*Rp;
  ri_loss: Rpi => ; k_ri_deg*(1 + fc*EGF)*Rpi;

  // Species initializations:
  R = R_init;
  Rp = 0;
  pAkt = 0;
  H = 0;
  pS6 = 0;
  Rpi = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 200000;
  EGF_0 = 0.1;
  EGF_rate = 0;
  EGF_end = 3600;
  k_bind = 0.001;
  k_deP = 0.001;
  k_actR = 0.005;
  Km_R = 10000;
  k_deA = 0.02;
  k_actH = 0.002;
  k_deH = 0.0015;
  k_actS = 1e-06;
  k_deS = 0.0002;
  k_int = 0.0005;
  k_auto = ;
  n_A = ;
  k_actI = ;
  n_H = ;
  Km_H = ;
  k_ri_deg = ;
  fc = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, R_init, EGF_0, EGF_rate, EGF_end, k_bind, k_deP, k_actR,
    Km_R;
  const k_deA, k_actH, k_deH, k_actS, k_deS, k_int, k_auto, n_A, k_actI,
    n_H;
  const Km_H, k_ri_deg, fc;
end
```

Listing 10: Archived Antimony for O2e: Biocurator off; task `56476121_4`; selected version 25

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species Rfree in cell, Rp in cell, Ri in cell, A0 in cell, Ap in cell,
    S6Kp in cell;
  species S6p in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := Rp;
  readout_pAkt_tot := Ap;
  readout_pS6_tot := S6p;
```

```
  // Reactions:
  egfr_activation: Rfree -> Rp; kon*EGF^n_hill*Rfree;
  egfr_dephos: Rp -> Rfree; kdep*Rp;
  egfr_internalization: Rp -> Ri; kint*Rp;
  akt_activation: A0 -> Ap; k3*A0*(Rp/K_A)^m_A/(1 + (Rp/K_A)^m_A)/(1 + (
    S6Kp/Kfb)^mfb);
  akt_dephos: Ap -> A0; k4*Ap;
  s6k_activation:  -> S6Kp; k5*Ap;
  s6k_dephos: S6Kp -> ; k6*S6Kp;
  s6_activation:  -> S6p; S6Kp/(1 + (S6p/Ks6)^m_s6);
  s6_dephos: S6p -> ; k8*S6p;
  rec: Ri -> Rfree; krec*Ri;

  // Species initializations:
  Rfree = Rfree_init;
  Rp = 0;
  Ri = 0;
  A0 = A0_init;
  Ap = 0;
  S6Kp = 0;
  S6p = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  Rfree_init = 20000000;
  A0_init = 1;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 3600;
  kon = 1.2e-10;
  kdep = 0.02;
  kint = 0.0035;
  k3 = 2.4e-09;
  k4 = 0.002;
  k5 = 1e-09;
  k6 = 0.0015;
  k8 = 0.0005;
  n_hill = ;
  K_A = ;
  m_A = ;
  Kfb = ;
  mfb = ;
  Ks6 = ;
  m_s6 = ;
  krec = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, Rfree_init, A0_init, EGF_0, EGF_rate, EGF_end, kon, kdep,
    kint;
  const k3, k4, k5, k6, k8, n_hill, K_A, m_A, Kfb, mfb, Ks6, m_s6, krec;
end
```

Listing 11: Archived Antimony for O3b: Null arm, glm-5.3-flash; task `56550502_1`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species R, Rp, Rp2, I, Akt, pAkt, S6K, S6, pS6, I2;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := Rp + Rp2;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  reaction_1: R -> Rp; k_act*EGF^h_act*R;
  reaction_2: Rp -> R; k_de*Rp;
  reaction_3: Rp -> Rp2; k_int*(EGF/(Ki + EGF))*Rp;
  reaction_4: Rp2 -> ; k_intd*Rp2;
  reaction_5:  -> I; kp*Rp;
  reaction_6: I -> ; kpl*I;
  reaction_6b:  -> I2; k2p*Rp2;
  reaction_6c: I2 -> ; k2l*I2;
  reaction_7: Akt -> pAkt; ka*(I + I2)*Akt;
  reaction_8: pAkt -> Akt; kd*pAkt;
  reaction_9:  -> S6K; ksk*pAkt;
  reaction_10: S6K -> ; kskd*S6K;
  reaction_11: S6 -> pS6; ks*S6K*S6;
  reaction_12: pS6 -> S6; ksd*pS6;

  // Species initializations:
  R = x0_R;
  Rp = 0;
  Rp2 = 0;
  I = 0;
  Akt = x0_Akt;
  pAkt = 0;
  S6K = 0;
  S6 = x0_S6;
  pS6 = 0;
  I2 = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  x0_R = 1000000;
  x0_Akt = 1;
  x0_S6 = 1;
  EGF_0 = 1;
  EGF_rate = 0;
  EGF_end = 3600;
  k_act = 1e-09;
  h_act = 1.4;
  k_de = 0.01;
  k_int = 0.001;
  Ki = 3;
  k_intd = 0.0005;
  kp = 1e-07;
  kpl = 0.01;
  k2p = 1e-05;
  k2l = 0.005;
  ka = 1;
```

```
  kd = 0.003;
  ksk = 0.001;
  kskd = 0.002;
  ks = 0.01;
  ksd = 0.0002;
  KE = 3;
  Kf = 0.1;
  KmS = 0.5;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, x0_R, x0_Akt, x0_S6, EGF_0, EGF_rate, EGF_end, k_act, h_act
    ;
  const k_de, k_int, Ki, k_intd, kp, kpl, k2p, k2l, ka, kd, ksk, kskd, ks
    ;
  const ksd, KE, Kf, KmS;
end
```

Listing 12: Archived Antimony for O3c: Null arm, glm-5.3-flash; task `56550502_2`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *EGFR_AKT_S6_v8()

  // Compartments and Species:
  compartment cell;
  species EGFR, E, Rp, Ri, Y, A1, A2, M, K, S6, S6p, S6i;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := Rp + fRi*Ri;
  readout_pAkt_tot := A1 + A2;
  readout_pS6_tot := S6p;

  // Reactions:
  reaction_1: EGFR -> E; k1*EGF*EGFR;
  Runb: E -> EGFR; kunb*E;
  Ract: E -> Rp; k2*E*E/(KhE2 + E*E);
  Rde: Rp -> EGFR; k3*Rp;
  Rint: Rp -> Ri; kint*Rp;
  Rrec: Ri -> EGFR; krec*Ri;
  Rdeg: Ri -> ; kdeg*Ri;
  Ry:  -> Y; k5y*(Rp + Ri)*(1 - Y)*(1 - S6i);
  Ryd: Y -> ; k6y*Y;
  Ra1:  -> A1; k7a*Rp*(1 - A1 - A2);
  Ra1d: A1 -> ; k8a*A1;
  Ra2:  -> A2; k7b*Y*(1 - A1 - A2);
  Ra2d: A2 -> ; k8b*A2;
  Rm:  -> M; k9m*(A1 + A2)*(1 - M);
  Rmd: M -> ; k10m*M;
  Rk:  -> K; (k9k*M^nK/(Kmk^nK + M^nK))*(1 - K);
  Rkd: K -> ; k10k*K;
  Rs6p: S6 -> S6p; k11s*K*S6;
  Rs6d: S6p -> S6; k12s*S6p;
  Rfb:  -> S6i; (k13*K/(Ki + K))*(1 - S6i);
  Rfbd: S6i -> ; k14*S6i;

  // Species initializations:
  EGFR = EGFR_0;
  E = 0;
  Rp = 0;
  Ri = 0;
  Y = 0;
  A1 = 0;
  A2 = 0;
```

```
  M = 0;
  K = 0;
  S6 = 1;
  S6p = 0;
  S6i = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGFR_0 = 1000000;
  EGF_0 = 1;
  EGF_rate = 0;
  EGF_end = 3600;
  k1 = 0.0025;
  kunb = 0.01;
  k2 = 10;
  KhE2 = 100000000;
  k3 = 0.01;
  kint = 0.01;
  krec = 0.001;
  kdeg = 0.0001;
  k5y = 1e-06;
  k6y = 0.005;
  k7a = 2e-07;
  k8a = 0.005;
  k7b = 0.001;
  k8b = 0.005;
  k9m = 0.002;
  k10m = 0.001;
  k9k = 0.002;
  nK = 2;
  Kmk = 0.0001;
  k10k = 0.001;
  k11s = 0.005;
  k12s = 0.001;
  k13 = 1;
  Ki = 0.0001;
  k14 = 0.05;
  fRi = 0.1;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_0, EGF_0, EGF_rate, EGF_end, k1, kunb, k2, KhE2, k3,
    kint;
  const krec, kdeg, k5y, k6y, k7a, k8a, k7b, k8b, k9m, k10m, k9k, nK, Kmk
    ;
  const k10k, k11s, k12s, k13, Ki, k14, fRi;
end
```

Listing 13: Archived Antimony for O3d: Null arm, glm-5.3-flash; task `56550502_3`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *EGFR_AKT_S6()

  // Compartments and Species:
  species EGFR, C, pEGFR, I, AKT, pAKT, pAKT2, S6K, pS6K, S6, pS6;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := pEGFR;
  readout_pAkt_tot := pAKT + pAKT2;
  readout_pS6_tot := pS6;

```

```
// Reactions:
R1: EGFR -> C; k_bind*EGF^h_coop*EGFR;
R2: C -> EGFR; k_off*C;
R3: C -> pEGFR; k_autophos*C;
R4: pEGFR -> C; k_dephosR*pEGFR;
R5: C -> I; k_int*C;
R6: I -> EGFR; k_rec*I;
R7: I -> ; k_deg*I;
R8: AKT -> pAKT; k_AKTphos*(pEGFR/(Km_EGFR + pEGFR))*(Ki_S6K/(Ki_S6K +
  pS6K))*AKT;
R9: pAKT -> pAKT2; k_AKT2phos*(pEGFR/(Km_EGFR + pEGFR))*(Ki_S6K/(Ki_S6K
   + pS6K))*pAKT;
R10: pAKT -> AKT; k_AKTdephos*pAKT;
R11: pAKT2 -> AKT; k_AKT2dephos*pAKT2;
R12: S6K -> pS6K; k_S6Kact*((pAKT + pAKT2)^h_S6K/(Km_AKT^h_S6K + (pAKT
  + pAKT2)^h_S6K))*S6K;
R13: pS6K -> S6K; k_S6Kdeact*pS6K;
R14: S6 -> pS6; k_S6phos*(pS6K/(Km_S6K + pS6K))*S6;
R15: pS6 -> S6; k_S6dephos*pS6;

// Species initializations:
EGFR = 400000;
C = 0;
pEGFR = 0;
I = 0;
AKT = 0.1;
pAKT = 0;
pAKT2 = 0;
S6K = 1;
pS6K = 0;
S6 = 3e-05;
pS6 = 0;

// Variable initializations:
EGF_0 = 0.1;
EGF_rate = 0;
EGF_end = 3600;
k_bind = 0.002;
k_off = 0.006;
k_autophos = 0.001;
k_dephosR = 0.025;
k_int = 0.005;
k_rec = 0.001;
k_deg = 0.0005;
k_AKTphos = 0.006;
Km_EGFR = 20000;
Ki_S6K = 1e-05;
k_AKTdephos = 0.017;
k_AKT2phos = 0.001;
k_AKT2dephos = 0.001;
k_S6Kact = 1e-06;
Km_AKT = 0.003;
k_S6Kdeact = 0.007;
k_S6phos = 0.009;
Km_S6K = 15;
k_S6dephos = 0.0017;
P_AKT = 0;
h_coop = ;
h_S6K = ;

// Other declarations:
var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
const EGF_0, EGF_rate, EGF_end, k_bind, k_off, k_autophos, k_dephosR,
  k_int;
```

```
  const k_rec, k_deg, k_AKTphos, Km_EGFR, Ki_S6K, k_AKTdephos, k_AKT2phos
    ;
  const k_AKT2dephos, k_S6Kact, Km_AKT, k_S6Kdeact, k_S6phos, Km_S6K,
    k_S6dephos;
  const P_AKT, h_coop, h_S6K;
end
```

Listing 14: Archived Antimony for O4a: Biology blinded; task `58644985_0`; selected version 15

```
// Created by libAntimony v3.1.3
model *entity_001()

  // Compartments and Species:
  species X1, X2, X3, W, X3a, X3b;

  // Assignment Rules:
  stimulus_001 := piecewise(input_002*time + input_001, time <= input_003
    , 0);
  readout_001 := X1;
  readout_002 := X2;
  readout_003 := X3;

  // Reactions:
  drive:  -> X1; d1*(stimulus_001^n1/(kb2^n1 + stimulus_001^n1));
  decay1: X1 -> ; d1*X1;
  inhibit: X1 -> ; kb*X1*W;
  relay:  -> X2; V2*(stimulus_001^n2/(M^n2 + stimulus_001^n2));
  decay2: X2 -> ; d2*X2;
  decay3: X3 -> ; d3a*X3 + d3b*X3*W;
  lagstage:  -> X3a; k3*X1;
  drive_w:  -> W; d3b*(stimulus_001^n_w/(K_w^n_w + stimulus_001^n_w)) +
    qa*(stimulus_001^n_w/(K_w^n_w + stimulus_001^n_w))*W*(1 - W/M);
  decay_w: W -> ; d3b*W;
  inhibit2: X2 -> ; kb2*X2*W;
  transfer3b: X3b => X3; d3b*X3b;
  transfer3ab: X3a => X3b; d3a*X3a;

  // Species initializations:
  X1 = 0;
  X2 = 0;
  X3 = 0;
  W = ;
  X3a = ;
  X3b = ;

  // Variable initializations:
  input_002 = ;
  input_001 = ;
  input_003 = ;
  d1 = ;
  n1 = ;
  kb2 = ;
  kb = ;
  V2 = ;
  n2 = ;
  M = ;
  d2 = ;
  d3a = ;
  d3b = ;
  k3 = ;
  n_w = ;
  K_w = ;
  qa = ;

  // Other declarations:
```

```
  var stimulus_001, readout_001, readout_002, readout_003;
  const input_002, input_001, input_003, d1, n1, kb2, kb, V2, n2, M, d2,
    d3a;
  const d3b, k3, n_w, K_w, qa;
end
```

Listing 15: Archived Antimony for O4b: Biology blinded; task `58644985_1`; selected version 28

```
// Created by libAntimony v3.1.3
model *petab_mcp_entity_001()

  // Compartments and Species:
  species input_001, input_002, input_003, X1, X2, Xm, Xm2, X3, A, AM, XF
    ;

  // Assignment Rules:
  stimulus_001 := piecewise(input_002*time + input_001, time <= input_003
    , 0);
  gF := stimulus_001/(KF + stimulus_001);
  gF2 := stimulus_001^n2/(KF^n2 + stimulus_001^n2);
  readout_001 := X2;
  readout_002 := XF + ki2*X2;
  readout_003 := X3;

  // Reactions:
  prod_X1:  => X1; a1*gF*(1 - AM) + a2;
  deg_X1: X1 => ; d1*X1;
  cas_X2:  => X2; k2*X1;
  deg_X2: X2 => ; (KH + ki2*stimulus_001^q3)*X2;
  adapt_AM:  => AM; ka*X2^n2*(1 - AM);
  prod_XF:  => XF; a2*gF2*(1 - A);
  deg_XF: XF => ; KH*XF;
  adapt_A:  => A; kax*XF*(1 - A);
  relay_Xm:  => Xm; kax*X2^n2/(KH^n2 + X2^n2);
  deg_Xm: Xm => ; ki2*Xm;
  cas_Xm2:  => Xm2; KH*Xm;
  deg_Xm2: Xm2 => ; ki2*Xm2;
  cas_X3:  => X3; KH*Xm2;
  deg_X3: X3 => ; (ki2 + ki3*stimulus_001^q3)*X3;

  // Species initializations:
  input_001 = 0.1;
  input_002 = 0;
  input_003 = 3600;
  X1 = 0;
  X2 = 0;
  Xm = 0;
  Xm2 = 0;
  X3 = 0;
  A = 0;
  AM = 0;
  XF = 0;

  // Variable initializations:
  KF = ;
  n2 = ;
  ki2 = ;
  a1 = ;
  a2 = ;
  d1 = ;
  k2 = ;
  KH = ;
  q3 = ;
  ka = ;
  kax = ;
```

```
  ki3 = ;

  // Other declarations:
  var stimulus_001, gF, gF2, readout_001, readout_002, readout_003;
  const KF, n2, ki2, a1, a2, d1, k2, KH, q3, ka, kax, ki3;
end
```

Listing 16: Archived Antimony for O4c: Biology blinded; task `58644985_2`; selected version 1

```
// Created by libAntimony v3.1.3
// Compartments and Species:
species X1, X2, X3;

// Assignment Rules:
stimulus_001 := piecewise(input_002*time + input_001, time <= input_003,
    0);
readout_001 := X2;
readout_002 := X1;
readout_003 := X3;

// Reactions:
R1:  => X1; k1*stimulus_001^n/(KD^n + stimulus_001^n);
R2: X1 => ; (d1 + ki*X3)*X1;
R3:  => X2; k2*X1;
R4: X2 => ; (d2 + k2b*X3)*X2;
R5:  => X3; k3*X1;
R6: X3 => ; (d3 + kd*X1)*X3;

// Species initializations:
X1 = 0;
X2 = 0;
X3 = 0;

// Variable initializations:
input_001 = 0.1;
input_002 = 0;
input_003 = 3600;
k1 = 30000;
KD = 1.5;
n = 1.4;
d1 = 0.001;
ki = 3000;
k2 = 2e-10;
d2 = 0.0004;
k2b = 6500;
k3 = 2e-17;
d3 = 0.0004;
kd = 1.2e-09;

// Other declarations:
var stimulus_001, readout_001, readout_002, readout_003;
const input_001, input_002, input_003, k1, KD, n, d1, ki, k2, d2, k2b, k3
    ;
const d3, kd;
```

Listing 17: Archived Antimony for O4d: Biology blinded; task `58644985_3`; selected version 29

```
// Created by libAntimony v3.1.3
model *entity_001()

  // Compartments and Species:
  species species_001, species_002, D1, D2, Xb, species_003, Xb2;

  // Assignment Rules:
  stimulus_001 := piecewise(input_002*time + input_001, time <= input_003
    , 0);
  Sact := stimulus_001^hS/(EC50^hS + stimulus_001^hS);
  readout_001 := species_001;
  readout_002 := species_002;
  readout_003 := species_003;

  // Reactions:
  RD1:  -> D1; ka*Sact*(1 - D1) - Xc*D1;
  RD2:  -> D2; ka2*Sact*D1*(1 - D2);
  R2:  -> species_002; k1*Sact*(1 - D1)*(1 - D2) - k2*species_002;
  R1:  -> species_001; k2l*species_002 - k2*species_001*(1 + (species_001
    /Xc)^(h4 - 1));
  Rb: species_001 -> Xb; db*(species_001 - Xb);
  R3:  -> species_003; k3*Xb2 - db*species_003;
  Rb2: Xb -> Xb2; db*(Xb - Xb2);

  // Species initializations:
  species_001 = 0;
  species_002 = 0;
  D1 = 0;
  D2 = 0;
  Xb = 0;
  species_003 = 0;
  Xb2 = ;

  // Variable initializations:
  input_002 = ;
  input_001 = ;
  input_003 = ;
  hS = 0.8414;
  EC50 = 12.24;
  Xc = 0.0002035;
  ka2 = 0.0211;
  k1 = 790391;
  k2 = 0.010855;
  h4 = 4.9903;
  db = 0.0010077;
  k3 = 2.698e-07;
  ka = ;
  k2l = ;

  // Other declarations:
  var stimulus_001, Sact, readout_001, readout_002, readout_003;
  const input_002, input_001, input_003, hS, EC50, Xc, ka2, k1, k2, h4,
    db;
  const k3, ka, k2l;
end
```

Listing 18: Archived Antimony for O4e: Biology blinded; task `58661494_4`; selected version 16

```
// Created by libAntimony v3.1.3
model *entity_001()

  // Compartments and Species:
  species input_001, input_002, input_003, A, X, Z, W, V, S;

  // Assignment Rules:
  readout_001 := X;
  readout_002 := A;
  readout_003 := Z;

  // Reactions:
  A_act:  -> A; P_A*piecewise(1, time <= input_003, 0)*(input_001^hA/(KaA
    ^hA + input_001^hA));
  A_act2:  -> A; k1*input_002*piecewise(1, time <= input_003, 0);
  A_off: A -> ; (kaa*A + kax*X + kzA*Z + ksA*S)*A;
  X_act:  -> X; k1*(A^h1/(KM_A^h1 + A^h1))*(1 - X);
  X_off: X -> ; (kxw + kxa*A + kxw*W^h3)*X;
  W_act:  -> W; X;
  W_off: W -> ; kW*W;
  V_act:  -> V; W;
  V_off: V -> ; koutZ*V;
  Z_act:  -> Z; 1e-8*V;
  Z_off: Z -> ; koutX*Z;
  S_act:  -> S; input_001*piecewise(1, time <= input_003, 0);
  S_off: S -> ; kS*S;

  // Species initializations:
  input_001 = ;
  input_002 = ;
  input_003 = ;
  A = 0;
  X = 0;
  Z = 0;
  W = 0;
  V = 0;
  S = 0;

  // Variable initializations:
  hA = 0.8106;
  KaA = 4.5842;
  kaa = 1.24007e-10;
  kax = 5;
  kzA = 2216.72;
  ksA = 2e-05;
  k1 = 0.000417755;
  h1 = 2.84337;
  KM_A = 132251000;
  kxa = 2.37605e-10;
  kxw = 0.0002565;
  h3 = 254.727;
  kW = 0.00295691;
  koutZ = 0.0041765;
  koutX = 0.000861815;
  P_A = 30000000;
  kS = 0.0006;

  // Other declarations:
  var readout_001, readout_002, readout_003;
  const hA, KaA, kaa, kax, kzA, ksA, k1, h1, KM_A, kxa, kxw, h3, kW,
    koutZ;
  const koutX, P_A, kS;
end
```

Listing 19: Archived Antimony for F2a: Panel, gpt-6-astra, biocurator off; task `56550507_0`; selected version 27

```
// Created by libAntimony v3.1.3
model *egf_cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, EGFRoff in cell, pAkt in cell,
    Kinase in cell;
  species pKinase in cell, pS6 in cell, pEGFRint, KinasePrimed, Feedback;
  species KinaseSecond, KinaseThird;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := pEGFR + pEGFRint;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  kinase_deactivation: pKinase => Kinase; k_Kdeph*pKinase;
  receptor_downreg: pEGFR => pEGFRint; k_internal*pEGFR^n_int;
  internal_dephos: pEGFRint => EGFRoff; k_intdeph*pEGFRint;
  kinase_activation: Kinase => KinasePrimed; k_Kact*pAkt*Kinase;
  feedback_induction:  => Feedback; pAkt;
  reaction_1: EGFR -> pEGFR; k_bind*EGF*EGFR - k_unbind*pEGFR;
  s6_activation:  => pS6; pKinase;
  s6_dephos: pS6 => ; k_Sdeph*pS6;
  akt_activation:  => pAkt; k_Aact*pEGFR/(1 + Feedback);
  akt_dephos: pAkt => ; k_unbind*pAkt;
  kinase_maturation: KinasePrimed => KinaseSecond; k_Kdeph*KinasePrimed;
  kinase_completion: KinaseSecond => KinaseThird; k_Kdeph*KinaseSecond;
  kinase_final: KinaseThird => pKinase; k_Kdeph*KinaseThird;

  // Species initializations:
  EGFR = EGFR_init;
  pEGFR = 0;
  EGFRoff = 0;
  pAkt = 0;
  Kinase = Kinase_init;
  pKinase = 0;
  pS6 = 0;
  pEGFRint = ;
  KinasePrimed = ;
  Feedback = ;
  KinaseSecond = ;
  KinaseThird = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGFR_init = 1;
  Kinase_init = 1;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 3600;
  k_bind = 0.001;
  k_internal = 0.001;
  k_Aact = 0.1;
  k_Kact = 0.01;
  k_Kdeph = 0.001;
  k_Sdeph = 0.001;
  n_int = ;
  k_intdeph = ;
```

```
  k_unbind = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_init, Kinase_init, EGF_0, EGF_rate, EGF_end, k_bind,
    k_internal;
  const k_Aact, k_Kact, k_Kdeph, k_Sdeph, n_int, k_intdeph, k_unbind;
end
```

Listing 20: Archived Antimony for F2b: Panel, gpt-6-astra, biocurator off; task `56550507_1`; selected version 16

```
// Created by libAntimony v3.1.3
model *egfr_akt_s6()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, pEGFR_internal in cell,
    EGFR_internal in cell;
  species Akt in cell, pAkt in cell, Kinase in cell, pKinase in cell, S6
    in cell;
  species pS6 in cell, PI3K_drive, S6K_active, EGFR_L;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(pEGFR + pEGFR_internal);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  egfr_internalize: pEGFR => pEGFR_internal; k_internal*pEGFR;
  internal_dephos: pEGFR_internal => EGFR_internal; k_internal_dephos*
    pEGFR_internal;
  akt_activation: Akt => pAkt; k_akt_act*PI3K_drive*Akt/(1 + (pKinase/
    K_feedback_akt)^n_feedback_akt);
  akt_dephos: pAkt => Akt; k_akt_dephos*pAkt;
  kinase_activation: Kinase => pKinase; k_kinase_max*Kinase*pAkt/(
    K_akt_kinase + pAkt);
  kinase_dephos: pKinase => Kinase; k_kinase_dephos*pKinase;
  s6_activation: S6 => pS6; k_kinase_max*S6*S6K_active^n_s6;
  s6_dephos: pS6 => S6; k_akt_dephos*pS6;
  reaction_1: EGFR -> EGFR_L; k_bind*EGF*EGFR - k_ligand_off*EGFR_L;
  egfr_dephos: pEGFR => EGFR_internal; k_egfr_dephos*pEGFR;
  egfr_autophos: 2 EGFR_L => pEGFR; k_dimer*EGFR_L*EGFR_L;
  s6k_maturation:  => S6K_active; k_kinase_dephos*(pKinase - S6K_active);
  pi3k_recruitment:  => PI3K_drive; k_pi3k_relax*(pEGFR +
    endosomal_activity*pEGFR_internal - PI3K_drive);
  egfr_basal_internalize: EGFR => EGFR_internal; k_basal_internal*EGFR;

  // Species initializations:
  EGFR = EGFR_init;
  pEGFR = 0;
  pEGFR_internal = 0;
  EGFR_internal = 0;
  Akt = Akt_init;
  pAkt = 0;
  Kinase = Kinase_init;
  pKinase = 0;
  S6 = S6_init;
  pS6 = 0;
  PI3K_drive = ;
  S6K_active = ;
  EGFR_L = ;

  // Compartment initializations:
```

```
  cell = 1;

  // Variable initializations:
  EGFR_init = 0.5;
  Akt_init = 0.5;
  Kinase_init = 1;
  S6_init = 0.5;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 0;
  k_bind = 0.001;
  k_egfr_dephos = 0.001;
  k_internal = 0.001;
  k_internal_dephos = 0.001;
  k_akt_act = 0.1;
  k_akt_dephos = 0.01;
  k_kinase_dephos = 0.001;
  K_feedback_akt = ;
  n_feedback_akt = ;
  k_kinase_max = ;
  K_akt_kinase = ;
  n_s6 = ;
  k_ligand_off = ;
  k_dimer = ;
  k_pi3k_relax = ;
  endosomal_activity = ;
  k_basal_internal = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_init, Akt_init, Kinase_init, S6_init, EGF_0, EGF_rate;
  const EGF_end, k_bind, k_egfr_dephos, k_internal, k_internal_dephos,
    k_akt_act;
  const k_akt_dephos, k_kinase_dephos, K_feedback_akt, n_feedback_akt,
    k_kinase_max;
  const K_akt_kinase, n_s6, k_ligand_off, k_dimer, k_pi3k_relax,
    endosomal_activity;
  const k_basal_internal;
end
```

Listing 21: Archived Antimony for F2c: Panel, gpt-6-astra, biocurator off; task `56550507_2`; selected version 20

```
// Created by libAntimony v3.1.3
model *egf_cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, ipEGFR in cell, iEGFR in cell, Akt
    in cell;
  species pAkt in cell, MTOR in cell, aMTOR in cell, S6K in cell, aS6K in
    cell;
  species S6 in cell, pS6 in cell, F, bEGFR;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(pEGFR + ipEGFR);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  akt_activation: Akt -> pAkt; (k_Kon*pEGFR/(K_A + pEGFR))*Akt/(1 + F);
  akt_dephos: pAkt -> Akt; k_Aoff*pAkt;
  mtor_activation: MTOR -> aMTOR; k_Mon*pAkt*MTOR;
  mtor_deactivation: aMTOR -> MTOR; k_Koff*aMTOR;
```

```
  s6k_activation: S6K -> aS6K; k_Kon*aMTOR*S6K;
  s6k_deactivation: aS6K -> S6K; k_Koff*aS6K;
  s6_activation: S6 -> pS6; aS6K^n_S*S6;
  s6_dephos: pS6 -> S6; k_Soff*pS6;
  reaction_1: EGFR -> bEGFR; k_bind*EGF*EGFR;
  receptor_dimerize: 2 bEGFR -> pEGFR; k_dim*bEGFR*bEGFR;
  receptor_recycle: iEGFR -> 2 EGFR; k_rec*iEGFR;
  receptor_internalize: pEGFR -> iEGFR; k_int*pEGFR;
  feedback_generation:  -> F; q_F*pAkt/(Akt + pAkt);
  receptor_protected_internalize: pEGFR -> ipEGFR; k_protect*pEGFR;
  internal_dephos: ipEGFR -> iEGFR; k_ioff*ipEGFR;
  receptor_degradation: iEGFR -> ; k_Rdeg*iEGFR;

  // Species initializations:
  EGFR = EGFR_init;
  pEGFR = 0;
  ipEGFR = 0;
  iEGFR = 0;
  Akt = Akt_init;
  pAkt = 0;
  MTOR = MTOR_init;
  aMTOR = 0;
  S6K = S6K_init;
  aS6K = 0;
  S6 = S6_init;
  pS6 = 0;
  F = ;
  bEGFR = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGFR_init = 2;
  Akt_init = 2;
  MTOR_init = 1;
  S6K_init = 1;
  S6_init = 2;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 3600;
  k_bind = 0.001;
  k_int = 0.003;
  k_ioff = 0.001;
  k_rec = 0.0001;
  k_Aoff = 0.01;
  k_Mon = 0.001;
  k_Kon = 0.001;
  k_Koff = 0.001;
  k_Soff = 0.001;
  K_A = ;
  n_S = ;
  k_dim = ;
  q_F = ;
  k_protect = ;
  k_Rdeg = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_init, Akt_init, MTOR_init, S6K_init, S6_init, EGF_0,
    EGF_rate;
  const EGF_end, k_bind, k_int, k_ioff, k_rec, k_Aoff, k_Mon, k_Kon,
    k_Koff;
  const k_Soff, K_A, n_S, k_dim, q_F, k_protect, k_Rdeg;
end
```

Listing 22: Archived Antimony for F2d: Panel, gpt-6-astra, biocurator off; task `56550507_3`; selected version 53

```
// Created by libAntimony v3.1.3
model *egfr_cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, Akt in cell, pAkt in cell, Kin in
    cell;
  species pKin in cell, S6 in cell, pS6 in cell, pKin2, pEGFR_i, EGF_EGFR
    ;
  species EGFR_des;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := pEGFR + pEGFR_i;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  receptor_downreg: pEGFR => ; k_int*pEGFR;
  akt_activation: Akt => pAkt; k_Apower*Akt*pEGFR/(1 + pKin/K_Afeedback);
  akt_dephos: pAkt => Akt; k_Aoff*pAkt;
  kinase_activation: Kin => pKin; k_Kmax*Kin*pAkt^n_K/(K_Afeedback^n_K +
    pAkt^n_K);
  kinase_deactivation: pKin => Kin; k_Koff*pKin;
  s6_activation: S6 => pS6; k_Scombined*S6*pKin2^n_K;
  s6_dephos: pS6 => S6; k_Soff*pS6;
  receptor_internalize: pEGFR => pEGFR_i; k_transfer*pEGFR^n_internal;
  internal_phospho_loss: pEGFR_i => ; k_recover*pEGFR_i;
  kinase2_activation:  => pKin2; pKin;
  kinase2_deactivation: pKin2 => ; k_Aoff*pKin2;
  reaction_1: EGFR => EGF_EGFR; k_bind*EGF*EGFR;
  receptor_feedback_desensitize: pEGFR => EGFR_des; (k_feedback*pKin^n_K
    /(K_Afeedback^n_K + pKin^n_K))*pEGFR;
  receptor_resensitize: EGFR_des => EGFR; k_recover*EGFR_des;
  receptor_activation: 2 EGF_EGFR => 2 pEGFR; k_dim*EGF_EGFR*EGF_EGFR;

  // Species initializations:
  EGFR = EGFR_init;
  pEGFR = 0;
  Akt = Akt_init;
  pAkt = 0;
  Kin = Kin_init;
  pKin = 0;
  S6 = S6_init;
  pS6 = 0;
  pKin2 = ;
  pEGFR_i = ;
  EGF_EGFR = ;
  EGFR_des = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGFR_init = 10000000;
  Akt_init = 0.1;
  Kin_init = 1;
  S6_init = 1;
  EGF_0 = 0;
  EGF_rate = 0;
```

```
  EGF_end = 0;
  k_bind = 0.001;
  k_int = 0.001;
  k_Aoff = 0.01;
  k_Koff = 0.001;
  k_Soff = 0.0001;
  k_Apower = ;
  K_Afeedback = ;
  k_Kmax = ;
  n_K = ;
  k_Scombined = ;
  k_transfer = ;
  n_internal = ;
  k_recover = ;
  k_feedback = ;
  k_dim = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_init, Akt_init, Kin_init, S6_init, EGF_0, EGF_rate,
    EGF_end;
  const k_bind, k_int, k_Aoff, k_Koff, k_Soff, k_Apower, K_Afeedback,
    k_Kmax;
  const n_K, k_Scombined, k_transfer, n_internal, k_recover, k_feedback,
    k_dim;
end
```

Listing 23: Archived Antimony for F2e: Panel, gpt-6-astra, biocurator off; task `56550507_4`; selected version 28

```
// Created by libAntimony v3.1.3
model *egf_cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, ipEGFR in cell, iEGFR in cell, Akt
     in cell;
  species pAkt in cell, Kinase in cell, pKinase in cell, S6 in cell, pS6
    in cell;
  species BoundEGFR, K1, K2, primedAkt;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(pEGFR + ipEGFR);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  akt_dephos: pAkt => Akt; d_A*pAkt;
  kinase_deactivation: pKinase => Kinase; d_A*pKinase;
  s6_activation: S6 => pS6; d_S*S6*pKinase^n_S/(K_S^n_S + pKinase^n_S);
  s6_dephos: pS6 => S6; d_S*pS6;
  reaction_1: EGFR -> BoundEGFR; k_bind*EGF*EGFR - k_unbind*BoundEGFR;
  receptor_dimerization: 2 BoundEGFR => pEGFR; k_dim*BoundEGFR*BoundEGFR;
  receptor_dephos: pEGFR => 2 BoundEGFR; k_Roff*pEGFR;
  kinase_activation: Kinase => K1; k_K*pAkt*Kinase;
  kinase_priming: K1 => K2; k_tr*K1;
  kinase_maturation: K2 => pKinase; k_tr*K2;
  singly_liganded_dimerization: BoundEGFR + EGFR => pEGFR; k_single*
    BoundEGFR*EGFR;
  receptor_recycling: iEGFR => 2 EGFR; k_recycle*iEGFR;
  receptor_internalization: pEGFR => ipEGFR; k_int*pEGFR;
  internal_dephos: ipEGFR => iEGFR; k_int*ipEGFR;
  receptor_direct_inactivation: pEGFR => iEGFR; k_direct*pEGFR;
```

```
  akt_activation: Akt => primedAkt; k_A*2*(pEGFR + eff_internal*ipEGFR)*
    Akt/(1 + K2/K_feedback);
  akt_maturation: primedAkt => pAkt; k_tr*primedAkt;
  primed_akt_dephos: primedAkt => Akt; d_A*primedAkt;

  // Species initializations:
  EGFR = EGFR_init;
  pEGFR = 0;
  ipEGFR = 0;
  iEGFR = 0;
  Akt = Akt_init;
  pAkt = 0;
  Kinase = Kinase_init;
  pKinase = 0;
  S6 = S6_init;
  pS6 = 0;
  BoundEGFR = ;
  K1 = ;
  K2 = ;
  primedAkt = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGFR_init = 2;
  Akt_init = 2;
  Kinase_init = 1;
  S6_init = 2;
  EGF_0 = 0;
  EGF_rate = 0;
  EGF_end = 3600;
  k_bind = 0.001;
  k_Roff = 0.001;
  k_int = 0.003;
  k_A = 0.01;
  d_A = 0.005;
  k_K = 0.001;
  d_S = 0.0003;
  n_S = ;
  K_S = ;
  k_unbind = ;
  k_dim = ;
  k_tr = ;
  k_single = ;
  k_recycle = ;
  k_direct = ;
  eff_internal = ;
  K_feedback = ;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGFR_init, Akt_init, Kinase_init, S6_init, EGF_0, EGF_rate;
  const EGF_end, k_bind, k_Roff, k_int, k_A, d_A, k_K, d_S, n_S, K_S,
    k_unbind;
  const k_dim, k_tr, k_single, k_recycle, k_direct, eff_internal,
    K_feedback;
end
```

Listing 24: Archived Antimony for F3a: Null arm, gpt-6-astra; task `56550470_0`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, boundEGFR in cell, pEGFR in cell, internalEGFR in
     cell;
  species pEGFRslow in cell, PI3K in cell, pPI3K in cell, Akt in cell,
    pAkt in cell;
  species Feedback in cell, mTOR in cell, pmTOR in cell, pKinase in cell;
  species S6 in cell, S6mono in cell, pS6 in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(pEGFR + pEGFRslow);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  reaction_1: EGFR -> boundEGFR; kon*EGF*EGFR;
  unbinding: boundEGFR -> EGFR; koff*boundEGFR;
  dimerize: 2 boundEGFR -> pEGFR; kdim*boundEGFR^2/10000000;
  internalize: pEGFR -> 2 internalEGFR; kint*pEGFR;
  stabilize: pEGFR -> pEGFRslow; kstable*pEGFR;
  slow_internalize: pEGFRslow -> 2 internalEGFR; kslow*pEGFRslow;
  activate_PI3K: PI3K -> pPI3K; ka*(2*(pEGFR + effslow*pEGFRslow)
    /10000000)*PI3K/(Kr + 2*(pEGFR + effslow*pEGFRslow)/10000000);
  deactivate_PI3K: pPI3K -> PI3K; kda*pPI3K;
  activate_Akt: Akt -> pAkt; ka*pPI3K*Akt;
  deactivate_Akt: pAkt -> Akt; kda*pAkt*(1 + Feedback);
  feedback_on:  -> Feedback; kfeedback*(pAkt/0.02);
  activate_mTOR: mTOR -> pmTOR; km*(pAkt/0.02)*mTOR;
  deactivate_mTOR: pmTOR -> mTOR; kdm*pmTOR;
  activate_Kinase:  -> pKinase; kdm*pmTOR;
  deactivate_Kinase: pKinase -> ; kdm*pKinase;
  activate_S6: S6 -> S6mono; ks*pKinase*S6;
  activate_S6_2: S6mono -> pS6; ks*pKinase*S6mono;
  deactivate_S6_1: S6mono -> S6; kds*S6mono/(Ksp + (S6mono + pS6)/1e-5);
  deactivate_S6: pS6 -> S6mono; kds*pS6/(Ksp + (S6mono + pS6)/1e-5);

  // Species initializations:
  EGFR = 10000000;
  boundEGFR = 0;
  pEGFR = 0;
  internalEGFR = 0;
  pEGFRslow = 0;
  PI3K = 1;
  pPI3K = 0;
  Akt = 0.02;
  pAkt = 0;
  Feedback = 0;
  mTOR = 1;
  pmTOR = 0;
  pKinase = 0;
  S6 = 1e-05;
  S6mono = 0;
  pS6 = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
```

```
  EGF_0 = 1;
  EGF_rate = 0;
  EGF_end = 3600;
  kon = 0.0010484828204757;
  koff = 0.0123994193637012;
  kdim = 1.05913723277252;
  kint = 0.183124205101635;
  kstable = 0.0004975209557139;
  kslow = 0.0003778017931776;
  ka = 0.0174302442337268;
  effslow = 0.305923625980726;
  Kr = 0.0065459749819684;
  kda = 0.0068039395762827;
  kfeedback = 0.39879513342466;
  km = 0.0897789220214074;
  kdm = 0.0063972241686999;
  ks = 0.0047984301559575;
  kds = 0.0007997642430367;
  Ksp = 0.129972876543754;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGF_0, EGF_rate, EGF_end, kon, koff, kdim, kint, kstable,
    kslow;
  const ka, effslow, Kr, kda, kfeedback, km, kdm, ks, kds, Ksp;
end
```

Listing 25: Archived Antimony for F3b: Null arm, gpt-6-astra; task `56550470_1`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *cascade()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, Bound in cell, pEGFR in cell, pEGFRi in cell,
    EGFRi in cell;
  species Traff in cell, pEGFRslow in cell, D3 in cell, D4 in cell, D5 in
     cell;
  species D6 in cell, D7 in cell, Akt in cell, pAkt in cell, F in cell,
    D1 in cell;
  species D2 in cell, S6 in cell, pS6 in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(pEGFR + pEGFRi + pEGFRslow);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  delay_7_on:  -> D7; k_delay*D6;
  delay_7_off: D7 -> ; k_delay*D7;
  delay_6_on:  -> D6; k_delay*D5;
  delay_6_off: D6 -> ; k_delay*D6;
  delay_5_on:  -> D5; k_delay*D4;
  delay_5_off: D5 -> ; k_delay*D5;
  delay_4_on:  -> D4; k_delay*D3;
  delay_4_off: D4 -> ; k_delay*D4;
  reaction_1: EGFR -> Bound; k_bind*EGF*EGFR;
  endosomal_sorting: pEGFRi -> pEGFRslow; k_mphos*pEGFRi;
  slow_endosomal_dephosphorylation: pEGFRslow -> EGFRi; k_mdephos*
    pEGFRslow;
  binding_reverse: Bound -> EGFR; k_off*Bound;
  receptor_phosphorylation: 2 Bound -> pEGFR; k_phos*Bound^2/10000000;
  internalization: pEGFR -> pEGFRi; k_int*Traff*pEGFR;
```

```
trafficking_on:  -> Traff; k_T*(pEGFR/10000000);
trafficking_off: Traff -> ; k_T*Traff;
endosomal_dephosphorylation: pEGFRi -> EGFRi; k_I_dephos*pEGFRi;
akt_activation: Akt -> pAkt; (k_A*(pEGFR/10000000)/(K_A + pEGFR
  /10000000))*Akt/(1 + F);
akt_dephosphorylation: pAkt -> Akt; k_Ad*pAkt;
feedback_on:  -> F; q_F*pAkt/0.03;
delay_1_on:  -> D1; k_delay*pAkt/0.03;
delay_1_off: D1 -> ; k_delay*D1;
delay_2_on:  -> D2; k_delay*D1;
delay_2_off: D2 -> ; k_delay*D2;
delay_3_on:  -> D3; k_delay*D2;
delay_3_off: D3 -> ; k_delay*D3;
s6_phosphorylation: S6 -> pS6; (k_fastS*pAkt/0.03 + k_S*D7^2/(K_D^2 +
  D7^2))*S6;
s6_dephosphorylation: pS6 -> S6; k_Sd*pS6;

// Species initializations:
EGFR = 10000000;
Bound = 0;
pEGFR = 0;
pEGFRi = 0;
EGFRi = 0;
Traff = 0;
pEGFRslow = 0;
D3 = 0;
D4 = 0;
D5 = 0;
D6 = 0;
D7 = 0;
Akt = 0.03;
pAkt = 0;
F = 0;
D1 = 0;
D2 = 0;
S6 = 1e-05;
pS6 = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
k_delay = 0.0189387834686229;
EGF_0 = 1;
EGF_rate = 0;
EGF_end = 3600;
k_bind = 0.0023628830896278;
k_mphos = 0.0022893038566943;
k_mdephos = 0.0014060558312014;
k_off = 0.0123161395951657;
k_phos = 0.0184911826473626;
k_int = 129.030019436401;
k_T = 0.0037418669592524;
k_I_dephos = 0.153134055708642;
k_A = 0.0002837604761323;
K_A = 0.0019936159135921;
k_Ad = 0.0179181697583067;
q_F = 0.944830525327276;
k_fastS = 0.0002684580119242;
k_S = 1.98298604237082e-05;
K_D = 0.0015850256266394;
k_Sd = 0.0005227435250706;

// Other declarations:
var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
```

```
  const cell, k_delay, EGF_0, EGF_rate, EGF_end, k_bind, k_mphos,
    k_mdephos;
  const k_off, k_phos, k_int, k_T, k_I_dephos, k_A, K_A, k_Ad, q_F,
    k_fastS;
  const k_S, K_D, k_Sd;
end
```

Listing 26: Archived Antimony for F3c: Null arm, gpt-6-astra; task `56550470_2`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *signaling()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, Bound in cell, Dimer in cell, Internal in cell,
    Dephos in cell;
  species preAkt in cell, SlowP in cell, T4 in cell, T5 in cell, Akt in
    cell;
  species pAkt in cell, T1 in cell, T2 in cell, T3 in cell, S6 in cell,
    pS6 in cell;
  species Feedback in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := 2*(Dimer + SlowP);
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  reaction_1: EGFR -> Bound; kon*EGF*EGFR;
  unbind: Bound -> EGFR; koff*Bound;
  dimerize: 2 Bound -> Dimer; kdim*Bound*Bound/10000000;
  slow_internalize: Dimer -> SlowP; kslow*Dimer;
  destroySlow: SlowP -> ; kdeg*SlowP;
  recycleSlow: SlowP -> 2 EGFR; krec*SlowP;
  internalize: Dimer -> Dephos; kint*Dimer;
  destroy: Dephos -> ; kdeg*Dephos;
  recycle2: Dephos -> 2 EGFR; krec*Dephos;
  activate_Akt: Akt -> preAkt; (kda*(2*Dimer/10000000)/(Kra + 2*Dimer
    /10000000))*0.03;
  prime: preAkt -> pAkt; kda*preAkt;
  deactivate_Akt: pAkt -> Akt; kda*(1 + Feedback/Kf)*pAkt;
  feedback:  -> Feedback; kf*(pAkt/0.03 - Feedback);
  relay1:  -> T1; kt*(pAkt/0.03 - T1);
  relay2:  -> T2; kt*(T1 - T2);
  relay3:  -> T3; kt*(T2 - T3);
  relay4:  -> T4; kt*(T3 - T4);
  relay5:  -> T5; kt*(T4 - T5);
  activate_S6: S6 -> pS6; (kds*T5*T5/(Ks*Ks + T5*T5))*1e-5;
  deactivate_S6: pS6 -> S6; kds*pS6;

  // Species initializations:
  EGFR = 10000000;
  Bound = 0;
  Dimer = 0;
  Internal = 0;
  Dephos = 0;
  preAkt = 0;
  SlowP = 0;
  T4 = 0;
  T5 = 0;
  Akt = 0.03;
  pAkt = 0;
  T1 = 0;
```

```
  T2 = 0;
  T3 = 0;
  S6 = 1e-05;
  pS6 = 0;
  Feedback = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGF_0 = 1;
  EGF_rate = 0;
  EGF_end = 3600;
  kon = 0.001169173184183;
  koff = 0.0052551870346624;
  kdim = 0.33662158403447;
  kslow = 0.000706851378637;
  kdeg = 0.0006666675563271;
  krec = 0.0002159547067057;
  kint = 0.220260215419609;
  kda = 0.0123620679512752;
  Kra = 0.0024610391331009;
  Kf = 0.0006278737986614;
  kf = 0.0003087620795029;
  kt = 0.0146449585413041;
  kds = 0.0004576464690307;
  Ks = 0.0268249287314807;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGF_0, EGF_rate, EGF_end, kon, koff, kdim, kslow, kdeg,
    krec;
  const kint, kda, Kra, Kf, kf, kt, kds, Ks;
end
```

Listing 27: Archived Antimony for F3d: Null arm, gpt-6-astra; task `56550470_3`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *pathway()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, pEGFRi in cell, Aktprimed in cell;
  species Akt in cell, pAkt in cell, Feedback in cell, PI3Ka in cell,
    mTORa in cell;
  species S6Ka in cell, S6primed in cell, S6 in cell, pS6 in cell;

  // Assignment Rules:
  EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
  readout_pEGFR_tot := pEGFR + pEGFRi;
  readout_pAkt_tot := pAkt;
  readout_pS6_tot := pS6;

  // Reactions:
  reaction_1: EGFR -> pEGFR; kon*EGF*EGFR;
  reaction_2: pEGFR -> EGFR; koff*pEGFR;
  reaction_3: pEGFR -> pEGFRi; kint*pEGFR*(pEGFR/20000000)^2;
  reaction_4: pEGFRi -> ; kdeg*pEGFRi;
  reaction_5: Akt -> Aktprimed; ka*PI3Ka*Akt/(1 + Feedback);
  akt_second: Aktprimed -> pAkt; ka*PI3Ka*Aktprimed/(1 + Feedback);
  reaction_6: pAkt -> Akt; kd*pAkt;
  reaction_7:  -> Feedback; kf*pAkt/0.025;
  pi_on:  -> PI3Ka; kp*pEGFR/20000000;
  pi_off: PI3Ka -> ; kp*PI3Ka;
```

```
  reaction_9:  -> mTORa; km*(pAkt/0.025)/(KmA + pAkt/0.025);
  reaction_10: mTORa -> ; km*mTORa;
  reaction_11:  -> S6Ka; km*mTORa;
  reaction_12: S6Ka -> ; km*S6Ka;
  rsk_branch: S6 -> pS6; krsk*(pAkt/0.025)*S6;
  reaction_13: S6 -> S6primed; ks*S6Ka*S6;
  s6_second: S6primed -> pS6; ks*S6Ka*S6primed;
  s6_reset: S6primed -> S6; ksd*S6primed;
  reaction_14: pS6 -> S6; ksd*pS6;

  // Species initializations:
  EGFR = 20000000;
  pEGFR = 0;
  pEGFRi = 0;
  Aktprimed = 0;
  Akt = 0.025;
  pAkt = 0;
  Feedback = 0;
  PI3Ka = 0;
  mTORa = 0;
  S6Ka = 0;
  S6primed = 0;
  S6 = 1.5e-05;
  pS6 = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  EGF_0 = 1;
  EGF_rate = 0;
  EGF_end = 3600;
  kon = 0.003646511794735;
  koff = 0.0447459634663253;
  kint = 4.17317471993005;
  kdeg = 0.0031815010428733;
  ka = 1.2410800833201;
  kd = 0.0141080295962312;
  kf = 0.107180449143803;
  kp = 0.0129732242439098;
  km = 0.0014013764156658;
  KmA = 0.480867524051315;
  krsk = 0.0002566593071405;
  ks = 0.242447981241375;
  ksd = 0.0244197104162129;

  // Other declarations:
  var EGF, readout_pEGFR_tot, readout_pAkt_tot, readout_pS6_tot;
  const cell, EGF_0, EGF_rate, EGF_end, kon, koff, kint, kdeg, ka, kd, kf
    ;
  const kp, km, KmA, krsk, ks, ksd;
end
```

Listing 28: Archived Antimony for F3e: Null arm, gpt-6-astra; task `56550470_4`; delivered problem, no version store; Antimony converted from the archived SBML

```
// Created by libAntimony v3.1.3
model *signaling()

  // Compartments and Species:
  compartment cell;
  species Inactive in cell, SlowEndosome in cell, EGFR in cell, Bound in
    cell;
  species pEGFR in cell, Internal in cell, Degraded in cell, Relay in
    cell;
```

```
species AktIntermediate in cell, Akt in cell, pAkt in cell, Feedback in cell;
species TOR in cell, S6K in cell, S6 in cell, pS6 in cell;

// Assignment Rules:
EGF := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0);
receptor_signal := 2*pEGFR/10000000;
readout_pEGFR_tot := 2*pEGFR + 2*SlowEndosome;
readout_pAkt_tot := pAkt;
readout_pS6_tot := pS6;

// Reactions:
reaction_1: EGFR -> Bound; kon*EGF*EGFR;
unbind: Bound -> EGFR; koff*Bound;
dimer_phosphorylation: 2 Bound -> pEGFR; kph*Bound*Bound/10000000;
slow_internalization: pEGFR -> SlowEndosome; kslow*pEGFR;
slow_dephosphorylation: SlowEndosome -> Inactive; kslow*SlowEndosome;
internalization: pEGFR -> Inactive; kint*pEGFR;
recycling: Inactive -> 2 EGFR; krec*Inactive;
degradation: Inactive -> Degraded; kds*Inactive;
akt_phosphorylation: Akt -> AktIntermediate; (ka*receptor_signal/(Ka + receptor_signal))*Akt;
akt_second: AktIntermediate -> pAkt; kda*AktIntermediate;
akt_dephosphorylation: pAkt -> Akt; kda*pAkt*(1 + Feedback/Kf);
feedback_build:  -> Feedback; kf*pAkt/0.03;
feedback_loss: Feedback -> ; kf*Feedback;
tor_activation:  -> TOR; klag*(pAkt/0.03)/(1 + pAkt/Ktor);
tor_deactivation: TOR -> ; klag*TOR;
relay_activation:  -> Relay; klag*TOR;
relay_loss: Relay -> ; klag*Relay;
s6k_activation:  -> S6K; klag*Relay;
s6k_deactivation: S6K -> ; klag*S6K;
direct_s6: S6 -> pS6; (kdirect*pAkt/0.03)*S6;
s6_phosphorylation: S6 -> pS6; (ks*S6K^4/(Ks^4 + S6K^4))*S6;
s6_dephosphorylation: pS6 -> S6; kds*pS6;

// Species initializations:
Inactive = 0;
SlowEndosome = 0;
EGFR = 10000000;
Bound = 0;
pEGFR = 0;
Internal = 0;
Degraded = 0;
Relay = 0;
AktIntermediate = 0;
Akt = 0.03;
pAkt = 0;
Feedback = 0;
TOR = 0;
S6K = 0;
S6 = 2e-05;
pS6 = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
EGF_0 = 1;
EGF_rate = 0;
EGF_end = 3600;
kon = 0.002;
koff = 0.02;
kph = 0.1;
kslow = 0.004;
```

```
  kint = 0.006;
  krec = 0.0001;
  kds = 0.001;
  ka = 0.04;
  Ka = 0.015;
  kda = 0.006;
  Kf = 0.1;
  kf = 0.002;
  klag = 0.003;
  Ktor = 0.01;
  kdirect = 0.0001;
  ks = 0.008;
  Ks = 0.3;

  // Other declarations:
  var EGF, receptor_signal, readout_pEGFR_tot, readout_pAkt_tot,
    readout_pS6_tot;
  const cell, EGF_0, EGF_rate, EGF_end, kon, koff, kph, kslow, kint, krec
    ;
  const kds, ka, Ka, kda, Kf, kf, klag, Ktor, kdirect, ks, Ks;
end
```

H AGENT-GENERATED BENCHMARK MODEL SOURCES

These listings provide the 85 additional Antimony models from the benchmark collection of Section 2.2. The five Fujita models shared with that collection already appear in Appendix G and are cross-referenced below, completing the 90-model cohort. Examples follow the benchmark order; model numbers within each example follow run start time, rather than fit quality. Captions identify the model, array task, and selected version; archive identifiers and source hashes are retained in the accompanying `paper/model_sources/benchmark/models.csv` manifest.

Sources are preserved byte for byte, including comments and unset initial values. Their numerical assignments need not equal the fitted values: the saved parameter tables are supplied separately alongside each source. These files are not complete PEtab problems; simulation also requires the original conditions, observables, and initialization semantics. No models were reconstructed and no parameters were refitted for this annex.

Shared Fujita sources.

Benchmark model	Array task	Selected version	Existing listing
`fujita_1`	`56476116_1`	8	O1b (Listing 2)
`fujita_2`	`56476116_0`	22	O1a (Listing 1)
`fujita_3`	`56476116_4`	20	O1e (Listing 5)
`fujita_4`	`56476116_3`	29	O1d (Listing 4)
`fujita_5`	`56476116_2`	28	O1c (Listing 3)

H.1 PERELSON

Listing 29: Archived Antimony for `perelson_1`; task `57485624_3`; selected version 1

```
// Created by libAntimony v3.1.3
model *hiv_ritonavir_pi()

  // Compartments and Species:
  compartment plasma;
  species I in plasma, Vi in plasma, Vn in plasma, rit in plasma;

  // Assignment Rules:
  readout_viral_load := Vi + Vn;

  // Reactions:
  i_repl:  -> I; delta_i*(1 - rit);
  i_death: I -> ; delta_i*I;
  vi_prod: I -> I + Vi; P_tot*I*(1 - rit);
  vn_prod: I -> I + Vn; P_tot*I*rit;
  vi_clear: Vi -> ; c_vi*Vi;
  vn_clear: Vn -> ; c_vn*Vn;

  // Species initializations:
  I = I_init;
  Vi = P_tot/c_vi;
  Vn = 0;
  rit = 0;

  // Compartment initializations:
  plasma = 1;

  // Variable initializations:
  I_init = 1;
  P_tot = 3000000;
  c_vi = 3;
  delta_i = 0.5;
  c_vn = 1;

  // Other declarations:
```

```
  var readout_viral_load;
  const plasma, I_init, P_tot, c_vi, delta_i, c_vn;
end
```

Listing 30: Archived Antimony for `perelson_2`; task `57581393_0`; selected version 4

```
// Created by libAntimony v3.1.3
model *hiv_pi()

  // Compartments and Species:
  compartment plasma;
  species I in plasma, Vi in plasma, Vn in plasma, E in plasma;

  // Assignment Rules:
  readout_viral_load := A_load*(Vi + Vn);

  // Reactions:
  infection:  -> E; delta*Vi;
  eclipse_out: E -> I; delta*E;
  ideath: I -> ; delta*I;
  prod_vi:  -> Vi; (1 - rit)*delta*I;
  prod_vn:  -> Vn; rit*delta*I;
  clear_vi: Vi -> ; delta*Vi;
  clear_vn: Vn -> ; c_vn*Vn;

  // Species initializations:
  I = I_init;
  Vi = Vi_init;
  Vn = 0;
  E = E_init;

  // Compartment initializations:
  plasma = 1;

  // Variable initializations:
  I_init = 1;
  Vi_init = 1;
  E_init = 1;
  delta = 4.17;
  A_load = 1300000;
  rit = 1;
  c_vn = ;

  // Other declarations:
  var readout_viral_load;
  const plasma, I_init, Vi_init, E_init, delta, A_load, rit, c_vn;
end
```

Listing 31: Archived Antimony for `perelson_3`; task `57581393_4`; selected version 1

```
// Created by libAntimony v3.1.3
model *hiv_pi_v1()

  // Compartments and Species:
  compartment plasma;
  species rit in plasma, X in plasma, V in plasma;

  // Assignment Rules:
  readout_viral_load := V;

  // Reactions:
  new_infected:  -> X; inf_rate*(1 - rit);
  ic_death: X -> ; c_vir*X;
  v_prod:  -> V; X;
```

```
  v_clear: V -> ; c_vir*V;

  // Species initializations:
  rit = 0;
  X = X_init;
  V = 0;

  // Compartment initializations:
  plasma = 1;

  // Variable initializations:
  X_init = 1;
  inf_rate = 500000;
  c_vir = 2;

  // Other declarations:
  var readout_viral_load;
  const plasma, X_init, inf_rate, c_vir;
end
```

Listing 32: Archived Antimony for `perelson_4`; task `57581393_2`; selected version 2

```
// Created by libAntimony v3.1.3
model *hiv_pi()

  // Compartments and Species:
  compartment plasma;
  species Vi in plasma, Vni in plasma, I in plasma;

  // Assignment Rules:
  readout_viral_load := Vi + Vni;

  // Reactions:
  i_repl:  -> I; (1 - rit)*delta*Vi/Vi_init;
  i_death: I -> ; delta*I;
  vi_prod:  -> Vi; (1 - rit)*c*Vi_init*I;
  vi_clear: Vi -> ; c*Vi;
  vni_prod:  -> Vni; rit*amp*c*Vi_init*I;
  vni_clear: Vni -> ; c*Vni;

  // Species initializations:
  Vi = Vi_init;
  Vni = 0;
  I = I_init;

  // Compartment initializations:
  plasma = 1;

  // Variable initializations:
  Vi_init = 1200000;
  I_init = 1;
  c = 3.4;
  delta = 0.5;
  amp = 2;
  rit = 0;

  // Other declarations:
  var readout_viral_load;
  const plasma, Vi_init, I_init, c, delta, amp, rit;
end
```

Listing 33: Archived Antimony for perelson_5; task 57581393_1; selected version 7

```
// Created by libAntimony v3.1.3
model *hiv_maturation_v7()

  // Compartments and Species:
  compartment plasma;
  species S in plasma, I in plasma, V_U in plasma, V_M in plasma;

  // Assignment Rules:
  Lv := c_U*VU0 + cascade*VM0;
  VU0 := f_U*V_tot0;
  VM0 := (1 - f_U)*V_tot0;
  m_mat := cascade*(1 - f_U)/f_U;
  lam := cascade*Lv/VM0;
  readout_viral_load := V_U + V_M;

  // Reactions:
  new_infection:  -> S; lam*V_M;
  eclipse_exit: S -> I; cascade*S;
  infected_death: I -> ; cascade*I;
  budding:  -> V_U; I;
  maturation: V_U -> V_M; m_mat*(1 - rit)*V_U;
  immature_clearance: V_U -> ; c_U*V_U;
  mature_clearance: V_M -> ; cascade*V_M;

  // Species initializations:
  S = Lv;
  I = Lv;
  V_U = VU0;
  V_M = VM0;

  // Compartment initializations:
  plasma = 1;

  // Variable initializations:
  V_tot0 = 1090207;
  f_U = 0.25734;
  cascade = 5.95;
  c_U = 0.5083;
  rit = 0.1;

  // Other declarations:
  var Lv, VU0, VM0, m_mat, lam, readout_viral_load;
  const plasma, V_tot0, f_U, cascade, c_U, rit;
end
```

H.2 CRAUSTE

Listing 34: Archived Antimony for crauste_1; task 56836848_25; selected version 11

```
// Created by libAntimony v3.1.3
model *CD8_response()

  // Compartments and Species:
  compartment blood;
  species Naive in blood, EarlyEffector in blood, LateEffector in blood
    , Memory in blood;
  species Pathogen in blood, EffectorTail;

  // Assignment Rules:
  readout_observable_Naive := Naive;
  readout_observable_EarlyEffector := EarlyEffector + EffectorTail;
  readout_observable_LateEffector := LateEffector;
```

```
  readout_observable_Memory := Memory;

  // Reactions:
  P_growth:  -> Pathogen; rPnet*Pathogen;
  P_killing: Pathogen -> ; kKill*Pathogen*(EarlyEffector + LateEffector);
  Naive_activation: Naive -> EarlyEffector; kAct*Naive*Pathogen;
  E_proliferation: EarlyEffector -> 2 EarlyEffector; kDiv*EarlyEffector*Pathogen^nP/(KP1^nP + Pathogen^nP);
  E_death: EarlyEffector -> ; dE*EarlyEffector;
  EL_differentiation: EarlyEffector -> LateEffector; kAct*EarlyEffector;
  L_death: LateEffector -> ; dL*LateEffector;
  LM_differentiation: LateEffector -> Memory; kLM*LateEffector;
  E_program: EarlyEffector -> EffectorTail; k12*EarlyEffector;
  Tail_death: EffectorTail -> ; dE2*EffectorTail;
  M_renewal:  -> Memory; rMnet*Memory;

  // Species initializations:
  Naive = Naive_init;
  EarlyEffector = 0;
  LateEffector = 0;
  Memory = 0;
  Pathogen = Pathogen_init;
  EffectorTail = ;

  // Compartment initializations:
  blood = 1;

  // Variable initializations:
  Naive_init = ;
  Pathogen_init = 1;
  kKill = 1e-05;
  kAct = 0.3;
  kDiv = 5;
  nP = 1;
  KP1 = 0.5;
  dE = 0.5;
  dL = 0.1;
  kLM = 0.1;
  rPnet = ;
  k12 = ;
  dE2 = ;
  rMnet = ;

  // Other declarations:
  var readout_observable_Naive, readout_observable_EarlyEffector,
    readout_observable_LateEffector;
  var readout_observable_Memory;
  const blood, Naive_init, Pathogen_init, kKill, kAct, kDiv, nP, KP1,
    dE;
  const dL, kLM, rPnet, k12, dE2, rMnet;
end
```

Listing 35: Archived Antimony for `crauste_2`; task `56836848_26`; selected version 11

```
// Created by libAntimony v3.1.3
model *cd8_vacv_primary()

  // Compartments and Species:
  compartment blood;
  species Pathogen in blood, Naive in blood, EarlyEffector in blood,
    EarlyPrecursor in blood;
  species LateEffector in blood, Memory in blood;

  // Assignment Rules:
  a_P := Pathogen/(alpha_act + Pathogen);
  readout_observable_Naive := Naive;
  readout_observable_EarlyEffector := EarlyEffector + EarlyPrecursor;
  readout_observable_LateEffector := LateEffector;
  readout_observable_Memory := Memory;

  // Reactions:
  p_growth:  -> Pathogen; r_P*Pathogen;
  p_clear: Pathogen -> ; k_clear*Pathogen*(EarlyEffector + LateEffector
    );
  n_act: Naive -> EarlyEffector; alpha_act*a_P*Naive;
  e1_prop: EarlyEffector -> 2 EarlyEffector; rho1*a_P*EarlyEffector;
  e1_die: EarlyEffector -> ; delta1*EarlyEffector;
  e1_diff: EarlyEffector -> LateEffector; gamma_d*EarlyEffector;
  e1_transit: EarlyEffector -> EarlyPrecursor; k_tp*EarlyEffector;
  e2_die: LateEffector -> ; c_fate*LateEffector;
  e2_mem: LateEffector -> Memory; c_fate*(1 - a_P)*LateEffector;

  // Species initializations:
  Pathogen = Pathogen_init;
  Naive = Naive_init;
  EarlyEffector = 0;
  EarlyPrecursor = 0;
  LateEffector = 0;
  Memory = 0;

  // Compartment initializations:
  blood = 1;

  // Variable initializations:
  Pathogen_init = 1;
  Naive_init = ;
  r_P = 0.284131787268448;
  k_clear = 7.64413561218487e-06;
  alpha_act = 1.2473207035373;
  rho1 = 29.8688815849035;
  delta1 = 21.9809724313313;
  gamma_d = 0.463051827525275;
  c_fate = 0.0969;
  k_tp = 0.0231819064996001;

  // Other declarations:
  var a_P, readout_observable_Naive, readout_observable_EarlyEffector,
    readout_observable_LateEffector;
  var readout_observable_Memory;
  const blood, Pathogen_init, Naive_init, r_P, k_clear, alpha_act, rho1
    , delta1;
  const gamma_d, c_fate, k_tp;
end
```

Listing 36: Archived Antimony for `crauste_3`; task `56836848_27`; selected version 16

```
// Created by libAntimony v3.1.3
model *cd8_vacv_response()

  // Compartments and Species:
  compartment cell;
  species Naive in cell, EarlyEffector in cell, LateEffector in cell,
    Memory in cell;
  species Pathogen in cell;

  // Assignment Rules:
  readout_observable_Naive := Naive;
  readout_observable_EarlyEffector := EarlyEffector;
  readout_observable_LateEffector := LateEffector;
  readout_observable_Memory := Memory;

  // Reactions:
  p_clear: Pathogen -> ; k_kill*(EarlyEffector + LateEffector)*Pathogen
    ;
  n_act: Naive -> EarlyEffector; (1 - f_L)*k_act*Naive*Pathogen;
  e_prolif: EarlyEffector -> 2 EarlyEffector; (p_E*Pathogen^8/(K_s^8 +
    Pathogen^8))*EarlyEffector;
  e_death: EarlyEffector -> ; d_E*EarlyEffector*Pathogen;
  e_to_l: EarlyEffector -> LateEffector; k_EL*EarlyEffector;
  l_to_m: LateEffector -> Memory; (K_s*0.001/(0.001 + Pathogen))*
    LateEffector;
  m_death: Memory -> ; k_EL*Memory;
  n_to_l: Naive -> LateEffector; f_L*k_act*Naive*Pathogen;

  // Species initializations:
  Naive = Naive_init;
  EarlyEffector = 0;
  LateEffector = 0;
  Memory = 0;
  Pathogen = Pathogen_init;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  Naive_init = ;
  Pathogen_init = 1;
  k_kill = 1.5e-05;
  k_act = 0.12;
  p_E = 1.75;
  d_E = 1.4;
  k_EL = 0.6;
  f_L = ;
  K_s = ;

  // Other declarations:
  var readout_observable_Naive, readout_observable_EarlyEffector,
    readout_observable_LateEffector;
  var readout_observable_Memory;
  const cell, Naive_init, Pathogen_init, k_kill, k_act, p_E, d_E, k_EL,
     f_L;
  const K_s;
end
```

Listing 37: Archived Antimony for `crauste_4`; task `56836848_28`; selected version 9

```
// Created by libAntimony v3.1.3
model *cd8_vaccinia_response()

  // Compartments and Species:
  compartment cell;
  species P in cell, N in cell, E in cell, L in cell, M in cell, Es;

  // Assignment Rules:
  readout_observable_Naive := N;
  readout_observable_EarlyEffector := E + Es;
  readout_observable_LateEffector := L;
  readout_observable_Memory := M;

  // Reactions:
  pathogen_growth: P -> 2 P; lam*P;
  naive_activation: N -> E; k_act*N*P;
  effector_proliferation: E -> 2 E; k_prol*E*P;
  effector_death: E -> ; d_E*E;
  effector_to_late: E -> L; k_EL*E;
  late_death: L -> ; d_L*L;
  late_to_memory: L -> M; k_LM*L;
  memory_selfrenewal: M -> 2 M; lam*M;
  late_proliferation: L -> 2 L; k_prol_L*L*P;
  effector_to_slow: E -> Es; lam*E;
  slow_effector_death: Es -> ; lam*Es;
  late_clearance: P -> ; k_clearL*L*P;

  // Species initializations:
  P = P_init;
  N = Naive_init;
  E = 0;
  L = 0;
  M = 0;
  Es = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  P_init = 1;
  Naive_init = ;
  k_act = 0.0001;
  k_prol = 0.001;
  d_E = 0.7;
  k_EL = 0.2;
  d_L = 0.1;
  k_LM = 0.02;
  lam = ;
  k_prol_L = ;
  k_clearL = ;

  // Other declarations:
  var readout_observable_Naive, readout_observable_EarlyEffector,
    readout_observable_LateEffector;
  var readout_observable_Memory;
  const cell, P_init, Naive_init, k_act, k_prol, d_E, k_EL, d_L, k_LM,
    lam;
  const k_prol_L, k_clearL;
end
```

Listing 38: Archived Antimony for `crauste_5`; task `56836848_29`; selected version 0

```
// Created by libAntimony v3.1.3
model *cd8_response()

  // Compartments and Species:
  compartment cell;
  species Naive in cell, EarlyEffector in cell, LateEffector in cell,
    Memory in cell;
  species Pathogen in cell;

  // Assignment Rules:
  readout_observable_Naive := Naive;
  readout_observable_EarlyEffector := EarlyEffector;
  readout_observable_LateEffector := LateEffector;
  readout_observable_Memory := Memory;

  // Reactions:
  naive_activation: Naive -> EarlyEffector; k_act*Pathogen*Naive;
  naive_loss: Naive -> ; k_dN*Naive;
  pathogen_replication:  -> Pathogen; r_v*Pathogen;
  pathogen_decay: Pathogen -> ; c_v*Pathogen;
  pathogen_clearance: Pathogen -> ; k_cl*Pathogen*EarlyEffector;
  early_proliferation: EarlyEffector -> 2 EarlyEffector; rho_1*Pathogen
    *EarlyEffector;
  early_death: EarlyEffector -> ; k_dE*EarlyEffector;
  early_to_late: EarlyEffector -> LateEffector; k_EL*EarlyEffector;
  late_proliferation: LateEffector -> 2 LateEffector; rho_2*Pathogen*
    LateEffector;
  late_death: LateEffector -> ; k_dL*LateEffector;
  late_to_memory: LateEffector -> Memory; k_LM*LateEffector;
  memory_loss: Memory -> ; k_dM*Memory;

  // Species initializations:
  Naive = Naive_init;
  EarlyEffector = 0;
  LateEffector = 0;
  Memory = 0;
  Pathogen = Pathogen_init;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  Naive_init = ;
  Pathogen_init = 1;
  k_act = 0.5;
  k_dN = 0.7;
  r_v = 2;
  c_v = 0.3;
  k_cl = 1e-05;
  rho_1 = 3;
  k_dE = 0.7;
  k_EL = 0.3;
  rho_2 = 1;
  k_dL = 0.15;
  k_LM = 0.15;
  k_dM = 0.02;

  // Other declarations:
  var readout_observable_Naive, readout_observable_EarlyEffector,
    readout_observable_LateEffector;
  var readout_observable_Memory;
  const cell, Naive_init, Pathogen_init, k_act, k_dN, r_v, c_v, k_cl,
    rho_1;
```

```
  const k_dE, k_EL, rho_2, k_dL, k_LM, k_dM;
end
```

H.3 RAHMAN

Listing 39: Archived Antimony for `rahman_1`; task `56836848_41`; selected version 12

```
// Created by libAntimony v3.1.3
model *hiv_prevalence()

  // Compartments and Species:
  compartment population;
  species S in population, I_n in population, I_m in population, I_w in
     population;
  species T_n in population, T_m in population, T_w in population;

  // Assignment Rules:
  N_pop := S + I_n + I_m + I_w + T_n + T_m + T_w;
  infected_total := I_n + I_m + I_w + T_n + T_m + T_w;
  foi := base_transmission_rate*exp(-behavior_sensitivity*
    infected_total/N_pop)*(I_n + I_m + I_w +
    treated_transmission_rate_relative*(T_n + T_m + T_w))/N_pop;
  readout_observable_prevalence := 100*infected_total/N_pop;

  // Reactions:
  recruitment:  -> S; recruitment_rate;
  infection: S -> I_n; foi*S;
  progression_nm: I_n -> I_m; infected_normal_death_rate*I_n;
  progression_to_treatment: I_m => T_w; infected_normal_death_rate*I_m;
  death_s: S -> ; susceptible_death_rate*S;
  death_in: I_n -> ; infected_normal_death_rate*I_n;
  death_im: I_m -> ; infected_moderate_death_rate*I_m;
  death_iw: I_w -> ; infected_weak_death_rate*I_w;
  death_tn: T_n -> ; treated_normal_death_rate*T_n;
  death_tm: T_m -> ; treated_moderate_death_rate*T_m;
  death_tw: T_w -> ; treated_weak_death_rate*T_w;

  // Species initializations:
  S = susceptible_init;
  I_n = infected_normal_init;
  I_m = infected_moderate_init;
  I_w = infected_weak_init;
  T_n = ;
  T_m = ;
  T_w = ;

  // Compartment initializations:
  population = 1;

  // Variable initializations:
  susceptible_init = ;
  infected_normal_init = ;
  infected_moderate_init = ;
  infected_weak_init = ;
  base_transmission_rate = 1.6455;
  behavior_sensitivity = ;
  treated_transmission_rate_relative = ;
  recruitment_rate = ;
  infected_normal_death_rate = ;
  susceptible_death_rate = ;
  infected_moderate_death_rate = ;
  infected_weak_death_rate = ;
  treated_normal_death_rate = ;
  treated_moderate_death_rate = ;
```

```
  treated_weak_death_rate = ;

  // Other declarations:
  var N_pop, infected_total, foi, readout_observable_prevalence;
  const population, susceptible_init, infected_normal_init,
    infected_moderate_init;
  const infected_weak_init, base_transmission_rate,
    behavior_sensitivity;
  const treated_transmission_rate_relative, recruitment_rate,
    infected_normal_death_rate;
  const susceptible_death_rate, infected_moderate_death_rate,
    infected_weak_death_rate;
  const treated_normal_death_rate, treated_moderate_death_rate,
    treated_weak_death_rate;
end
```

Listing 40: Archived Antimony for `rahman_2`; task `56836848_40`; selected version 7

```
// Created by libAntimony v3.1.3
model *hiv_prevalence()

  // Compartments and Species:
  compartment pop;
  species S in pop, I_n in pop, I_m in pop, I_w in pop, T_n in pop, T_m
     in pop;
  species T_w in pop;

  // Assignment Rules:
  N_total := S + I_n + I_m + I_w + T_n + T_m + T_w;
  beta_t := beta_lo + (beta_hi - beta_lo)/(1 + exp(c_kt*(time - c_kt)))
    ;
  readout_observable_prevalence := 100*(I_n + I_m + I_w + T_n + T_m +
    T_w)/N_total;

  // Reactions:
  recruitment:  -> S; recruitment_rate;
  s_death: S -> ; susceptible_death_rate*S;
  transmission: S -> I_n; beta_t*S*(I_n + I_m + I_w +
    treated_transmission_rate_relative*(T_n + T_m + T_w))/N_total;
  in_death: I_n -> ; infected_normal_death_rate*I_n;
  im_death: I_m -> ; infected_moderate_death_rate*I_m;
  iw_death: I_w -> ; infected_weak_death_rate*I_w;
  tn_death: T_n -> ; treated_normal_death_rate*T_n;
  tm_death: T_m -> ; treated_moderate_death_rate*T_m;
  tw_death: T_w -> ; treated_weak_death_rate*T_w;

  // Species initializations:
  S = susceptible_init;
  I_n = infected_normal_init;
  I_m = infected_moderate_init;
  I_w = infected_weak_init;
  T_n = 0;
  T_m = 0;
  T_w = 0;

  // Compartment initializations:
  pop = 1;

  // Variable initializations:
  susceptible_init = ;
  infected_normal_init = ;
  infected_moderate_init = ;
  infected_weak_init = ;
  beta_lo = ;
  beta_hi = 0.9;
```

```
  c_kt = ;
  recruitment_rate = ;
  susceptible_death_rate = ;
  treated_transmission_rate_relative = ;
  infected_normal_death_rate = ;
  infected_moderate_death_rate = ;
  infected_weak_death_rate = ;
  treated_normal_death_rate = ;
  treated_moderate_death_rate = ;
  treated_weak_death_rate = ;

  // Other declarations:
  var N_total, beta_t, readout_observable_prevalence;
  const pop, susceptible_init, infected_normal_init,
    infected_moderate_init;
  const infected_weak_init, beta_lo, beta_hi, c_kt, recruitment_rate,
    susceptible_death_rate;
  const treated_transmission_rate_relative, infected_normal_death_rate,
     infected_moderate_death_rate;
  const infected_weak_death_rate, treated_normal_death_rate,
    treated_moderate_death_rate;
  const treated_weak_death_rate;
end
```

Listing 41: Archived Antimony for `rahman_3`; task `56836848_42`; selected version 8

```
// Created by libAntimony v3.1.3
model *hiv_staged_epidemic()

  // Compartments and Species:
  compartment population;
  species susceptible in population, infected_normal in population,
    infected_moderate in population;
  species infected_weak in population, treated_normal in population,
    treated_moderate in population;
  species treated_weak in population;

  // Assignment Rules:
  N_total := susceptible + infected_normal + infected_moderate +
    infected_weak + treated_normal + treated_moderate + treated_weak;
  beta_eff := beta_end + (beta_0 - beta_end)*exp(-time/T_behavior);
  readout_observable_prevalence := 100*(infected_normal +
    infected_moderate + infected_weak + treated_normal +
    treated_moderate + treated_weak)/N_total;

  // Reactions:
  transmission: susceptible -> infected_normal; beta_eff*susceptible*(
    infected_normal + infected_moderate + infected_weak +
    treated_transmission_rate_relative*(treated_normal +
    treated_moderate + treated_weak))/N_total;
  progression_nm: infected_normal -> infected_moderate; r_prog*
    infected_normal;
  progression_mw: infected_moderate -> infected_weak; r_prog*
    infected_moderate;
  treatment_entry: infected_weak -> treated_weak; infected_weak;
  improvement_wm: treated_weak -> treated_moderate; r_prog*treated_weak
    ;
  improvement_mn: treated_moderate -> treated_normal; r_prog*
    treated_moderate;
  recruitment:  -> susceptible; recruitment_rate;
  death_susceptible: susceptible -> ; susceptible_death_rate*
    susceptible;
  death_infected_normal: infected_normal -> ;
    infected_normal_death_rate*infected_normal;
```

```
  death_infected_moderate: infected_moderate -> ;
    infected_moderate_death_rate*infected_moderate;
  death_infected_weak: infected_weak -> ; infected_weak_death_rate*
    infected_weak;
  death_treated_normal: treated_normal -> ; treated_normal_death_rate*
    treated_normal;
  death_treated_moderate: treated_moderate -> ;
    treated_moderate_death_rate*treated_moderate;
  death_treated_weak: treated_weak -> ; treated_weak_death_rate*
    treated_weak;

  // Species initializations:
  susceptible = susceptible_init;
  infected_normal = infected_normal_init;
  infected_moderate = infected_moderate_init;
  infected_weak = infected_weak_init;
  treated_normal = 0;
  treated_moderate = 0;
  treated_weak = 0;

  // Compartment initializations:
  population = 1;

  // Variable initializations:
  susceptible_init = 17940000;
  infected_normal_init = 16300;
  infected_moderate_init = 9000;
  infected_weak_init = 11000;
  beta_0 = 1;
  T_behavior = 13;
  beta_end = ;
  treated_transmission_rate_relative = ;
  r_prog = ;
  recruitment_rate = ;
  susceptible_death_rate = ;
  infected_normal_death_rate = ;
  infected_moderate_death_rate = ;
  infected_weak_death_rate = ;
  treated_normal_death_rate = ;
  treated_moderate_death_rate = ;
  treated_weak_death_rate = ;

  // Other declarations:
  var N_total, beta_eff, readout_observable_prevalence;
  const population, susceptible_init, infected_normal_init,
    infected_moderate_init;
  const infected_weak_init, beta_0, T_behavior, beta_end,
    treated_transmission_rate_relative;
  const r_prog, recruitment_rate, susceptible_death_rate,
    infected_normal_death_rate;
  const infected_moderate_death_rate, infected_weak_death_rate,
    treated_normal_death_rate;
  const treated_moderate_death_rate, treated_weak_death_rate;
end
```

Listing 42: Archived Antimony for `rahman_4`; task `56836848_43`; selected version 4

```
// Created by libAntimony v3.1.3
model *hiv_epidemic()

  // Compartments and Species:
  compartment pop;
  species S in pop, I_norm in pop, I_mod in pop, I_weak in pop, T_norm
    in pop;
  species T_mod in pop, T_weak in pop;

  // Assignment Rules:
  I_total := I_norm + I_mod + I_weak + T_norm + T_mod + T_weak;
  N_total := S + I_total;
  beta_now := beta_floor + beta_excess*exp(-beta_floor*time);
  readout_observable_prevalence := 100*I_total/N_total;

  // Reactions:
  recruitment:  -> S; recruitment_rate;
  infection: S -> I_norm; beta_now*(I_norm + I_mod + I_weak +
    treated_transmission_rate_relative*(T_norm + T_mod + T_weak))*S/
    N_total;
  progression_nm: I_norm -> I_mod; beta_excess*I_norm;
  reconstitution_wm: T_weak -> T_mod; beta_floor*T_weak;
  reconstitution_mn: T_mod -> T_norm; beta_floor*T_mod;
  death_s: S -> ; susceptible_death_rate*S;
  death_in: I_norm -> ; infected_normal_death_rate*I_norm;
  death_im: I_mod -> ; infected_moderate_death_rate*I_mod;
  death_iw: I_weak -> ; infected_weak_death_rate*I_weak;
  death_tn: T_norm -> ; treated_normal_death_rate*T_norm;
  death_tm: T_mod -> ; treated_moderate_death_rate*T_mod;
  death_tw: T_weak -> ; treated_weak_death_rate*T_weak;
  progression_aid: I_mod -> T_weak; infected_moderate_death_rate*I_mod;

  // Species initializations:
  S = susceptible_init;
  I_norm = infected_normal_init;
  I_mod = infected_moderate_init;
  I_weak = infected_weak_init;
  T_norm = 0;
  T_mod = 0;
  T_weak = 0;

  // Compartment initializations:
  pop = 1;

  // Variable initializations:
  susceptible_init = ;
  infected_normal_init = ;
  infected_moderate_init = ;
  infected_weak_init = ;
  recruitment_rate = 1032672;
  susceptible_death_rate = 0.0288;
  infected_normal_death_rate = 0.0888;
  infected_moderate_death_rate = 0.1368;
  infected_weak_death_rate = 0.3108;
  treated_normal_death_rate = 0.0408;
  treated_moderate_death_rate = 0.0528;
  treated_weak_death_rate = 0.1752;
  treated_transmission_rate_relative = 0.04;
  beta_excess = 0.15;
  beta_floor = 0.7;

  // Other declarations:
  var I_total, N_total, beta_now, readout_observable_prevalence;
```

```
  const pop, susceptible_init, infected_normal_init,
    infected_moderate_init;
  const infected_weak_init, recruitment_rate, susceptible_death_rate,
    infected_normal_death_rate;
  const infected_moderate_death_rate, infected_weak_death_rate,
    treated_normal_death_rate;
  const treated_moderate_death_rate, treated_weak_death_rate,
    treated_transmission_rate_relative;
  const beta_excess, beta_floor;
end
```

Listing 43: Archived Antimony for `rahman_5`; task `56836848_44`; selected version 9

```
// Created by libAntimony v3.1.3
model *hiv_epidemic()

  // Compartments and Species:
  compartment pop;
  species S in pop, I_N in pop, I_M in pop, I_W in pop, T_N in pop, T_M
     in pop;
  species T_W in pop;

  // Assignment Rules:
  N_total := S + I_N + I_M + I_W + T_N + T_M + T_W;
  readout_observable_prevalence := 100*(I_N + I_M + I_W + T_N + T_M +
    T_W)/N_total;

  // Reactions:
  recruitment:  -> S; recruitment_rate;
  treatment_entry: I_W -> T_W; tau_treat*I_W;
  death_s: S -> ; susceptible_death_rate*S;
  death_in: I_N -> ; infected_normal_death_rate*I_N;
  death_im: I_M -> ; infected_moderate_death_rate*I_M;
  death_iw: I_W -> ; infected_weak_death_rate*I_W;
  death_tn: T_N -> ; treated_normal_death_rate*T_N;
  death_tm: T_M -> ; treated_moderate_death_rate*T_M;
  death_tw: T_W -> ; treated_weak_death_rate*T_W;
  infection_w: S -> I_W; beta_trans*S*(I_N + I_M + I_W +
    treated_transmission_rate_relative*(T_N + T_M + T_W))/(N_total*(1 +
     eta_saturation*(I_N + I_M + I_W + T_N + T_M + T_W)/N_total));

  // Species initializations:
  S = susceptible_init;
  I_N = infected_normal_init;
  I_M = infected_moderate_init;
  I_W = infected_weak_init;
  T_N = 0;
  T_M = 0;
  T_W = 0;

  // Compartment initializations:
  pop = 1;

  // Variable initializations:
  susceptible_init = ;
  infected_normal_init = ;
  infected_moderate_init = ;
  infected_weak_init = ;
  beta_trans = 0.7;
  tau_treat = 0.2;
  recruitment_rate = ;
  susceptible_death_rate = ;
  infected_normal_death_rate = ;
  infected_moderate_death_rate = ;
  infected_weak_death_rate = ;
```

```
  treated_normal_death_rate = ;
  treated_moderate_death_rate = ;
  treated_weak_death_rate = ;
  treated_transmission_rate_relative = ;
  eta_saturation = ;

  // Other declarations:
  var N_total, readout_observable_prevalence;
  const pop, susceptible_init, infected_normal_init,
    infected_moderate_init;
  const infected_weak_init, beta_trans, tau_treat, recruitment_rate,
    susceptible_death_rate;
  const infected_normal_death_rate, infected_moderate_death_rate,
    infected_weak_death_rate;
  const treated_normal_death_rate, treated_moderate_death_rate,
    treated_weak_death_rate;
  const treated_transmission_rate_relative, eta_saturation;
end
```

H.4 BRANNMARK

Listing 44: Archived Antimony for `brannmark_1`; task `56836848_15`; selected version 17

```
// Created by libAntimony v3.1.3
model *insulin_signaling()

  // Compartments and Species:
  compartment cell;
  species R_a in cell, R_p in cell, R_i in cell, IRS1 in cell, S_p in
    cell;

  // Assignment Rules:
  insulin := insulin_dose_1*piecewise(0, time < insulin_time_1, 1) +
    insulin_dose_2*piecewise(0, time < insulin_time_2, 1);
  readout_IR_P := R_p;
  readout_IRS1_P := S_p;

  // Reactions:
  ir_autophos: R_a => R_p; k_rec_p_eff*insulin^n_H*R_a;
  ir_internalize: R_p => R_i; k_ptp*R_p;
  ir_surf_dephos: R_p => R_a; k_ptp*R_p;
  ir_recycle: R_i => R_a; k_rec_p_eff*R_i;
  irs1_phos: IRS1 => S_p; k_phos*R_p*IRS1;
  irs1_dephos: S_p => IRS1; k_ptp*S_p;
  ir_degradation: R_i => ; k_deg*R_i;

  // Species initializations:
  R_a = R_a_init;
  R_p = 0;
  R_i = 0;
  IRS1 = IRS1_init;
  S_p = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_a_init = 1;
  IRS1_init = 1;
  insulin_time_1 = 0;
  insulin_dose_1 = 0;
  insulin_time_2 = 0;
  insulin_dose_2 = 0;
  n_H = 0.6;
```

```
  k_phos = 0.5;
  k_rec_p_eff = ;
  k_ptp = ;
  k_deg = ;

  // Other declarations:
  var insulin, readout_IR_P, readout_IRS1_P;
  const cell, R_a_init, IRS1_init, insulin_time_1, insulin_dose_1,
    insulin_time_2;
  const insulin_dose_2, n_H, k_phos, k_rec_p_eff, k_ptp, k_deg;
end
```

Listing 45: Archived Antimony for `brannmark_2`; task `56836848_16`; selected version 12

```
// Created by libAntimony v3.1.3
model *insulin_signaling()

  // Compartments and Species:
  species R, Rp, Rd;

  // Assignment Rules:
  insulin := insulin_dose_1*piecewise(0, time < insulin_time_1, 1) +
    insulin_dose_2*piecewise(0, time < insulin_time_2, 1);
  occ := insulin/(Kd + insulin);
  readout_IR_P := Rp;
  readout_IRS1_P := 1 + Kd*Rp;

  // Reactions:
  receptor_activation: R -> Rp; k_act*occ*R;
  receptor_dephosphorylation: Rp -> R; k_dp*Rp;
  receptor_internalization: Rp -> Rd; k_int*Rp;
  receptor_recycling: Rd -> R; k_dp*Rd;
  receptor_degradation: Rd -> ; k_deg*Rd;

  // Species initializations:
  R = R_init;
  Rp = 0;
  Rd = 0;

  // Variable initializations:
  R_init = 1;
  insulin_time_1 = 0;
  insulin_dose_1 = 0;
  insulin_time_2 = 0;
  insulin_dose_2 = 0;
  Kd = 2.5;
  k_act = 3;
  k_dp = 0.1;
  k_int = 0.1;
  k_deg = ;

  // Other declarations:
  var insulin, occ, readout_IR_P, readout_IRS1_P;
  const R_init, insulin_time_1, insulin_dose_1, insulin_time_2,
    insulin_dose_2;
  const Kd, k_act, k_dp, k_int, k_deg;
end
```

Listing 46: Archived Antimony for `brannmark_3`; task `56836848_17`; selected version 6

```
// Created by libAntimony v3.1.3
model *insulin_signaling()

  // Compartments and Species:
  compartment cell;
  species R in cell, Rp in cell, Ri in cell, IRS1 in cell, IRS1P in
    cell;

  // Assignment Rules:
  insulin := insulin_dose_1*piecewise(0, time < insulin_time_1, 1) +
    insulin_dose_2*piecewise(0, time < insulin_time_2, 1);
  readout_IR_P := Rp;
  readout_IRS1_P := IRS1P;

  // Reactions:
  ir_activation: R -> Rp; (k_act*insulin^n_ins/(Kd_ins^n_ins + insulin^
    n_ins))*R;
  ir_dephos: Rp -> R; k_deph*Rp;
  ir_internalization: Rp -> Ri; k_deph*Rp;
  irs1_phos: IRS1 -> IRS1P; k_pho*Rp*IRS1;
  irs1_basal_phos: IRS1 -> IRS1P; k_bas*IRS1;
  irs1_dephos: IRS1P -> IRS1; k_deph*IRS1P;
  ir_downreg: Rp -> ; k_des*Rp;
  ir_recycling: Ri -> R; k_rec*Ri;

  // Species initializations:
  R = R_init;
  Rp = 0;
  Ri = 0;
  IRS1 = IRS1_init;
  IRS1P = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 1;
  IRS1_init = 1;
  insulin_dose_1 = 0;
  insulin_time_1 = 0;
  insulin_dose_2 = 0;
  insulin_time_2 = 4;
  k_act = 1;
  Kd_ins = 2;
  n_ins = 1.5;
  k_deph = 0.15;
  k_rec = 0.02;
  k_pho = 1;
  k_bas = 0.01;
  k_des = ;

  // Other declarations:
  var insulin, readout_IR_P, readout_IRS1_P;
  const cell, R_init, IRS1_init, insulin_dose_1, insulin_time_1,
    insulin_dose_2;
  const insulin_time_2, k_act, Kd_ins, n_ins, k_deph, k_rec, k_pho,
    k_bas;
  const k_des;
end
```

Listing 47: Archived Antimony for `brannmark_4`; task `56836848_18`; selected version 12

```
// Created by libAntimony v3.1.3
model *insulin_IRS1_min()

  // Compartments and Species:
  compartment cell;
  species IR in cell, IRp in cell, X2 in cell;

  // Assignment Rules:
  insulin := insulin_dose_1*piecewise(0, time < insulin_time_1, 1) +
    insulin_dose_2*piecewise(0, time < insulin_time_2, 1);
  IRS1p_lin := IRp + r0;
  readout_IR_P := IRp;
  readout_IRS1_P := IRS1p_lin;

  // Reactions:
  ir_activation: IR -> IRp; k_act*(insulin/(Kd_ins + insulin))*IR;
  ir_dephos: IRp -> IR; k_degX2*(1 + X2)*IRp;
  x2_prod:  -> X2; k_degX2*A_brk*IRS1p_lin*IRS1p_lin;
  x2_deg: X2 -> ; k_degX2*X2;

  // Species initializations:
  IR = IR_init;
  IRp = 0;
  X2 = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  IR_init = 1;
  k_act = 1.34411036500783;
  Kd_ins = 3.95594896058505;
  k_degX2 = 0.157152426477848;
  A_brk = 268.940963211424;
  r0 = 0.0207197422328594;
  insulin_time_1 = 0;
  insulin_dose_1 = 0;
  insulin_time_2 = 4;
  insulin_dose_2 = 0;

  // Other declarations:
  var insulin, IRS1p_lin, readout_IR_P, readout_IRS1_P;
  const cell, IR_init, k_act, Kd_ins, k_degX2, A_brk, r0,
    insulin_time_1;
  const insulin_dose_1, insulin_time_2, insulin_dose_2;
end
```

Listing 48: Archived Antimony for `brannmark_5`; task `56836848_19`; selected version 5

```
// Created by libAntimony v3.1.3
model *insulin_signaling()

  // Compartments and Species:
  compartment cell;
  species R in cell, Rp in cell, Rd in cell, S in cell, Sp in cell;

  // Assignment Rules:
  insulin := insulin_dose_1*piecewise(0, time < insulin_time_1, 1) +
    insulin_dose_2*piecewise(0, time < insulin_time_2, 1);
  readout_IR_P := Rp;
  readout_IRS1_P := Sp;

  // Reactions:
```

```
  receptor_activation: R => Rp; (k_act*insulin^0.7/(Km_act^0.7 +
    insulin^0.7))*R;
  receptor_dephos: Rp => R; k_deP*Rp;
  receptor_internalization: Rp => Rd; k_int*insulin*Rp;
  receptor_recycling: Rd => R; k_rec*Rd;
  irs1_phos: S => Sp; k_phos*Rp*S;
  irs1_dephos: Sp => S; k_deP*Sp;
  receptor_degradation: Rd => ; k_deg*Rd;

  // Species initializations:
  R = R_init;
  Rp = 0;
  Rd = 0;
  S = S_init;
  Sp = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 1;
  S_init = 1;
  insulin_time_1 = 0;
  insulin_dose_1 = 0;
  insulin_time_2 = 0;
  insulin_dose_2 = 0;
  k_act = 10;
  Km_act = 30;
  k_deP = 1;
  k_int = 0.004;
  k_rec = 0.3;
  k_phos = 10;
  k_deg = ;

  // Other declarations:
  var insulin, readout_IR_P, readout_IRS1_P;
  const cell, R_init, S_init, insulin_time_1, insulin_dose_1,
    insulin_time_2;
  const insulin_dose_2, k_act, Km_act, k_deP, k_int, k_rec, k_phos,
    k_deg;
end
```

H.5 BOEHM

Listing 49: Archived Antimony for `boehm_1`; task `56836848_10`; selected version 13

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cyt, nuc;
species STAT5A_c in cyt, pSTAT5A_c in cyt, STAT5B_c in cyt, pSTAT5B_c
    in cyt;
species STAT5A_n in nuc, pSTAT5A_n in nuc, STAT5B_n in nuc, pSTAT5B_n
    in nuc;

// Assignment Rules:
BaF3_Epo := Epo_dose*exp(-Epo_degradation_BaF3*time);
readout_pSTAT5A_rel := pSTAT5A_c/(STAT5A_c + pSTAT5A_c);
readout_pSTAT5B_rel := pSTAT5B_c/(STAT5B_c + pSTAT5B_c);
readout_rSTAT5A_rel := specC17*(STAT5A_c + pSTAT5A_c)/(specC17*(
    STAT5A_c + pSTAT5A_c) + cyt*(STAT5B_c + pSTAT5B_c));

// Reactions:
stat5a_phosphorylation: STAT5A_c => pSTAT5A_c; k_shut_b*BaF3_Epo*
    STAT5A_c/Epo_dose;
```

```
stat5b_phosphorylation: STAT5B_c => pSTAT5B_c; k_drive_b*BaF3_Epo*
    STAT5B_c/Epo_dose;
stat5a_import: pSTAT5A_c => pSTAT5A_n; k_imp_a*pSTAT5A_c;
stat5b_import: pSTAT5B_c => pSTAT5B_n; k_imp_b*pSTAT5B_c;
stat5a_nuc_dephos: pSTAT5A_n => STAT5A_n; k_dephos_nuc_a*pSTAT5A_n;
stat5b_nuc_dephos: pSTAT5B_n => STAT5B_n; k_dephos_nuc_a*pSTAT5B_n;
stat5a_export: STAT5A_n => STAT5A_c; k_dephos_nuc_a*STAT5A_n;
stat5b_export: STAT5B_n => STAT5B_c; k_dephos_nuc_a*STAT5B_n;
stat5a_constitutive_shuttle: STAT5A_c => STAT5A_n; k_shut_a*STAT5A_c;
stat5b_constitutive_shuttle: STAT5B_c => STAT5B_n; k_shut_b*STAT5B_c;

// Species initializations:
STAT5A_c = STAT5_total*ratio;
pSTAT5A_c = pSTAT5A_c_init;
STAT5B_c = STAT5_total*(1 - ratio);
pSTAT5B_c = pSTAT5B_c_init;
STAT5A_n = 0;
pSTAT5A_n = 0;
STAT5B_n = 0;
pSTAT5B_n = 0;

// Compartment initializations:
cyt = 1.4;
nuc = 0.45;

// Variable initializations:
STAT5_total = ;
ratio = ;
pSTAT5A_c_init = 11;
pSTAT5B_c_init = 3;
Epo_degradation_BaF3 = 0.02;
k_imp_a = 0.1;
k_imp_b = 0.1;
k_dephos_nuc_a = 0.1;
Epo_dose = ;
specC17 = ;
k_shut_b = ;
k_drive_b = ;
k_shut_a = ;

// Other declarations:
var BaF3_Epo, readout_pSTAT5A_rel, readout_pSTAT5B_rel,
    readout_rSTAT5A_rel;
const cyt, nuc, STAT5_total, ratio, pSTAT5A_c_init, pSTAT5B_c_init,
    Epo_degradation_BaF3;
const k_imp_a, k_imp_b, k_dephos_nuc_a, Epo_dose, specC17, k_shut_b,
    k_drive_b;
const k_shut_a;
```

Listing 50: Archived Antimony for `boehm_2`; task `56836848_11`; selected version 3

```
// Created by libAntimony v3.1.3
model *BaF3_Epo_STAT5_collapsed()

  // Compartments and Species:
  compartment cyt, nuc;
  species STAT5A_c in cyt, pSTAT5A_c in cyt, STAT5B_c in cyt, pSTAT5B_c
     in cyt;
  species STAT5A_n in nuc, pSTAT5A_n in nuc, STAT5B_n in nuc, pSTAT5B_n
     in nuc;

  // Assignment Rules:
  BaF3_Epo := Epo_dose*exp(-Epo_degradation_BaF3*time);
  cytA_mols := STAT5A_c + pSTAT5A_c;
  cytB_mols := STAT5B_c + pSTAT5B_c;
```

```
  nucA_mols := STAT5A_n + pSTAT5A_n;
  Atot_mols := cytA_mols + nucA_mols;
  readout_pSTAT5A_rel := ((1 - specC17)*pSTAT5A_c + specC17*pSTAT5B_c)
    /((1 - specC17)*cytA_mols + specC17*cytB_mols);
  readout_pSTAT5B_rel := ((1 - specC17)*pSTAT5B_c + specC17*pSTAT5A_c)
    /((1 - specC17)*cytB_mols + specC17*cytA_mols);
  readout_rSTAT5A_rel := (2*specC17*Atot_mols + nucA_mols)/STAT5_total;

  // Reactions:
  phos_A: STAT5A_c -> pSTAT5A_c; k_phos_A*BaF3_Epo*STAT5A_c*cyt;
  phos_B: STAT5B_c -> pSTAT5B_c; k_phos_B*BaF3_Epo*STAT5B_c*cyt;
  imp_A: pSTAT5A_c -> pSTAT5A_n; k_imp*pSTAT5A_c*cyt;
  imp_B: pSTAT5B_c -> pSTAT5B_n; k_imp*pSTAT5B_c*cyt;
  dephos_A: pSTAT5A_n -> STAT5A_n; k_exp*pSTAT5A_n*nuc;
  dephos_B: pSTAT5B_n -> STAT5B_n; k_exp*pSTAT5B_n*nuc;
  exp_A: STAT5A_n -> STAT5A_c; k_exp*STAT5A_n*nuc;
  exp_B: STAT5B_n -> STAT5B_c; k_exp*STAT5B_n*nuc;

  // Species initializations:
  STAT5A_c = STAT5_total*ratio;
  pSTAT5A_c = 0;
  STAT5B_c = STAT5_total*(1 - ratio);
  pSTAT5B_c = 0;
  STAT5A_n = 0;
  pSTAT5A_n = 0;
  STAT5B_n = 0;
  pSTAT5B_n = 0;

  // Compartment initializations:
  cyt = 1.4;
  nuc = 0.45;

  // Variable initializations:
  STAT5_total = ;
  ratio = ;
  Epo_dose = ;
  Epo_degradation_BaF3 = 0.021;
  k_phos_A = 3900000;
  k_phos_B = 1050000;
  k_imp = 0.0265;
  k_exp = 0.334;
  specC17 = ;

  // Other declarations:
  var BaF3_Epo, cytA_mols, cytB_mols, nucA_mols, Atot_mols,
    readout_pSTAT5A_rel;
  var readout_pSTAT5B_rel, readout_rSTAT5A_rel;
  const cyt, nuc, STAT5_total, ratio, Epo_dose, Epo_degradation_BaF3,
    k_phos_A;
  const k_phos_B, k_imp, k_exp, specC17;
end
```

Listing 51: Archived Antimony for `boehm_3`; task `56836848_12`; selected version 4

```
// Created by libAntimony v3.1.3
model *STAT5_epo_BaF3()

  // Compartments and Species:
  compartment cyt, nuc;
  species STAT5A in cyt, STAT5B in cyt, pA_m in cyt, pB_m in cyt,
    nSTAT5A in nuc;
  species nSTAT5B in nuc;

  // Assignment Rules:
  BaF3_Epo := Epo_dose*exp(-Epo_degradation_BaF3*time);
```

```
  readout_pSTAT5A_rel := pA_m/(STAT5A + pA_m);
  readout_pSTAT5B_rel := pB_m/(STAT5B + pB_m);
  readout_rSTAT5A_rel := specC17*(STAT5A + pA_m + STAT5B + pB_m + w_p*(
    pA_m + pB_m))/(STAT5A + pA_m + specC17*(STAT5B + pB_m));

  // Reactions:
  STAT5A_phos: STAT5A -> pA_m; k_phos_A*BaF3_Epo*STAT5A + k_bas_A*
    STAT5A;
  STAT5B_phos: STAT5B -> pB_m; k_phos_B*(BaF3_Epo/Epo_dose)^n_B*
    Epo_dose*STAT5B + k_bas_B*STAT5B;
  exp_B: nSTAT5B -> STAT5B; k_exp_B*nSTAT5B;
  imp_A_u: pA_m -> nSTAT5A; k_imp*pA_m;
  imp_B_u: pB_m -> nSTAT5B; k_imp*pB_m;

  // Species initializations:
  STAT5A = STAT5_total*ratio;
  STAT5B = STAT5_total*(1 - ratio);
  pA_m = pA_m_init;
  pB_m = 0;
  nSTAT5A = 0;
  nSTAT5B = 0;

  // Compartment initializations:
  cyt = 1.4;
  nuc = 0.45;

  // Variable initializations:
  STAT5_total = ;
  ratio = ;
  pA_m_init = 11.37;
  Epo_degradation_BaF3 = 0.02;
  Epo_dose = ;
  specC17 = ;
  w_p = ;
  k_phos_A = ;
  k_bas_A = ;
  k_phos_B = ;
  n_B = ;
  k_bas_B = ;
  k_exp_B = ;
  k_imp = ;

  // Other declarations:
  var BaF3_Epo, readout_pSTAT5A_rel, readout_pSTAT5B_rel,
    readout_rSTAT5A_rel;
  const cyt, nuc, STAT5_total, ratio, pA_m_init, Epo_degradation_BaF3,
    Epo_dose;
  const specC17, w_p, k_phos_A, k_bas_A, k_phos_B, n_B, k_bas_B,
    k_exp_B;
  const k_imp;
end
```

Listing 52: Archived Antimony for `boehm_4`; task `56836848_13`; selected version 5

```
// Created by libAntimony v3.1.3
model *STAT5_Epo_BaF3()

  // Compartments and Species:
  compartment cyt, nuc;
  species STAT5A in cyt, STAT5B in cyt, pA in cyt, pB in cyt, AA_n in
    nuc;
  species BB_n in nuc, AB_n;

  // Assignment Rules:
  BaF3_Epo := Epo_dose*exp(-Epo_degradation_BaF3*time);
```

```
  readout_pSTAT5A_rel := pA/(STAT5A + pA);
  readout_pSTAT5B_rel := pB/(STAT5B + pB);
  readout_rSTAT5A_rel := specC17*(STAT5A + pA)/(specC17*(STAT5A + pA) +
     cyt*(STAT5B + pB));

  // Reactions:
  phos_A: STAT5A => pA; cyt*k_phosA*BaF3_Epo*STAT5A;
  phos_B: STAT5B => pB; cyt*k_phosB*BaF3_Epo*STAT5B;
  seq_AA: 2 pA => AA_n; cyt*k_dimAA*pA*pA;
  seq_BB: 2 pB => BB_n; cyt*k_dimBB*pB*pB;
  rec_A: AA_n => 2 STAT5A; nuc*k_retA*AA_n;
  rec_B: BB_n => 2 STAT5B; nuc*k_retB*BB_n;
  dim_AB: pA + pB => AB_n; cyt*k_dimAB*pA*pB;
  rec_AB: AB_n => STAT5A + STAT5B; nuc*k_retAB*AB_n;

  // Species initializations:
  STAT5A = STAT5_total*ratio;
  STAT5B = STAT5_total*(1 - ratio);
  pA = 0;
  pB = 0;
  AA_n = 0;
  BB_n = 0;
  AB_n = ;

  // Compartment initializations:
  cyt = 1.4;
  nuc = 0.45;

  // Variable initializations:
  STAT5_total = ;
  ratio = ;
  Epo_dose = ;
  Epo_degradation_BaF3 = ;
  specC17 = ;
  k_phosA = ;
  k_phosB = ;
  k_dimAA = ;
  k_dimBB = ;
  k_retA = ;
  k_retB = ;
  k_dimAB = ;
  k_retAB = ;

  // Other declarations:
  var BaF3_Epo, readout_pSTAT5A_rel, readout_pSTAT5B_rel,
    readout_rSTAT5A_rel;
  const cyt, nuc, STAT5_total, ratio, Epo_dose, Epo_degradation_BaF3,
    specC17;
  const k_phosA, k_phosB, k_dimAA, k_dimBB, k_retA, k_retB, k_dimAB,
    k_retAB;
end
```

Listing 53: Archived Antimony for `boehm_5`; task `56836848_14`; selected version 12

```
// Created by libAntimony v3.1.3
model *Epo_STAT5_AB()

  // Compartments and Species:
  compartment cyt, nuc;
  species STAT5A in cyt, STAT5B in cyt, pSTAT5A in cyt, pSTAT5B in cyt,
     pApB in cyt;
  species nucSTAT5A in nuc, nucSTAT5B in nuc;

  // Assignment Rules:
  BaF3_Epo := Epo_dose*exp(-Epo_degradation_BaF3*time);
```

```
  EpoR_act := Epo_dose*(exp(-Epo_degradation_BaF3*time) - exp(-k_deact1*time))/(k_deact1 - Epo_degradation_BaF3);
  readout_pSTAT5A_rel := (pSTAT5A + pApB)/(STAT5A + pSTAT5A + pApB);
  readout_pSTAT5B_rel := (pSTAT5B + pApB)/(STAT5B + pSTAT5B + pApB);
  readout_rSTAT5A_rel := specC17*(cyt*(STAT5A + pSTAT5A + pApB + STAT5B + pSTAT5B) + nuc*(nucSTAT5A + nucSTAT5B))/(cyt*(STAT5A + pSTAT5A + pApB));

  // Reactions:
  stat5a_phos: STAT5A => pSTAT5A; k_phos_A*EpoR_act*STAT5A*cyt;
  stat5b_phos: STAT5B => pSTAT5B; k_phos_B*EpoR_act*STAT5B*cyt;
  pA_cyt_dephos: pSTAT5A => STAT5A; k_deg_A*pSTAT5A*cyt;
  pB_cyt_dephos: pSTAT5B => STAT5B; k_deg_B*pSTAT5B*cyt;
  pApB_form: pSTAT5A + pSTAT5B => pApB; k_dim_AB*pSTAT5A*pSTAT5B*cyt;
  pApB_dissoc: pApB => STAT5A + STAT5B; k_deg_B*pApB*cyt;
  A_nuc_import: STAT5A => nucSTAT5A; k_phos_A*BaF3_Epo*STAT5A*cyt;
  B_nuc_import: STAT5B => nucSTAT5B; k_phos_B*BaF3_Epo*STAT5B*cyt;
  A_nuc_export: nucSTAT5A => STAT5A; k_exp*nucSTAT5A*nuc;
  B_nuc_export: nucSTAT5B => STAT5B; k_exp*nucSTAT5B*nuc;

  // Species initializations:
  STAT5A = STAT5_total*ratio;
  STAT5B = STAT5_total*(1 - ratio);
  pSTAT5A = pSTAT5A_init;
  pSTAT5B = pSTAT5B_init;
  pApB = 0;
  nucSTAT5A = 0;
  nucSTAT5B = 0;

  // Compartment initializations:
  cyt = 1.4;
  nuc = 0.45;

  // Variable initializations:
  STAT5_total = ;
  ratio = ;
  pSTAT5A_init = 7;
  pSTAT5B_init = 3;
  Epo_degradation_BaF3 = 0.02;
  Epo_dose = ;
  k_phos_A = 200000000;
  k_phos_B = 90000000;
  k_deg_A = 1.1;
  k_deg_B = 2;
  k_dim_AB = 0.01;
  k_exp = 0.002;
  k_deact1 = ;
  specC17 = ;

  // Other declarations:
  var BaF3_Epo, EpoR_act, readout_pSTAT5A_rel, readout_pSTAT5B_rel, readout_rSTAT5A_rel;
  const cyt, nuc, STAT5_total, ratio, pSTAT5A_init, pSTAT5B_init, Epo_degradation_BaF3;
  const Epo_dose, k_phos_A, k_phos_B, k_deg_A, k_deg_B, k_dim_AB, k_exp, k_deact1;
  const specC17;
end
```

H.6 ARMISTEAD

Listing 54: Archived Antimony for armistead_1; task 56836848_0; selected version 2

```
// Created by libAntimony v3.1.3
model *sphingolipid_hai1a()

  // Compartments and Species:
  compartment cell;
  species Sphinga in cell, Cer in cell, Sphingo in cell, S1P in cell;

  // Assignment Rules:
  readout_Sphinga_obs := Sphinga;
  readout_Cer_obs := Cer;
  readout_Sphingo_obs := Sphingo;
  readout_S1P_obs := S1P;

  // Reactions:
  csl_to_cer:  -> Cer; u_cer;
  cer_exit: Cer -> ; k_cer_exit*S_on*Cer;
  sphinga_exit: Sphinga -> ; k_sphinga*(S_on + m_sphinga_mut*(1 - S_on)
    )*Sphinga;
  sphingo_exit: Sphingo -> ; k_cer_exit*(S_on + m_sphinga_mut*(1 - S_on
    ))*Sphingo;

  // Species initializations:
  Sphinga = 0.000732;
  Cer = 0.988;
  Sphingo = 0.000395;
  S1P = 1.551;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  S_on = 1;
  u_cer = 0.05;
  k_cer_exit = 0.23;
  k_sphinga = 0.76;
  m_sphinga_mut = 0.5;

  // Other declarations:
  var readout_Sphinga_obs, readout_Cer_obs, readout_Sphingo_obs,
    readout_S1P_obs;
  const cell, S_on, u_cer, k_cer_exit, k_sphinga, m_sphinga_mut;
end
```

Listing 55: Archived Antimony for armistead_2; task 56836848_1; selected version 3

```
// Created by libAntimony v3.1.3
model *sphingolipid_genotype()

  // Compartments and Species:
  compartment cell;
  species Sphinga in cell, Cer in cell, Sphingo in cell, S1P in cell;

  // Assignment Rules:
  u_c_wt := k_so_wt;
  u_c_mut := k_c;
  readout_Sphinga_obs := Sphinga;
  readout_Cer_obs := Cer;
  readout_Sphingo_obs := Sphingo;
  readout_S1P_obs := S1P;

  // Reactions:
```

```
  R1_de_novo_acylation: Sphinga -> Cer; (k_a_wt*S_on + k_a_mut*(1 -
    S_on))*Sphinga;
  R2_ceramide_production:  -> Cer; u_c_wt*S_on + u_c_mut*(1 - S_on);
  R3_ceramide_loss: Cer -> ; k_c*Cer;
  R4_sphingo_drain: Sphingo -> ; (k_so_wt*S_on + k_so_mut*(1 - S_on))*
    Sphingo;

  // Species initializations:
  Sphinga = ;
  Cer = ;
  Sphingo = ;
  S1P = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  S_on = 1;
  k_a_wt = 0.76;
  k_a_mut = 0.6;
  k_c = 0.5;
  k_so_wt = 0.23;
  k_so_mut = 0.11;

  // Other declarations:
  var u_c_wt, u_c_mut, readout_Sphinga_obs, readout_Cer_obs,
    readout_Sphingo_obs;
  var readout_S1P_obs;
  const cell, S_on, k_a_wt, k_a_mut, k_c, k_so_wt, k_so_mut;
end
```

Listing 56: Archived Antimony for `armistead_3`; task `56836848_2`; selected version 5

```
// Created by libAntimony v3.1.3
model *sphingolipid_salvage()

  // Compartments and Species:
  compartment cell;
  species Sphinga in cell, Cer in cell, Sphingo in cell, S1P in cell;

  // Assignment Rules:
  readout_Sphinga_obs := Sphinga;
  readout_Cer_obs := Cer;
  readout_Sphingo_obs := Sphingo;
  readout_S1P_obs := S1P;

  // Reactions:
  spha_net_loss: Sphinga -> ; (k_S0 + S_on*k_SW)*Sphinga;
  salv_in:  -> Cer; v_S0;
  cer_exit_wt: Cer -> ; S_on*k_CW*Cer;
  sph_form: Cer -> Sphingo; v_src*Cer;
  sph_loss: Sphingo -> ; k_S0*Sphingo;

  // Species initializations:
  Sphinga = 0.000732;
  Cer = 0.988;
  Sphingo = 0.000395;
  S1P = 1.551;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  S_on = 1;
  k_S0 = 1;
```

```
  k_SW = 0.16;
  v_S0 = 0.05;
  k_CW = 0.27;
  v_src = ;

  // Other declarations:
  var readout_Sphinga_obs, readout_Cer_obs, readout_Sphingo_obs,
    readout_S1P_obs;
  const cell, S_on, k_S0, k_SW, v_S0, k_CW, v_src;
end
```

Listing 57: Archived Antimony for `armistead_4`; task `56836848_3`; selected version 1

```
// Created by libAntimony v3.1.3
model *sphingolipid_genotype()

  // Compartments and Species:
  compartment cell;
  species Sphinga in cell, Cer in cell, Sphingo in cell, S1P in cell;

  // Assignment Rules:
  readout_Sphinga_obs := Sphinga;
  readout_Cer_obs := Cer;
  readout_Sphingo_obs := Sphingo;
  readout_S1P_obs := S1P;

  // Reactions:
  sphinga_acylation: Sphinga -> Cer; (kSpha_wt*S_on + kSpha_mut*(1 -
    S_on))*Sphinga;
  cer_drain_wt: Cer -> ; kCer_wt*S_on*Cer;
  cer_accum_mut:  -> Cer; kCer_g_mut*(1 - S_on)*Cer;
  sphingo_turnover: Sphingo -> ; (kCer_wt*S_on + kSphi_mut*(1 - S_on))*
    Sphingo;

  // Species initializations:
  Sphinga = 0.000732;
  Cer = 0.988;
  Sphingo = 0.000395;
  S1P = 1.551;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  S_on = 1;
  kSpha_wt = 0.76;
  kSpha_mut = 0.6;
  kCer_wt = 0.2;
  kCer_g_mut = 0.058;
  kSphi_mut = 0.11;

  // Other declarations:
  var readout_Sphinga_obs, readout_Cer_obs, readout_Sphingo_obs,
    readout_S1P_obs;
  const cell, S_on, kSpha_wt, kSpha_mut, kCer_wt, kCer_g_mut, kSphi_mut
    ;
end
```

Listing 58: Archived Antimony for `armistead_5`; task `56836848_4`; selected version 5

```
// Created by libAntimony v3.1.3
model *sphingolipid_metabolism()

  // Compartments and Species:
  compartment cell;
  species Sphinga in cell, Cer in cell, Sphingo in cell, S1P in cell;

  // Assignment Rules:
  readout_Sphinga_obs := Sphinga;
  readout_Cer_obs := Cer;
  readout_Sphingo_obs := Sphingo;
  readout_S1P_obs := S1P;

  // Reactions:
  s_cer_syn: Sphinga -> Cer; k_cs*(1 + S_on*f_s)*Sphinga;
  s_salvage:  -> Cer; k_cs*(1 + S_on*g);
  s_cer_exit: Cer -> ; k_cs*Cer;
  s_ceramidase: Cer -> Sphingo; k_cd*Cer;
  s_sphingo_exit: Sphingo -> ; k_cs*Sphingo;

  // Species initializations:
  Sphinga = 0.000732;
  Cer = 0.988;
  Sphingo = 0.000395;
  S1P = 1.551;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  S_on = 1;
  k_cs = 0.3;
  f_s = 0.3;
  k_cd = 5e-05;
  g = ;

  // Other declarations:
  var readout_Sphinga_obs, readout_Cer_obs, readout_Sphingo_obs,
    readout_S1P_obs;
  const cell, S_on, k_cs, f_s, k_cd, g;
end
```

H.7 FIEDLER

Listing 59: Archived Antimony for `fiedler_1`; task `57351389_5`; selected version 21

```
// Created by libAntimony v3.1.3
model *zero_shot_problem()

  // Compartments and Species:
  species pRAF, pMEK, pERK, Sorafenib, UO126, stimulus_on, I;

  // Assignment Rules:
  readout_pMek := pMEK + bkg_pMek;
  readout_pErk := pERK + bkg_pErk;

  // Reactions:
  raf_activation:  -> pRAF; stimulus_on*(RAF_total - pRAF)/(1 + 100000*
    I);
  raf_deactivation: pRAF -> ; k_deph*pRAF;
  mek_phosphorylation:  -> pMEK; k1*pRAF*(MEK_total - pMEK)*Kd_sor/(
    Kd_sor + Sorafenib);
```

```
  mek_dephosphorylation: pMEK -> ; k_deph*pMEK;
  erk_phosphorylation:  -> pERK; k3*pMEK*(ERK_total - pERK)*Kd_uo/(
    Kd_uo + UO126);
  erk_dephosphorylation: pERK -> ; k_deph*pERK;
  fb_mediator_production:  -> I; pERK;

  // Species initializations:
  pRAF = 0;
  pMEK = 0;
  pERK = 0;
  Sorafenib = 0;
  UO126 = 0;
  stimulus_on = 0;
  I = ;

  // Variable initializations:
  k1 = 1;
  k3 = 1;
  Kd_sor = 0.2;
  Kd_uo = 0.07;
  bkg_pMek = ;
  bkg_pErk = ;
  RAF_total = ;
  k_deph = ;
  MEK_total = ;
  ERK_total = ;

  // Other declarations:
  var readout_pMek, readout_pErk;
  const k1, k3, Kd_sor, Kd_uo, bkg_pMek, bkg_pErk, RAF_total, k_deph,
    MEK_total;
  const ERK_total;
end
```

Listing 60: Archived Antimony for `fiedler_2`; task `57351389_6`; selected version 10

```
// Created by libAntimony v3.1.3
model *raf_mek_erk_feedback()

  // Compartments and Species:
  species pRAF, pMEK, pERK, Mfb, Sorafenib, UO126, stimulus_on;

  // Assignment Rules:
  readout_pMek := pMEK;
  readout_pErk := pERK;

  // Reactions:
  raf_activation:  -> pRAF; k_s*stimulus_on*exp(-100*time)*(RAF_total -
     pRAF);
  raf_deactivation: pRAF -> ; k_d*pRAF;
  mek_phosphorylation:  -> pMEK; (k_b2 + pRAF/(1 + Sorafenib/IC50_sor))
    *(MEK_total - pMEK)/(1 + Mfb^n_fb);
  mek_dephosphorylation: pMEK -> ; k2*pMEK;
  erk_phosphorylation:  -> pERK; k3*pMEK*(ERK_total - pERK)/(1 + UO126/
    IC50_uo);
  erk_dephosphorylation: pERK -> ; k4*pERK;
  fb_synthesis:  -> Mfb; pERK;
  fb_degradation: Mfb -> ; k_m*Mfb;

  // Species initializations:
  pRAF = 0;
  pMEK = 0;
  pERK = 0;
  Mfb = 0;
  Sorafenib = 0;
```

```
  UO126 = 0;
  stimulus_on = 0;

  // Variable initializations:
  RAF_total = 1;
  MEK_total = 1;
  ERK_total = 1;
  k_s = 5;
  k_d = 0.1;
  k2 = 0.1;
  k3 = 1;
  k4 = 0.07;
  k_m = 0.2;
  n_fb = 2;
  IC50_sor = 0.5;
  IC50_uo = 1;
  k_b2 = ;

  // Other declarations:
  var readout_pMek, readout_pErk;
  const RAF_total, MEK_total, ERK_total, k_s, k_d, k2, k3, k4, k_m,
    n_fb;
  const IC50_sor, IC50_uo, k_b2;
end
```

Listing 61: Archived Antimony for `fiedler_3`; task `57351389_7`; selected version 16

```
// Created by libAntimony v3.1.3
model *raf_mek_erk_cascade()

  // Compartments and Species:
  compartment cell;
  species Sorafenib in cell, UO126 in cell, stimulus_on in cell, RAFa
    in cell;
  species $MEKp in cell, ERKp in cell;

  // Assignment Rules:
  MEKp := MEK_total*RAFa/(RAFa + K_M);
  Stim := stimulus_on*exp(-k_stim*time);
  readout_pMek := MEKp;
  readout_pErk := ERKp;

  // Reactions:
  raf_act_basal:  -> RAFa; K_M*(RAF_total - RAFa);
  raf_act_stim:  -> RAFa; 100*Stim*(RAF_total - RAFa)/(1 + Sorafenib/
    IC50_sor_raf);
  raf_deact: RAFa -> ; (k_erk_dephos + k_fb*ERKp)*RAFa;
  erk_phos:  -> ERKp; k_erk_dephos*MEKp*(ERK_total - ERKp)/(1 + UO126/
    IC50_uo);
  erk_dephos: ERKp -> ; k_erk_dephos*ERKp;

  // Species initializations:
  Sorafenib = 0;
  UO126 = 0;
  stimulus_on = 0;
  RAFa = 0;
  ERKp = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  MEK_total = ;
  K_M = 0.5;
  k_erk_dephos = 0.5;
```

```
  k_fb = 100;
  IC50_uo = 0.5;
  k_stim = ;
  RAF_total = ;
  IC50_sor_raf = ;
  ERK_total = ;

  // Other declarations:
  var Stim, readout_pMek, readout_pErk;
  const cell, MEK_total, K_M, k_erk_dephos, k_fb, IC50_uo, k_stim,
    RAF_total;
  const IC50_sor_raf, ERK_total;
end
```

Listing 62: Archived Antimony for `fiedler_4`; task `57351389_8`; selected version 16

```
// Created by libAntimony v3.1.3
// Compartments and Species:
species RAFa, MekP, ErkP, Xinh;

// Assignment Rules:
I_sor := 1/(1 + Sorafenib/Ki_sor);
I_uo := 1/(1 + UO126/Ki_uo);
Ffb := 1/(1 + (Xinh/V2)^n_fb);
readout_pMek := MekP;
readout_pErk := ErkP;

// Reactions:
raf_activation:  -> RAFa; V2*(RAF_total - RAFa)*I_sor*Ffb + V1*
    stimulus_on*(RAF_total - RAFa)*I_sor*Ffb;
raf_deactivation: RAFa -> ; V2*RAFa;
mek_phosphorylation:  -> MekP; V2*RAFa*(MEK_total - MekP);
mek_dephosphorylation: MekP -> ; V1*MekP;
erk_phosphorylation:  -> ErkP; V1*MekP*(ERK_total - ErkP)*I_uo;
erk_dephosphorylation: ErkP -> ; V2*ErkP;
feedback_recruitment:  -> Xinh; a_mk*MekP + ErkP;
feedback_decay: Xinh -> ; vfe*Xinh;

// Species initializations:
RAFa = 0;
MekP = 0;
ErkP = 0;
Xinh = 0;

// Variable initializations:
Sorafenib = 0;
UO126 = 0;
stimulus_on = 0;
V2 = 0.5;
V1 = 5;
vfe = 0.2;
n_fb = 2;
Ki_sor = 0.1;
Ki_uo = 0.3;
RAF_total = ;
MEK_total = ;
ERK_total = ;
a_mk = ;

// Other declarations:
var I_sor, I_uo, Ffb, readout_pMek, readout_pErk;
const Sorafenib, UO126, stimulus_on, V2, V1, vfe, n_fb, Ki_sor, Ki_uo,
    RAF_total;
const MEK_total, ERK_total, a_mk;
```

Listing 63: Archived Antimony for `fiedler_5`; task `57351389_9`; selected version 15

```
// Created by libAntimony v3.1.3
model *m5b()

  // Compartments and Species:
  species pRAF, pMEK, pERK, w, A_stim, Sorafenib, UO126, stimulus_on;

  // Assignment Rules:
  pRAF_b := k_phos_M/(k_dephos_E + k_phos_M);
  m_b := k_phos_M*pRAF_b/(k_dephos_E + k_phos_M*pRAF_b);
  e_b := k_phos_E*m_b/(k_dephos_E + k_phos_E*m_b);
  readout_pMek := pMEK/MEK_total;
  readout_pErk := pERK/ERK_total;

  // Reactions:
  raf_act:  => pRAF; A_stim*(k_phos_M + stimulus_on*k_dephos_E)*(
    RAF_total - pRAF)*e_b*e_b/w;
  raf_deact: pRAF => ; k_dephos_E*pRAF;
  mek_phos:  => pMEK; k_phos_M*pRAF*(MEK_total - pMEK)*Ki_sor/(Ki_sor +
     Sorafenib);
  mek_dephos: pMEK => ; k_dephos_E*pMEK;
  erk_phos:  => pERK; k_phos_E*pMEK*(ERK_total - pERK)*Ki_sor/(Ki_sor +
     UO126);
  erk_dephos: pERK => ; k_dephos_E*pERK;
  fb_charge:  => w; k_adapt*pERK*pERK;
  fb_decay: w => ; k_adapt*w;
  adapt_drive:  => A_stim; k_adapt*(1 - stimulus_on);
  adapt_relax: A_stim => ; k_adapt*A_stim;

  // Species initializations:
  pRAF = 0;
  pMEK = 0;
  pERK = 0;
  w = w_init;
  A_stim = A_stim_init;
  Sorafenib = 0.1;
  UO126 = 0.1;
  stimulus_on = 0.1;

  // Variable initializations:
  w_init = 1;
  A_stim_init = 1;
  k_phos_M = 0.47;
  k_dephos_E = 3.36;
  k_adapt = 0.068;
  k_phos_E = 682;
  Ki_sor = 0.92;
  MEK_total = ;
  ERK_total = ;
  RAF_total = ;

  // Other declarations:
  var pRAF_b, m_b, e_b, readout_pMek, readout_pErk;
  const w_init, A_stim_init, k_phos_M, k_dephos_E, k_adapt, k_phos_E,
    Ki_sor;
  const MEK_total, ERK_total, RAF_total;
end
```

H.8 BRUNO

Listing 64: Archived Antimony for `bruno_1`; task `56836848_20`; selected version 2

```
// Created by libAntimony v3.1.3
model *zero_shot_problem()

  // Compartments and Species:
  species bcar, bcry, zea, b10, ohb10, bio, ohbio;

  // Assignment Rules:
  readout_ob10 := b10;
  readout_obcar := bcar;
  readout_obcry := bcry;
  readout_obio := bio;
  readout_oohb10 := ohb10;
  readout_ozea := zea;

  // Reactions:
  bcar_cleave: bcar -> b10 + bio; kb1*kb1_multiplier*bcar;
  b10_cleave: b10 -> bio; k_apo*kb2_multiplier*b10;
  bcry_cleave_1: bcry -> b10 + ohbio; kc1*kc1_multiplier*bcry;
  bcry_cleave_2: bcry -> ohb10 + bio; k_apo*kc2_multiplier*bcry;
  zea_cleave: zea -> ohb10 + ohbio; k5*k5_multiplier*zea;
  ohb10_cleave: ohb10 -> ohbio; k_apo*kc4_multiplier*ohb10;

  // Species initializations:
  bcar = init_bcar;
  bcry = init_bcry;
  zea = init_zea;
  b10 = init_b10;
  ohb10 = init_ohb10;
  bio = 0;
  ohbio = 0;

  // Variable initializations:
  init_bcar = 0;
  init_bcry = 0;
  init_zea = 0;
  init_b10 = 0;
  init_ohb10 = 0;
  k5_multiplier = 0;
  kb1_multiplier = 0;
  kb2_multiplier = 0;
  kc1_multiplier = 0;
  kc2_multiplier = 0;
  kc4_multiplier = 0;
  kb1 = 0.014;
  k_apo = 0.0058;
  kc1 = 0.0017;
  k5 = 0.0033;

  // Other declarations:
  var readout_ob10, readout_obcar, readout_obcry, readout_obio,
    readout_oohb10;
  var readout_ozea;
  const init_bcar, init_bcry, init_zea, init_b10, init_ohb10,
    k5_multiplier;
  const kb1_multiplier, kb2_multiplier, kc1_multiplier, kc2_multiplier,
     kc4_multiplier;
  const kb1, k_apo, kc1, k5;
end
```

Listing 65: Archived Antimony for `bruno_2`; task `56836848_21`; selected version 6

```
// Created by libAntimony v3.1.3
model *carotenoid_panel()

  // Compartments and Species:
  compartment cell;
  species bcar in cell, bcry in cell, zea in cell, b10 in cell, ohb10
    in cell;
  species bio in cell, ohbio in cell, c14 in cell;

  // Assignment Rules:
  readout_obcar := bcar;
  readout_obcry := bcry;
  readout_ozea := zea;
  readout_ob10 := b10;
  readout_oohb10 := ohb10;
  readout_obio := bio;

  // Reactions:
  bcar_cleave: bcar -> b10 + bio; kb1*kb1_multiplier*bcar;
  bcry_to_b10: bcry -> b10 + ohbio; kc1*kc1_multiplier*bcry;
  bcry_to_ohb10: bcry -> ohb10 + bio; kb2*kc2_multiplier*bcry;
  zea_cleave: zea -> ohb10 + ohbio; k5*k5_multiplier*zea;
  b10_cleave: b10 -> bio + c14; kb2*kb2_multiplier*b10;
  ohb10_cleave: ohb10 -> ohbio + c14; kb2*kc4_multiplier*ohb10;

  // Species initializations:
  bcar = init_bcar;
  bcry = init_bcry;
  zea = init_zea;
  b10 = init_b10;
  ohb10 = init_ohb10;
  bio = 0;
  ohbio = 0;
  c14 = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  init_bcar = 0;
  init_bcry = 0;
  init_zea = 0;
  init_b10 = 0;
  init_ohb10 = 0;
  kb1_multiplier = 0;
  kb2_multiplier = 0;
  kc1_multiplier = 0;
  kc2_multiplier = 0;
  k5_multiplier = 0;
  kc4_multiplier = 0;
  kb1 = 0.01;
  kb2 = 0.006;
  kc1 = 0.002;
  k5 = 0.002;

  // Other declarations:
  var readout_obcar, readout_obcry, readout_ozea, readout_ob10,
    readout_oohb10;
  var readout_obio;
  const cell, init_bcar, init_bcry, init_zea, init_b10, init_ohb10,
    kb1_multiplier;
  const kb2_multiplier, kc1_multiplier, kc2_multiplier, k5_multiplier,
    kc4_multiplier;
```

```
  const kb1, kb2, kc1, k5;
end
```

Listing 66: Archived Antimony for `bruno_3`; task `56836848_22`; selected version 1

```
// Created by libAntimony v3.1.3
model *carotenoid_cleavage()

  // Compartments and Species:
  compartment assay;
  species bcar in assay, bcry in assay, zea in assay, b10 in assay,
    ohb10 in assay;
  species bio in assay, ohbio in assay;

  // Assignment Rules:
  kc4 := kb2;
  readout_ob10 := b10;
  readout_obcar := bcar;
  readout_obcry := bcry;
  readout_obio := bio;
  readout_oohb10 := ohb10;
  readout_ozea := zea;

  // Reactions:
  bcar_cleave: bcar -> b10 + bio; kb1*kb1_multiplier*bcar;
  b10_cleave: b10 -> bio; kb2*kb2_multiplier*b10;
  bcry_cleave_plain: bcry -> b10 + ohbio; kc1*kc1_multiplier*bcry;
  bcry_cleave_oh: bcry -> ohb10 + bio; kc2*kc2_multiplier*bcry;
  ohb10_cleave: ohb10 -> ohbio; kc4*kc4_multiplier*ohb10;
  zea_cleave: zea -> ohb10 + ohbio; k5*k5_multiplier*zea;

  // Species initializations:
  bcar = init_bcar;
  bcry = init_bcry;
  zea = init_zea;
  b10 = init_b10;
  ohb10 = init_ohb10;
  bio = 0;
  ohbio = 0;

  // Compartment initializations:
  assay = 1;

  // Variable initializations:
  init_bcar = 0;
  init_bcry = 0;
  init_zea = 0;
  init_b10 = 0;
  init_ohb10 = 0;
  k5_multiplier = 0;
  kb1_multiplier = 0;
  kb2_multiplier = 0;
  kc1_multiplier = 0;
  kc2_multiplier = 0;
  kc4_multiplier = 0;
  init_b10_1 = 4.22;
  init_bcar1 = 4.44;
  init_bcar2 = 3.3;
  init_bcry_1 = 5.22;
  init_ohb10_1 = 1.74;
  init_zea_1 = 2.96;
  szea = 0.517;
  kb1 = 0.015;
  kb2 = 0.007;
  kc1 = 0.002;
```

```
  kc2 = 0.008;
  k5 = 0.003;

  // Other declarations:
  var kc4, readout_ob10, readout_obcar, readout_obcry, readout_obio,
    readout_oohb10;
  var readout_ozea;
  const assay, init_bcar, init_bcry, init_zea, init_b10, init_ohb10,
    k5_multiplier;
  const kb1_multiplier, kb2_multiplier, kc1_multiplier, kc2_multiplier,
     kc4_multiplier;
  const init_b10_1, init_bcar1, init_bcar2, init_bcry_1, init_ohb10_1,
    init_zea_1;
  const szea, kb1, kb2, kc1, kc2, k5;
end
```

Listing 67: Archived Antimony for `bruno_4`; task `56836848_23`; selected version 5

```
// Created by libAntimony v3.1.3
model *carotenoid_ccd_assay()

  // Compartments and Species:
  compartment cell;
  species bcar in cell, bcry in cell, zea in cell, b10 in cell, ohb10a
    in cell;
  species io in cell, ohio in cell, c14dial in cell;

  // Assignment Rules:
  readout_ob10 := b10;
  readout_obcar := bcar;
  readout_obcry := bcry;
  readout_obio := io;
  readout_oohb10 := ohb10a;
  readout_ozea := zea;

  // Reactions:
  bcar_cleave: bcar => b10 + io; kb1_multiplier*kb1*bcar;
  b10_cleave: b10 => io + c14dial; kb2_multiplier*kcleave*b10;
  bcry_minor: bcry => b10 + ohio; kc1_multiplier*kc1*bcry;
  ohb10_cleave: ohb10a => ohio + c14dial; kc4_multiplier*kcleave*ohb10a
    ;
  bcry_major: bcry => ohb10a + io; kc2_multiplier*kcleave*bcry;
  zea_cleave: zea => ohb10a + ohio; k5_multiplier*k5*zea;

  // Species initializations:
  bcar = init_bcar;
  bcry = init_bcry;
  zea = init_zea;
  b10 = init_b10;
  ohb10a = init_ohb10;
  io = 0;
  ohio = 0;
  c14dial = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  init_bcar = 4.44;
  init_bcry = 5.22;
  init_zea = 2.96;
  init_b10 = 4.22;
  init_ohb10 = 1.74;
  k5_multiplier = 0;
  kb1_multiplier = 0;
```

```
  kb2_multiplier = 0;
  kc1_multiplier = 0;
  kc2_multiplier = 0;
  kc4_multiplier = 0;
  kb1 = 0.015;
  kc1 = 0.002;
  k5 = 0.0035;
  kcleave = ;

  // Other declarations:
  var readout_ob10, readout_obcar, readout_obcry, readout_obio,
    readout_oohb10;
  var readout_ozea;
  const cell, init_bcar, init_bcry, init_zea, init_b10, init_ohb10,
    k5_multiplier;
  const kb1_multiplier, kb2_multiplier, kc1_multiplier, kc2_multiplier,
     kc4_multiplier;
  const kb1, kc1, k5, kcleave;
end
```

Listing 68: Archived Antimony for `bruno_5`; task `56836848_24`; selected version 2

```
// Created by libAntimony v3.1.3
model *ccd_assay()

  // Compartments and Species:
  compartment assay;
  species bcar in assay, bcry in assay, zea in assay, b10 in assay,
    ohb10 in assay;
  species bion in assay, ohbion in assay;

  // Assignment Rules:
  readout_ob10 := b10;
  readout_obcar := bcar;
  readout_obcry := bcry;
  readout_oohb10 := ohb10;
  readout_obio := bion;
  readout_ozea := zea;

  // Reactions:
  bcar_cleavage: bcar -> b10 + bion; k_bcar*kb1_multiplier*bcar;
  bcry_to_ohb10: bcry -> ohb10 + bion; k_ccd910*kc2_multiplier*bcry;
  bcry_to_b10: bcry -> b10 + ohbion; k_bcry_apo*kc1_multiplier*bcry;
  zea_cleavage: zea -> ohb10 + ohbion; k_zea*k5_multiplier*zea;
  b10_second_cleavage: b10 -> bion; k_ccd910*kb2_multiplier*b10;
  ohb10_second_cleavage: ohb10 -> ohbion; k_ccd910*kc4_multiplier*ohb10
    ;

  // Species initializations:
  bcar = init_bcar;
  bcry = init_bcry;
  zea = init_zea;
  b10 = init_b10;
  ohb10 = init_ohb10;
  bion = 0;
  ohbion = 0;

  // Compartment initializations:
  assay = 1;

  // Variable initializations:
  init_bcar = 0;
  init_bcry = 0;
  init_zea = 0;
  init_b10 = 0;
```

```
  init_ohb10 = 0;
  k5_multiplier = 0;
  kb1_multiplier = 0;
  kb2_multiplier = 0;
  kc1_multiplier = 0;
  kc2_multiplier = 0;
  kc4_multiplier = 0;
  szea = 0.5;
  k_bcar = 0.017;
  k_bcry_apo = 0.003;
  k_zea = 0.003;
  k_ccd910 = ;

  // Other declarations:
  var readout_ob10, readout_obcar, readout_obcry, readout_oohb10,
    readout_obio;
  var readout_ozea;
  const assay, init_bcar, init_bcry, init_zea, init_b10, init_ohb10,
    k5_multiplier;
  const kb1_multiplier, kb2_multiplier, kc1_multiplier, kc2_multiplier,
     kc4_multiplier;
  const szea, k_bcar, k_bcry_apo, k_zea, k_ccd910;
end
```

H.9 OLIVEIRA

Listing 69: Archived Antimony for `oliveira_1`; task `57351389_10`; selected version 12

```
// Created by libAntimony v3.1.3
model *covid_interior()

  // Compartments and Species:
  compartment region;
  species S in region, E in region, A in region, I in region, H in
    region;
  species U in region, R in region, C_cases in region, C_deaths in
    region;

  // Assignment Rules:
  beta_t := alpha_hu + beta_2_multiplier*beta_high + (beta_high -
    alpha_hu)*piecewise(1, time < t_int, 0);
  readout_cumulative_cases := population*C_cases;
  readout_cumulative_deaths := population*rho_conf*C_deaths;

  // Reactions:
  infection: S -> E; beta_t*(I + A)*S;
  latent_symptomatic: E -> I; kappa*p_symp_rate*E;
  latent_asymptomatic: E -> A; kappa*(1 - p_symp_rate)*E;
  asymptomatic_recovery: A -> R; gamma_a*A;
  symptomatic_recovery: I -> R; gamma_s*(1 - xi)*I;
  symptomatic_hospitalization: I -> H; gamma_s*xi*I;
  ward_to_icu: H -> U; alpha_hu*H;
  ward_death: H -> ; mu_h*H;
  ward_recovery: H -> R; omega_h*H;
  icu_death: U -> ; mu_u*U;
  icu_recovery: U -> R; omega_u*U;
  case_accumulation:  -> C_cases; gamma_s*xi*I;
  death_accumulation:  -> C_deaths; mu_h*H + mu_u*U;

  // Species initializations:
  S = S_init;
  E = 0;
  A = asymptomatic_init_concentration;
  I = 0;
```

```
  H = 0;
  U = 0;
  R = 0;
  C_cases = 0;
  C_deaths = 0;

  // Compartment initializations:
  region = 1;

  // Variable initializations:
  S_init = 1;
  asymptomatic_init_concentration = 6.9e-07;
  kappa = 0.25;
  p_symp_rate = 0.2;
  gamma_a = 0.285714;
  gamma_s = 0.25;
  xi = 0.53;
  mu_h = 0.15;
  mu_u = 0.4;
  omega_h = 0.14;
  omega_u = 0.29;
  population = 12098867;
  t_int = 20;
  alpha_hu = 0.1;
  rho_conf = 0.65;
  beta_2_multiplier = 0;
  beta_high = ;

  // Other declarations:
  var beta_t, readout_cumulative_cases, readout_cumulative_deaths;
  const region, S_init, asymptomatic_init_concentration, kappa,
    p_symp_rate;
  const gamma_a, gamma_s, xi, mu_h, mu_u, omega_h, omega_u, population,
    t_int;
  const alpha_hu, rho_conf, beta_2_multiplier, beta_high;
end
```

Listing 70: Archived Antimony for `oliveira_2`; task `57351389_11`; selected version 66

```
// Created by libAntimony v3.1.3
model *covid_interior()

  // Compartments and Species:
  species S, E, A, I, I_h, H, U, R, C, D, W_D, W_D3, W_D4;

  // Assignment Rules:
  h_hosp := 1;
  nu := rho_d1;
  rho_d1 := beta_min;
  t_sev := t_step;
  readout_cumulative_cases := C;
  readout_cumulative_deaths := D;

  // Reactions:
  infection: S -> E; (beta_min + beta_1*exp(-k_dec*time) + step_gain*
    exp(-k_dec*(time - t_step))*piecewise(time > t_step, 1, 0))*
    piecewise(time < 60, 1, beta_2_multiplier)*(A + I)*S/population;
  e_to_a: E -> A; (1 - p_symp_rate)*kappa*E;
  e_to_i: E -> I; p_symp_rate*kappa*E;
  a_to_r: A -> R; gamma_a*A;
  i_to_ih: I -> I_h; h_hosp*gamma_s*I;
  ih_to_h: I_h -> H; (1 - xi)*nu*I_h;
  ih_to_u: I_h -> U; xi*nu*I_h;
  h_to_r: H -> R; omega_h*H;
  u_to_r: U -> R; omega_u*U;
```

```
  h_to_wd: H -> W_D; (1 - sev_drop*piecewise(time > t_sev, 1, 0))*mu_h*
    H;
  u_to_wd: U -> W_D; (1 - sev_drop*piecewise(time > t_sev, 1, 0))*mu_u*
    U;
  case_acc:  -> C; p_symp_rate*kappa*E;
  wd_to_wd3: W_D -> W_D3; rho_stage*W_D;
  wd3_to_wd4: W_D3 -> W_D4; rho_stage*W_D3;
  wd4_to_d: W_D4 -> D; rho_stage*W_D4;

  // Species initializations:
  S = population - 3;
  E = E_init;
  A = A_init;
  I = I_init;
  I_h = 0;
  H = 0;
  U = 0;
  R = 0;
  C = 0;
  D = 0;
  W_D = 0;
  W_D3 = ;
  W_D4 = ;

  // Variable initializations:
  population = ;
  E_init = 1;
  A_init = 1;
  I_init = 1;
  beta_1 = 1.5;
  k_dec = 0.0937;
  beta_min = 0.311;
  beta_2_multiplier = 0.1;
  t_step = ;
  step_gain = ;
  p_symp_rate = ;
  kappa = ;
  gamma_a = ;
  gamma_s = ;
  xi = ;
  omega_h = ;
  omega_u = ;
  sev_drop = ;
  mu_h = ;
  mu_u = ;
  rho_stage = ;

  // Other declarations:
  var h_hosp, nu, rho_d1, t_sev, readout_cumulative_cases,
    readout_cumulative_deaths;
  const population, E_init, A_init, I_init, beta_1, k_dec, beta_min,
    beta_2_multiplier;
  const t_step, step_gain, p_symp_rate, kappa, gamma_a, gamma_s, xi,
    omega_h;
  const omega_u, sev_drop, mu_h, mu_u, rho_stage;
end
```

Listing 71: Archived Antimony for `oliveira_3`; task `57351389_12`; selected version 14

```
// Created by libAntimony v3.1.3
model *covid_state()

  // Compartments and Species:
  compartment world;
  species S in world, E in world, A in world, I in world, H in world, U
     in world;
  species R in world, D in world, D_p in world, cum_cases in world,
    cum_deaths in world;
  species D_pp;

  // Assignment Rules:
  w_asc := k_rep;
  u_t := 0.5*(1 - tanh((time - t_int)/0.25));
  beta_t := beta_r + beta_1*(beta_2_multiplier + (1 - beta_2_multiplier
    )*u_t);
  asc := 1 + (c_asc - 1)*(1 - 1/(1 + exp((time - t_asc)/w_asc)));
  readout_cumulative_cases := cum_cases;
  readout_cumulative_deaths := scale_d*cum_deaths;

  // Reactions:
  infection: S -> E; beta_t*S*(A + I)/population;
  e_to_a: E -> A; kappa*(1 - p_symp_rate)*E;
  e_to_i: E -> I; kappa*p_symp_rate*E;
  a_to_r: A -> R; gamma_a*A;
  i_to_h: I -> H; xi*gamma_s*I;
  i_to_r: I -> R; (1 - xi)*gamma_s*I;
  h_to_u: H -> U; omega_h*H;
  h_to_d: H -> D; mu_h*H;
  u_to_r: U -> R; omega_u*U;
  u_to_d: U -> D; mu_u*U;
  d_report: D -> D_p; k_rep*D;
  case_onset:  -> cum_cases; kappa*p_symp_rate*E*asc;
  d_p2: D_p -> D_pp; k_rep*D_p;
  d_confirm2: D_pp -> cum_deaths; k_rep*D_pp;

  // Species initializations:
  S = population*(1 - asymptomatic_init_concentration);
  E = 0;
  A = asymptomatic_init_concentration*population;
  I = 0;
  H = 0;
  U = 0;
  R = 0;
  D = 0;
  D_p = 0;
  cum_cases = 0;
  cum_deaths = 0;
  D_pp = ;

  // Compartment initializations:
  world = 1;

  // Variable initializations:
  population = 12098867;
  asymptomatic_init_concentration = 4.11e-07;
  beta_1 = 0.4227;
  beta_r = 0.2934;
  t_int = 29.53;
  k_rep = 0.8567;
  scale_d = 0.2048;
  c_asc = 2.5169;
  t_asc = 43.72;
```

```
  beta_2_multiplier = 0;
  kappa = 0.25;
  p_symp_rate = 0.2;
  gamma_a = 0.28571429;
  gamma_s = 0.25;
  xi = 0.53;
  mu_h = 0.15;
  mu_u = 0.4;
  omega_h = 0.14;
  omega_u = 0.29;

  // Other declarations:
  var w_asc, u_t, beta_t, asc, readout_cumulative_cases,
    readout_cumulative_deaths;
  const world, population, asymptomatic_init_concentration, beta_1,
    beta_r;
  const t_int, k_rep, scale_d, c_asc, t_asc, beta_2_multiplier, kappa,
    p_symp_rate;
  const gamma_a, gamma_s, xi, mu_h, mu_u, omega_h, omega_u;
end
```

Listing 72: Archived Antimony for `oliveira_4`; task `57351389_13`; selected version 8

```
// Created by libAntimony v3.1.3
model *covid_pa()

  // Compartments and Species:
  compartment world;
  species S in world, E in world, A in world, I in world, H in world, U
     in world;
  species R in world, D in world, C_cum in world, C_rep, D_rep;

  // Assignment Rules:
  Sgm := 1/(1 + exp(-(time - t_int)/w_sw));
  foi := beta_1*(1 - f_int*Sgm)*(1 - f2_eff*Sgm2)*(A + I)/population;
  f2_eff := f_int + beta_2_multiplier*(1 - f_int);
  Sgm2 := 1/(1 + exp(-(time - t2_int)/w2_sw));
  Sgm6 := exp(-(time - t6_int)^2/(2*w6_sw^2));
  rho_t := rho_deaths*Sgm3*(1 + h_up*Sgm5)*(1 - d_d*Sgm4);
  Sgm3 := 1/(1 + exp(-(time - t3_int)/w3_sw));
  Sgm5 := 1/(1 + exp(-(time - t5_int)/w5_sw));
  Sgm4 := 1/(1 + exp(-(time - t4_int)/w2_sw));
  readout_cumulative_cases := C_rep;
  readout_cumulative_deaths := D_rep;

  // Reactions:
  infection: S -> E; foi*S;
  e_to_a: E -> A; (1 - p_symp_rate)*kappa*E;
  e_to_i: E -> I; p_symp_rate*kappa*E;
  a_to_r: A -> R; gamma_a*A;
  i_to_h: I -> H; xi*gamma_s*I;
  i_to_r: I -> R; (1 - xi)*gamma_s*I;
  h_to_u: H -> U; omega_h*H;
  h_to_r: H -> R; mu_h*H;
  u_to_d: U -> D; omega_u*U;
  u_to_r: U -> R; mu_u*U;
  case_onset:  -> C_cum; p_symp_rate*kappa*E;
  c_report:  -> C_rep; (1 - c_dip*Sgm6)*p_symp_rate*kappa*E;
  d_report:  -> D_rep; rho_t*omega_u*U;

  // Species initializations:
  S = population*(1 - symptomatic_init_concentration);
  E = 0;
  A = 0;
  I = population*symptomatic_init_concentration;
```

```
  H = 0;
  U = 0;
  R = 0;
  D = 0;
  C_cum = 0;
  C_rep = ;
  D_rep = ;

  // Compartment initializations:
  world = 1;

  // Variable initializations:
  population = ;
  symptomatic_init_concentration = ;
  beta_2_multiplier = 0;
  beta_1 = 0.8;
  f_int = 0.5;
  t_int = 27;
  w_sw = 6;
  t2_int = ;
  w2_sw = ;
  t6_int = ;
  w6_sw = ;
  rho_deaths = ;
  h_up = ;
  d_d = ;
  t3_int = ;
  w3_sw = ;
  t4_int = ;
  t5_int = ;
  w5_sw = ;
  p_symp_rate = ;
  kappa = ;
  gamma_a = ;
  xi = ;
  gamma_s = ;
  omega_h = ;
  mu_h = ;
  omega_u = ;
  mu_u = ;
  c_dip = ;

  // Other declarations:
  var Sgm, foi, f2_eff, Sgm2, Sgm6, rho_t, Sgm3, Sgm5, Sgm4,
    readout_cumulative_cases;
  var readout_cumulative_deaths;
  const world, population, symptomatic_init_concentration,
    beta_2_multiplier;
  const beta_1, f_int, t_int, w_sw, t2_int, w2_sw, t6_int, w6_sw,
    rho_deaths;
  const h_up, d_d, t3_int, w3_sw, t4_int, t5_int, w5_sw, p_symp_rate,
    kappa;
  const gamma_a, xi, gamma_s, omega_h, mu_h, omega_u, mu_u, c_dip;
end
```

Listing 73: Archived Antimony for `oliveira_5`; task `57351389_14`; selected version 10

```
// Created by libAntimony v3.1.3
model *covid_wave_interior()

  // Compartments and Species:
  compartment region;
  species S in region, E in region, A in region, I in region, H in
    region;
  species U in region, R in region, D in region, C in region;

  // Assignment Rules:
  asymptomatic_init_concentration := 0;
  symptomatic_init_concentration := asymptomatic_init_concentration;
  switch_width := t_int;
  intervention_switch := 0.5*(1 - tanh((time - t_int)/switch_width));
  readout_cumulative_cases := rep_min*C + (1 - rep_min)*C/(1 + exp(-(
    time - t_asc)/w_asc))^nu;
  w_asc := t_int;
  readout_cumulative_deaths := D;
  K_detect := rep_min;

  // Reactions:
  infection: S -> E + C; (beta_a*A + beta_s*(beta_2_multiplier + (1 -
    beta_2_multiplier)*intervention_switch)*I)*S/population;
  incubation: E -> A; kappa*E;
  onset_symptomatic: A -> I; p_symp_rate*gamma_a*A;
  recovery_asymptomatic: A -> R; (1 - p_symp_rate)*gamma_a*A;
  hospitalization: I -> H; xi*gamma_s*I;
  recovery_symptomatic: I -> R; (1 - xi)*gamma_s*I;
  icu_admission: H -> U; mu_h*H;
  discharge_hospital: H -> R; omega_h*H;
  death_icu: U -> D; mu_u*U;
  stepdown_icu: U -> R; omega_u*U;

  // Species initializations:
  S = population*(1 - exposed_init_concentration -
    asymptomatic_init_concentration - symptomatic_init_concentration);
  E = population*exposed_init_concentration;
  A = 0;
  I = 0;
  H = 0;
  U = 0;
  R = 0;
  D = 0;
  C = 0;

  // Compartment initializations:
  region = 1;

  // Variable initializations:
  population = ;
  exposed_init_concentration = ;
  beta_a = 3.2;
  beta_s = 13;
  t_int = 30;
  beta_2_multiplier = 0;
  rep_min = ;
  t_asc = ;
  nu = ;
  kappa = ;
  p_symp_rate = ;
  gamma_a = ;
  xi = ;
  gamma_s = ;
```

```
  mu_h = ;
  omega_h = ;
  mu_u = ;
  omega_u = ;

  // Other declarations:
  var asymptomatic_init_concentration, symptomatic_init_concentration,
    switch_width;
  var intervention_switch, readout_cumulative_cases, w_asc,
    readout_cumulative_deaths;
  var K_detect;
  const region, population, exposed_init_concentration, beta_a, beta_s,
     t_int;
  const beta_2_multiplier, rep_min, t_asc, nu, kappa, p_symp_rate,
    gamma_a;
  const xi, gamma_s, mu_h, omega_h, mu_u, omega_u;
end
```

H.10 SNEYD

Listing 74: Archived Antimony for `sneyd_1`; task `57351389_0`; selected version 21

```
// Created by libAntimony v3.1.3
model *ip3r_gating()

  // Compartments and Species:
  compartment cell;
  species IP3 in cell, Ca in cell, xu in cell, xb in cell, iu in cell,
    ib in cell;
  species du in cell, db in cell, lu in cell, lb in cell, O in cell, b2
    ;

  // Assignment Rules:
  readout_open_probability := O;

  // Reactions:
  ip3_bind: xu -> xb; kx*IP3^n_ip3*xu;
  ip3_unbind: xb -> xu; kxr*xb;
  ca_block_bind: iu -> ib; (kii*Ca^n_inh/(K_ii^n_inh + Ca^n_inh))*iu;
  ca_block_unbind: ib -> iu; K_ii*ib;
  desens: du -> db; kd*xb^n_d*du;
  resens: db -> du; krc*Ca^n_d*db;
  lock_bind: lu -> lb; kop*Ca^n_g*db*lu;
  lock_unbind: lb -> lu; K_ii*lb;
  gate_open:  -> O; kop*b2*iu*du*lu*(1 - O);
  gate_close: O -> ; kxr*O;
  b2_form:  -> b2; kii*xb^n_g*(1 - b2);
  b2_loss: b2 -> ; kxr*b2;

  // Species initializations:
  IP3 = 10;
  Ca = 0.4;
  xu = xu_init;
  xb = 0;
  iu = iu_init;
  ib = 0;
  du = du_init;
  db = 0;
  lu = lu_init;
  lb = 0;
  O = 0;
  b2 = ;

  // Compartment initializations:
```

```
  cell = 1;

  // Variable initializations:
  xu_init = 1;
  iu_init = 1;
  du_init = 1;
  lu_init = 1;
  kx = 0.03;
  kxr = 9;
  n_ip3 = 3.9;
  kd = 11;
  n_d = 1;
  krc = 0.1;
  kop = 24;
  n_g = 1;
  K_ii = 25;
  kii = 40;
  n_inh = 2;

  // Other declarations:
  var readout_open_probability;
  const cell, xu_init, iu_init, du_init, lu_init, kx, kxr, n_ip3, kd,
    n_d;
  const krc, kop, n_g, K_ii, kii, n_inh;
end
```

Listing 75: Archived Antimony for `sneyd_2`; task `57351389_1`; selected version 10

```
// Created by libAntimony v3.1.3
model *IP3R_gating()

  // Compartments and Species:
  compartment cell;
  species IP3 in cell, Ca in cell, C in cell, O in cell, I in cell;

  // Assignment Rules:
  readout_open_probability := O;

  // Reactions:
  channel_activation: C -> O; k_gate*(IP3^n_ip3/(K_ip3^n_ip3 + IP3^
    n_ip3))*(Ca^n_caa/(K_ca_shared^n_caa + Ca^n_caa))*C;
  channel_deactivation: O -> C; k_fast*O;
  channel_recovery: I -> C; k_rec*I;
  channel_refractory: C -> I; k_fast*(IP3^n_ip3/(K_ip3^n_ip3 + IP3^
    n_ip3))*C;
  channel_cablock: O -> C; k_gate*(Ca^n_ci/(K_ca_shared^n_ci + Ca^n_ci)
    )*O;

  // Species initializations:
  IP3 = 0.1;
  Ca = 0.1;
  C = C_init;
  O = 0;
  I = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  C_init = 1;
  k_gate = 50;
  k_fast = 25;
  k_rec = 0.2;
  K_ip3 = 4.5;
  n_ip3 = 2.5;
```

```
  K_ca_shared = 1;
  n_caa = 1.5;
  n_ci = 1.5;

  // Other declarations:
  var readout_open_probability;
  const cell, C_init, k_gate, k_fast, k_rec, K_ip3, n_ip3, K_ca_shared,
     n_caa;
  const n_ci;
end
```

Listing 76: Archived Antimony for `sneyd_3`; task `57351389_2`; selected version 7

```
// Created by libAntimony v3.1.3
model *IP3R_gating()

  // Compartments and Species:
  compartment cell;
  species R in cell, O in cell, I in cell, IP3 in cell, Ca in cell;

  // Assignment Rules:
  BI := IP3^hI/(KI^hI + IP3^hI);
  Ba := Ca^hA/(Ka^hA + Ca^hA);
  Bi := Ca^hH/(Ki^hH + Ca^hH);
  readout_open_probability := O/(R + O + I);

  // Reactions:
  channel_opening: R => O; a*BI*Ba*R;
  ca_inactivation: O => I; c_inact*Bi*O;
  recovery: I => R; k_rec*I;

  // Species initializations:
  R = R_init;
  O = 0;
  I = 0;
  IP3 = 1;
  Ca = 1;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 1;
  hI = 2;
  KI = 4;
  hA = 1.5;
  Ka = 0.2;
  hH = 1.5;
  Ki = 1;
  a = 10;
  c_inact = 1;
  k_rec = 0.3;

  // Other declarations:
  var BI, Ba, Bi, readout_open_probability;
  const cell, R_init, hI, KI, hA, Ka, hH, Ki, a, c_inact, k_rec;
end
```

Listing 77: Archived Antimony for `sneyd_4`; task `57351389_3`; selected version 6

```
// Created by libAntimony v3.1.3
model *IP3R_gating()

  // Compartments and Species:
  compartment cell;
  species IP3 in cell, Ca in cell, R in cell, A in cell, I in cell;

  // Assignment Rules:
  readout_open_probability := A;

  // Reactions:
  channel_activation: R -> A; k_act*(IP3^n_IP3/(K_IP3^n_IP3 + IP3^n_IP3
    ))*(Ca^p_act/(K_actCa^p_act + Ca^p_act))*(K_inhCa^m_inh/(K_inhCa^
    m_inh + Ca^m_inh))*R;
  channel_deactivation: A -> R; k_deact*A;
  channel_inactivation: A -> I; k_inact*(Ca/(K_CaInact + Ca))*A;
  channel_recovery: I -> R; k_recov*I;

  // Species initializations:
  IP3 = 10;
  Ca = 0.4;
  R = R_init;
  A = 0;
  I = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  R_init = 1;
  k_act = 20;
  K_IP3 = 3.7;
  n_IP3 = 3;
  K_actCa = 0.2;
  k_deact = 45;
  k_inact = 2;
  K_inhCa = 3;
  k_recov = 0.1;
  p_act = ;
  m_inh = ;
  K_CaInact = ;

  // Other declarations:
  var readout_open_probability;
  const cell, R_init, k_act, K_IP3, n_IP3, K_actCa, k_deact, k_inact,
    K_inhCa;
  const k_recov, p_act, m_inh, K_CaInact;
end
```

Listing 78: Archived Antimony for `sneyd_5`; task `57351389_4`; selected version 19

```
// Created by libAntimony v3.1.3
model *ip3r_gating()

  // Compartments and Species:
  compartment cell;
  species IP3 in cell, Ca in cell, gopen in cell, gavail in cell,
    gprimed;

  // Assignment Rules:
  fip3 := IP3^hI/(Kip3^hI + IP3^hI);
  fact := Ca^nact/(Kact^nact + Ca^nact);
  uinh := Ca^ninh/(Kinh^ninh + Ca^ninh);
```

```
  readout_open_probability := 4*(act*gopen*gavail)^3 - 3*(act*gopen*
    gavail)^4;

  // Reactions:
  J_close: gopen -> ; (kclA*(1 - fact) + kclI*uinh)*gopen;
  J_inact: gavail -> ; kin_max*uinh*gprimed^hIi*gavail;
  J_recover:  -> gavail; krec*(1 - gavail);
  J_prime:  -> gprimed; kpr*fip3*(1 - gprimed - gopen);
  J_gate: gprimed -> gopen; kpr*gprimed;

  // Species initializations:
  IP3 = 0.1;
  Ca = 0.1;
  gopen = 0;
  gavail = gavail_init;
  gprimed = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  gavail_init = 1;
  hI = 2.5;
  Kip3 = 4;
  nact = 2;
  Kact = 0.2;
  ninh = 2;
  Kinh = 0.6;
  kin_max = 15;
  krec = 0.5;
  kpr = 30;
  act = ;
  kclA = ;
  kclI = ;
  hIi = ;

  // Other declarations:
  var fip3, fact, uinh, readout_open_probability;
  const cell, gavail_init, hI, Kip3, nact, Kact, ninh, Kinh, kin_max,
    krec;
  const kpr, act, kclA, kclI, hIi;
end
```

H.11 BLASI

Listing 79: Archived Antimony for `blasi_1`; task `56836848_5`; selected version 16

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cell;
species s0 in cell, s5 in cell, s8 in cell, s12 in cell, s16 in cell,
    s58 in cell;
species s5_12 in cell, s5_16 in cell, s8_12 in cell, s8_16 in cell,
    s12_16 in cell;
species s5812 in cell, s5816 in cell, s5_12_16 in cell, s8_12_16 in
    cell;
species s581216 in cell;

// Assignment Rules:
readout_observable_0ac := s0;
readout_observable_k5 := s5;
readout_observable_k8 := s8;
readout_observable_k12 := s12;
readout_observable_k16 := s16;
```

```
readout_observable_k5k8 := s58;
readout_observable_k5k12 := s5_12;
readout_observable_k5k16 := s5_16;
readout_observable_k8k12 := s8_12;
readout_observable_k8k16 := s8_16;
readout_observable_k12k16 := s12_16;
readout_observable_k5k8k12 := s5812;
readout_observable_k5k8k16 := s5816;
readout_observable_k5k12k16 := s5_12_16;
readout_observable_k8k12k16 := s8_12_16;
readout_observable_4ac := s581216;

// Reactions:
a_s0_s8: s0 -> s8; k8*s0;
a_s0_s12: s0 -> s12; k12*s0;
a_s0_s16: s0 -> s16; k12*s0;
a_s5_s58: s5 -> s58; k8*c58*s5;
a_s5_s5_12: s5 -> s5_12; k12*c2*s5;
a_s5_s5_16: s5 -> s5_16; k12*c5_16*s5;
a_s8_s8_12: s8 -> s8_12; k12*c2*s8;
a_s8_s8_16: s8 -> s8_16; k12*c2*s8;
a_s12_s8_12: s12 -> s8_12; k8*c2*s12;
a_s12_s12_16: s12 -> s12_16; k12*c2*s12;
a_s16_s8_16: s16 -> s8_16; k8*c2*s16;
a_s16_s12_16: s16 -> s12_16; k12*c2*s16;
a_s58_s5812: s58 -> s5812; k12*c2*c2*t_5812*s58;
a_s58_s5816: s58 -> s5816; k12*c5_16*c2*s58;
a_s5_12_s5812: s5_12 -> s5812; k8*c58*c2*t_5812*s5_12;
a_s5_12_s5_12_16: s5_12 -> s5_12_16; k12*c5_16*c2*s5_12;
a_s5_16_s5816: s5_16 -> s5816; k8*c58*c2*s5_16;
a_s5_16_s5_12_16: s5_16 -> s5_12_16; k12*c2*c2*s5_16;
a_s8_12_s8_12_16: s8_12 -> s8_12_16; k12*c2*c2*t_81216*s8_12;
a_s8_16_s8_12_16: s8_16 -> s8_12_16; k12*c2*c2*t_81216*s8_16;
a_s12_16_s8_12_16: s12_16 -> s8_12_16; k8*c2*c2*t_81216*s12_16;
a_s5812_s581216: s5812 -> s581216; k12*c5_16*c2*c2*t_81216*t_81216*
    s5812;
a_s5816_s581216: s5816 -> s581216; k12*c2*c2*c2*c2*t_5812*t_81216*t_81216*
    s5816;
a_s5_12_16_s581216: s5_12_16 -> s581216; k8*c58*c2*c2*t_5812*t_81216*
    t_81216*s5_12_16;
r_s5_s0: s5 -> s0; d*s5;
r_s8_s0: s8 -> s0; d*s8;
r_s12_s0: s12 -> s0; d*s12;
r_s16_s0: s16 -> s0; d*s16;
r_s58_s5: s58 -> s5; d*s58;
r_s58_s8: s58 -> s8; d*s58;
r_s5_12_s5: s5_12 -> s5; d*s5_12;
r_s5_12_s12: s5_12 -> s12; d*s5_12;
r_s5_16_s5: s5_16 -> s5; d*s5_16;
r_s5_16_s16: s5_16 -> s16; d*s5_16;
r_s8_12_s8: s8_12 -> s8; d*s8_12;
r_s8_12_s12: s8_12 -> s12; d*s8_12;
r_s8_16_s8: s8_16 -> s8; d*s8_16;
r_s8_16_s16: s8_16 -> s16; d*s8_16;
r_s12_16_s12: s12_16 -> s12; d*s12_16;
r_s12_16_s16: s12_16 -> s16; d*s12_16;
r_s5812_s58: s5812 -> s58; d*s5812;
r_s5812_s5_12: s5812 -> s5_12; d*s5812;
r_s5812_s8_12: s5812 -> s8_12; d*s5812;
r_s5816_s58: s5816 -> s58; d*s5816;
r_s5816_s5_16: s5816 -> s5_16; d*s5816;
r_s5816_s8_16: s5816 -> s8_16; d*s5816;
r_s5_12_16_s5_12: s5_12_16 -> s5_12; d*s5_12_16;
r_s5_12_16_s5_16: s5_12_16 -> s5_16; d*s5_12_16;
r_s5_12_16_s12_16: s5_12_16 -> s12_16; d*s5_12_16;
```

```
r_s8_12_16_s8_12: s8_12_16 -> s8_12; d*s8_12_16;
r_s8_12_16_s8_16: s8_12_16 -> s8_16; d*s8_12_16;
r_s8_12_16_s12_16: s8_12_16 -> s12_16; d*s8_12_16;
r_s581216_s5812: s581216 -> s5812; d*s581216;
r_s581216_s5816: s581216 -> s5816; d*s581216;
r_s581216_s5_12_16: s581216 -> s5_12_16; d*s581216;
r_s581216_s8_12_16: s581216 -> s8_12_16; d*s581216;
hat1_bare: s0 => s5_12; k_h*s0;

// Species initializations:
s0 = s0_init;
s5 = 0;
s8 = 0;
s12 = 0;
s16 = 0;
s58 = 0;
s5_12 = 0;
s5_16 = 0;
s8_12 = 0;
s8_16 = 0;
s12_16 = 0;
s5812 = 0;
s5816 = 0;
s5_12_16 = 0;
s8_12_16 = 0;
s581216 = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
s0_init = 1;
k8 = 0.0314;
k12 = 0.0337;
c58 = 5.5;
c2 = 3.2;
c5_16 = 1;
d = 1;
t_5812 = ;
t_81216 = ;
k_h = ;

// Other declarations:
var readout_observable_0ac, readout_observable_k5,
    readout_observable_k8;
var readout_observable_k12, readout_observable_k16,
    readout_observable_k5k8;
var readout_observable_k5k12, readout_observable_k5k16,
    readout_observable_k8k12;
var readout_observable_k8k16, readout_observable_k12k16,
    readout_observable_k5k8k12;
var readout_observable_k5k8k16, readout_observable_k5k12k16,
    readout_observable_k8k12k16;
var readout_observable_4ac;
const cell, s0_init, k8, k12, c58, c2, c5_16, d, t_5812, t_81216, k_h;
```

Listing 80: Archived Antimony for `blasi_2`; task `56836848_6`; selected version 10

```
// Created by libAntimony v3.1.3
model *H4_acetylation()

  // Compartments and Species:
  compartment cell;
  species H4_0ac in cell, H4_K5 in cell, H4_K8 in cell, H4_K12 in cell,
     H4_K16 in cell;
  species H4_K5K8 in cell, H4_K5K12 in cell, H4_K5K16 in cell, H4_K8K12
     in cell;
  species H4_K8K16 in cell, H4_K12K16 in cell, H4_K5K8K12 in cell,
    H4_K5K8K16 in cell;
  species H4_K5K12K16 in cell, H4_K8K12K16 in cell, H4_4ac in cell;

  // Assignment Rules:
  readout_observable_0ac := H4_0ac;
  readout_observable_k5 := H4_K5;
  readout_observable_k8 := H4_K8;
  readout_observable_k12 := H4_K12;
  readout_observable_k16 := H4_K16;
  readout_observable_k5k8 := H4_K5K8;
  readout_observable_k5k12 := H4_K5K12;
  readout_observable_k5k16 := H4_K5K16;
  readout_observable_k8k12 := H4_K8K12;
  readout_observable_k8k16 := H4_K8K16;
  readout_observable_k12k16 := H4_K12K16;
  readout_observable_k5k8k12 := H4_K5K8K12;
  readout_observable_k5k8k16 := H4_K5K8K16;
  readout_observable_k5k12k16 := H4_K5K12K16;
  readout_observable_k8k12k16 := H4_K8K12K16;
  readout_observable_4ac := H4_4ac;

  // Reactions:
  ac_0ac_to_K8: H4_0ac -> H4_K8; k8*H4_0ac;
  ac_0ac_to_K12: H4_0ac -> H4_K12; k12*H4_0ac;
  ac_0ac_to_K16: H4_0ac -> H4_K16; k16*H4_0ac;
  ac_K5_to_K5K8: H4_K5 -> H4_K5K8; k8*g58*H4_K5;
  ac_K5_to_K5K12: H4_K5 -> H4_K5K12; k12*g5_12*H4_K5;
  ac_K5_to_K5K16: H4_K5 -> H4_K5K16; k16*g5_12*H4_K5;
  ac_K8_to_K8K12: H4_K8 -> H4_K8K12; k12*g5_12*H4_K8;
  ac_K8_to_K8K16: H4_K8 -> H4_K8K16; k16*g8_16*H4_K8;
  ac_K12_to_K8K12: H4_K12 -> H4_K8K12; k8*g5_12*H4_K12;
  ac_K12_to_K12K16: H4_K12 -> H4_K12K16; k16*g5_12*H4_K12;
  ac_K16_to_K8K16: H4_K16 -> H4_K8K16; k8*g8_16*H4_K16;
  ac_K16_to_K12K16: H4_K16 -> H4_K12K16; k12*g5_12*H4_K16;
  ac_K5K8_to_K5K8K12: H4_K5K8 -> H4_K5K8K12; k12*g5_12*g5_12*H4_K5K8;
  ac_K5K8_to_K5K8K16: H4_K5K8 -> H4_K5K8K16; k16*g8_16*g5_12*H4_K5K8;
  ac_K5K12_to_K5K8K12: H4_K5K12 -> H4_K5K8K12; k8*g58*g5_12*H4_K5K12;
  ac_K5K12_to_K5K12K16: H4_K5K12 -> H4_K5K12K16; k16*g5_12*g5_12*
    H4_K5K12;
  ac_K5K16_to_K5K8K16: H4_K5K16 -> H4_K5K8K16; k8*g58*g8_16*H4_K5K16;
  ac_K5K16_to_K5K12K16: H4_K5K16 -> H4_K5K12K16; k12*g5_12*g58*H4_K5K16
    ;
  ac_K8K12_to_K8K12K16: H4_K8K12 -> H4_K8K12K16; k16*g8_16*g5_12*
    H4_K8K12;
  ac_K8K16_to_K8K12K16: H4_K8K16 -> H4_K8K12K16; k12*g5_12*g58*H4_K8K16
    ;
  ac_K12K16_to_K8K12K16: H4_K12K16 -> H4_K8K12K16; k8*g5_12*g8_16*
    H4_K12K16;
  ac_K5K8K12_to_4ac: H4_K5K8K12 -> H4_4ac; k16*g8_16*g8_16*g5_12*
    H4_K5K8K12;
  ac_K5K8K16_to_4ac: H4_K5K8K16 -> H4_4ac; k12*g5_12*g5_12*g58*
    H4_K5K8K16;
```

```
ac_K5K12K16_to_4ac: H4_K5K12K16 -> H4_4ac; k8*g58*g5_12*g8_16*
  H4_K5K12K16;
de_K5_to_0ac: H4_K5 -> H4_0ac; d*H4_K5;
de_K8_to_0ac: H4_K8 -> H4_0ac; d*H4_K8;
de_K12_to_0ac: H4_K12 -> H4_0ac; d*H4_K12;
de_K16_to_0ac: H4_K16 -> H4_0ac; d*H4_K16;
de_K5K8_to_K5: H4_K5K8 -> H4_K5; d*H4_K5K8;
de_K5K8_to_K8: H4_K5K8 -> H4_K8; d*H4_K5K8;
de_K5K12_to_K5: H4_K5K12 -> H4_K5; d*H4_K5K12;
de_K5K12_to_K12: H4_K5K12 -> H4_K12; d*H4_K5K12;
de_K5K16_to_K5: H4_K5K16 -> H4_K5; d*H4_K5K16;
de_K5K16_to_K16: H4_K5K16 -> H4_K16; d*H4_K5K16;
de_K8K12_to_K8: H4_K8K12 -> H4_K8; d*H4_K8K12;
de_K8K12_to_K12: H4_K8K12 -> H4_K12; d*H4_K8K12;
de_K8K16_to_K8: H4_K8K16 -> H4_K8; d*H4_K8K16;
de_K8K16_to_K16: H4_K8K16 -> H4_K16; d*H4_K8K16;
de_K12K16_to_K12: H4_K12K16 -> H4_K12; d*H4_K12K16;
de_K12K16_to_K16: H4_K12K16 -> H4_K16; d*H4_K12K16;
de_K5K8K12_to_K5K8: H4_K5K8K12 -> H4_K5K8; d*H4_K5K8K12;
de_K5K8K12_to_K5K12: H4_K5K8K12 -> H4_K5K12; d*H4_K5K8K12;
de_K5K8K12_to_K8K12: H4_K5K8K12 -> H4_K8K12; d*H4_K5K8K12;
de_K5K8K16_to_K5K8: H4_K5K8K16 -> H4_K5K8; d*H4_K5K8K16;
de_K5K8K16_to_K5K16: H4_K5K8K16 -> H4_K5K16; d*H4_K5K8K16;
de_K5K8K16_to_K8K16: H4_K5K8K16 -> H4_K8K16; d*H4_K5K8K16;
de_K5K12K16_to_K5K12: H4_K5K12K16 -> H4_K5K12; d*H4_K5K12K16;
de_K5K12K16_to_K5K16: H4_K5K12K16 -> H4_K5K16; d*H4_K5K12K16;
de_K5K12K16_to_K12K16: H4_K5K12K16 -> H4_K12K16; d*H4_K5K12K16;
de_K8K12K16_to_K8K12: H4_K8K12K16 -> H4_K8K12; d*H4_K8K12K16;
de_K8K12K16_to_K8K16: H4_K8K12K16 -> H4_K8K16; d*H4_K8K12K16;
de_K8K12K16_to_K12K16: H4_K8K12K16 -> H4_K12K16; d*H4_K8K12K16;
de_4ac_to_K5K8K12: H4_4ac -> H4_K5K8K12; d*H4_4ac;
de_4ac_to_K5K8K16: H4_4ac -> H4_K5K8K16; d*H4_4ac;
de_4ac_to_K5K12K16: H4_4ac -> H4_K5K12K16; d*H4_4ac;
de_4ac_to_K8K12K16: H4_4ac -> H4_K8K12K16; d*H4_4ac;
hat1_diacetyl: H4_0ac -> H4_K5K12; h_hat1*H4_0ac;

// Species initializations:
H4_0ac = H4_0ac_init;
H4_K5 = ;
H4_K8 = ;
H4_K12 = ;
H4_K16 = ;
H4_K5K8 = ;
H4_K5K12 = ;
H4_K5K16 = ;
H4_K8K12 = ;
H4_K8K16 = ;
H4_K12K16 = ;
H4_K5K8K12 = ;
H4_K5K8K16 = ;
H4_K5K12K16 = ;
H4_K8K12K16 = ;
H4_4ac = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
H4_0ac_init = 1;
k8 = 0.04;
k12 = 0.1;
k16 = 0.045;
d = 1;
g58 = ;
g5_12 = ;
```

```
  g8_16 = ;
  h_hat1 = ;

  // Other declarations:
  var readout_observable_0ac, readout_observable_k5,
    readout_observable_k8;
  var readout_observable_k12, readout_observable_k16,
    readout_observable_k5k8;
  var readout_observable_k5k12, readout_observable_k5k16,
    readout_observable_k8k12;
  var readout_observable_k8k16, readout_observable_k12k16,
    readout_observable_k5k8k12;
  var readout_observable_k5k8k16, readout_observable_k5k12k16,
    readout_observable_k8k12k16;
  var readout_observable_4ac;
  const cell, H4_0ac_init, k8, k12, k16, d, g58, g5_12, g8_16, h_hat1;
end
```

Listing 81: Archived Antimony for `blasi_3`; task `56836848_7`; selected version 4

```
// Created by libAntimony v3.1.3
model *H4_acetylation()

  // Compartments and Species:
  compartment cell;
  species H4_0ac in cell, H4_k5 in cell, H4_k8 in cell, H4_k12 in cell,
     H4_k16 in cell;
  species H4_k5k8 in cell, H4_k5k12 in cell, H4_k5k16 in cell, H4_k8k12
     in cell;
  species H4_k8k16 in cell, H4_k12k16 in cell, H4_k5k8k12 in cell,
    H4_k5k8k16 in cell;
  species H4_k5k12k16 in cell, H4_k8k12k16 in cell, H4_4ac in cell;

  // Assignment Rules:
  readout_observable_0ac := H4_0ac;
  readout_observable_k5 := H4_k5;
  readout_observable_k8 := H4_k8;
  readout_observable_k12 := H4_k12;
  readout_observable_k16 := H4_k16;
  readout_observable_k5k8 := H4_k5k8;
  readout_observable_k5k12 := H4_k5k12;
  readout_observable_k5k16 := H4_k5k16;
  readout_observable_k8k12 := H4_k8k12;
  readout_observable_k8k16 := H4_k8k16;
  readout_observable_k12k16 := H4_k12k16;
  readout_observable_k5k8k12 := H4_k5k8k12;
  readout_observable_k5k8k16 := H4_k5k8k16;
  readout_observable_k5k12k16 := H4_k5k12k16;
  readout_observable_k8k12k16 := H4_k8k12k16;
  readout_observable_4ac := H4_4ac;

  // Reactions:
  w5_0: H4_0ac -> H4_k5; k8*H4_0ac;
  w8_0: H4_0ac -> H4_k8; k8*H4_0ac;
  w12_0: H4_0ac -> H4_k12; k12*H4_0ac;
  w16_0: H4_0ac -> H4_k16; k8*H4_0ac;
  w8_5: H4_k5 -> H4_k5k8; k8*rho58*H4_k5;
  w12_5: H4_k5 -> H4_k5k12; k12*rho512*H4_k5;
  w16_5: H4_k5 -> H4_k5k16; k8*rho516*H4_k5;
  w5_8: H4_k8 -> H4_k5k8; k8*rho58*H4_k8;
  w12_8: H4_k8 -> H4_k8k12; k12*rho812*H4_k8;
  w16_8: H4_k8 -> H4_k8k16; k8*rho812*H4_k8;
  w5_12: H4_k12 -> H4_k5k12; k8*rho512*H4_k12;
  w8_12: H4_k12 -> H4_k8k12; k8*rho812*H4_k12;
  w16_12: H4_k12 -> H4_k12k16; k8*rho812*H4_k12;
```

```
w5_16: H4_k16 -> H4_k5k16; k8*rho516*H4_k16;
w8_16: H4_k16 -> H4_k8k16; k8*rho812*H4_k16;
w12_16: H4_k16 -> H4_k12k16; k12*rho812*H4_k16;
w12_58: H4_k5k8 -> H4_k5k8k12; k12*rho512*rho812*sat*H4_k5k8;
w16_58: H4_k5k8 -> H4_k5k8k16; k8*rho516*rho812*H4_k5k8;
w8_512: H4_k5k12 -> H4_k5k8k12; k8*rho58*rho812*sat*H4_k5k12;
w16_512: H4_k5k12 -> H4_k5k12k16; k8*rho516*rho812*sat*H4_k5k12;
w8_516: H4_k5k16 -> H4_k5k8k16; k8*rho58*rho812*H4_k5k16;
w12_516: H4_k5k16 -> H4_k5k12k16; k12*rho512*rho812*sat*H4_k5k16;
w5_812: H4_k8k12 -> H4_k5k8k12; k8*rho58*rho512*sat*H4_k8k12;
w16_812: H4_k8k12 -> H4_k8k12k16; k8*rho812*rho812*H4_k8k12;
w5_816: H4_k8k16 -> H4_k5k8k16; k8*rho58*rho516*H4_k8k16;
w12_816: H4_k8k16 -> H4_k8k12k16; k12*rho812*rho812*H4_k8k16;
w5_1216: H4_k12k16 -> H4_k5k12k16; k8*rho512*rho516*sat*H4_k12k16;
w8_1216: H4_k12k16 -> H4_k8k12k16; k8*rho812*rho812*H4_k12k16;
w16_5812: H4_k5k8k12 -> H4_4ac; k8*rho516*rho812*rho812*H4_k5k8k12;
w12_5816: H4_k5k8k16 -> H4_4ac; k12*rho512*rho812*rho812*sat*
  H4_k5k8k16;
w8_51216: H4_k5k12k16 -> H4_4ac; k8*rho58*rho812*rho812*H4_k5k12k16;
w5_81216: H4_k8k12k16 -> H4_4ac; k8*rho58*rho512*rho516*sat*
  H4_k8k12k16;
d5_5: H4_k5 -> H4_0ac; d*H4_k5;
d8_8: H4_k8 -> H4_0ac; d*H4_k8;
d12_12: H4_k12 -> H4_0ac; d*H4_k12;
d16_16: H4_k16 -> H4_0ac; d*H4_k16;
d8_58: H4_k5k8 -> H4_k5; d*H4_k5k8;
d5_58: H4_k5k8 -> H4_k8; d*H4_k5k8;
d12_512: H4_k5k12 -> H4_k5; d*H4_k5k12;
d5_512: H4_k5k12 -> H4_k12; d*H4_k5k12;
d16_516: H4_k5k16 -> H4_k5; d*H4_k5k16;
d5_516: H4_k5k16 -> H4_k16; d*H4_k5k16;
d12_812: H4_k8k12 -> H4_k8; d*H4_k8k12;
d8_812: H4_k8k12 -> H4_k12; d*H4_k8k12;
d16_816: H4_k8k16 -> H4_k8; d*H4_k8k16;
d8_816: H4_k8k16 -> H4_k16; d*H4_k8k16;
d16_1216: H4_k12k16 -> H4_k12; d*H4_k12k16;
d12_1216: H4_k12k16 -> H4_k16; d*H4_k12k16;
d5_5812: H4_k5k8k12 -> H4_k8k12; d*H4_k5k8k12;
d8_5812: H4_k5k8k12 -> H4_k5k12; d*H4_k5k8k12;
d12_5812: H4_k5k8k12 -> H4_k5k8; d*H4_k5k8k12;
d5_5816: H4_k5k8k16 -> H4_k8k16; d*H4_k5k8k16;
d8_5816: H4_k5k8k16 -> H4_k5k16; d*H4_k5k8k16;
d16_5816: H4_k5k8k16 -> H4_k5k8; d*H4_k5k8k16;
d5_51216: H4_k5k12k16 -> H4_k12k16; d*H4_k5k12k16;
d12_51216: H4_k5k12k16 -> H4_k5k16; d*H4_k5k12k16;
d16_51216: H4_k5k12k16 -> H4_k5k12; d*H4_k5k12k16;
d8_81216: H4_k8k12k16 -> H4_k12k16; d*H4_k8k12k16;
d12_81216: H4_k8k12k16 -> H4_k8k16; d*H4_k8k12k16;
d16_81216: H4_k8k12k16 -> H4_k8k12; d*H4_k8k12k16;
d5_4: H4_4ac -> H4_k8k12k16; d*H4_4ac;
d8_4: H4_4ac -> H4_k5k12k16; d*H4_4ac;
d12_4: H4_4ac -> H4_k5k8k16; d*H4_4ac;
d16_4: H4_4ac -> H4_k5k8k12; d*H4_4ac;

// Species initializations:
H4_0ac = H4_0ac_init;
H4_k5 = 0;
H4_k8 = 0;
H4_k12 = 0;
H4_k16 = 0;
H4_k5k8 = 0;
H4_k5k12 = 0;
H4_k5k16 = 0;
H4_k8k12 = 0;
H4_k8k16 = 0;
```

```
  H4_k12k16 = 0;
  H4_k5k8k12 = 0;
  H4_k5k8k16 = 0;
  H4_k5k12k16 = 0;
  H4_k8k12k16 = 0;
  H4_4ac = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  H4_0ac_init = 1;
  k8 = 0.043;
  k12 = 0.101;
  rho58 = 4.1;
  rho512 = 13.7;
  rho516 = 1.44;
  rho812 = 2.5;
  sat = ;
  d = ;

  // Other declarations:
  var readout_observable_0ac, readout_observable_k5,
    readout_observable_k8;
  var readout_observable_k12, readout_observable_k16,
    readout_observable_k5k8;
  var readout_observable_k5k12, readout_observable_k5k16,
    readout_observable_k8k12;
  var readout_observable_k8k16, readout_observable_k12k16,
    readout_observable_k5k8k12;
  var readout_observable_k5k8k16, readout_observable_k5k12k16,
    readout_observable_k8k12k16;
  var readout_observable_4ac;
  const cell, H4_0ac_init, k8, k12, rho58, rho512, rho516, rho812, sat,
    d;
end
```

Listing 82: Archived Antimony for `blasi_4`; task `56836848_8`; selected version 10

```
// Created by libAntimony v3.1.3
model *H4kac()

  // Compartments and Species:
  compartment cell;
  species P0 in cell, P5 in cell, P8 in cell, P12 in cell, P16 in cell,
    P58 in cell;
  species P512 in cell, P516 in cell, P812 in cell, P816 in cell, P1216
    in cell;
  species P5812 in cell, P5816 in cell, P51216 in cell, P81216 in cell,
    P4ac in cell;

  // Assignment Rules:
  readout_observable_0ac := P0;
  readout_observable_k5 := P5;
  readout_observable_k8 := P8;
  readout_observable_k12 := P12;
  readout_observable_k16 := P16;
  readout_observable_k5k8 := P58;
  readout_observable_k5k12 := P512;
  readout_observable_k5k16 := P516;
  readout_observable_k8k12 := P812;
  readout_observable_k8k16 := P816;
  readout_observable_k12k16 := P1216;
  readout_observable_k5k8k12 := P5812;
  readout_observable_k5k8k16 := P5816;
```

```
readout_observable_k5k12k16 := P51216;
readout_observable_k8k12k16 := P81216;
readout_observable_4ac := P4ac;

// Reactions:
ac_P0_P5: P0 -> P5; a5*P0;
ac_P0_P8: P0 -> P8; a5*P0;
ac_P0_P12: P0 -> P12; a12*P0;
ac_P0_P16: P0 -> P16; a5*P0;
ac_P5_P58: P5 -> P58; a5*m58*P5;
ac_P5_P512: P5 -> P512; a12*m512*P5;
ac_P5_P516: P5 -> P516; a5*m516*P5;
ac_P8_P58: P8 -> P58; a5*m58*P8;
ac_P8_P812: P8 -> P812; a12*m516*P8;
ac_P8_P816: P8 -> P816; a5*m516*P8;
ac_P12_P512: P12 -> P512; a5*m512*P12;
ac_P12_P812: P12 -> P812; a5*m516*P12;
ac_P12_P1216: P12 -> P1216; a5*m516*P12;
ac_P16_P516: P16 -> P516; a5*m516*P16;
ac_P16_P816: P16 -> P816; a5*m516*P16;
ac_P16_P1216: P16 -> P1216; a12*m516*P16;
ac_P58_P5812: P58 -> P5812; a12*m512*m516*t5812*P58;
ac_P58_P5816: P58 -> P5816; a5*m516*m516*P58;
ac_P512_P5812: P512 -> P5812; a5*m58*m516*t5812*P512;
ac_P512_P51216: P512 -> P51216; a5*m516*m516*t51216*P512;
ac_P516_P5816: P516 -> P5816; a5*m58*m516*t51216*P516;
ac_P516_P51216: P516 -> P51216; a12*m512*m516*t51216*P516;
ac_P812_P5812: P812 -> P5812; a5*m58*m512*t5812*P812;
ac_P812_P81216: P812 -> P81216; a5*m516*m516*P812;
ac_P816_P5816: P816 -> P5816; a5*m58*m516*P816;
ac_P816_P81216: P816 -> P81216; a12*m516*m516*P816;
ac_P1216_P51216: P1216 -> P51216; a5*m512*m516*t51216*P1216;
ac_P1216_P81216: P1216 -> P81216; a5*m516*m516*P1216;
ac_P5812_P4ac: P5812 -> P4ac; a5*m516*m516*m516*t51216*m58*P5812;
ac_P5816_P4ac: P5816 -> P4ac; a12*m512*m516*m516*t5812*t51216*m58*
  P5816;
ac_P51216_P4ac: P51216 -> P4ac; a5*m58*m516*m516*t5812*m58*P51216;
ac_P81216_P4ac: P81216 -> P4ac; a5*m58*m512*m516*t5812*t51216*t51216*
  m58*P81216;
de_P5_P0: P5 -> P0; d*P5;
de_P8_P0: P8 -> P0; d*P8;
de_P12_P0: P12 -> P0; d*P12;
de_P16_P0: P16 -> P0; d*P16;
de_P58_P5: P58 -> P5; d*P58;
de_P58_P8: P58 -> P8; d*P58;
de_P512_P5: P512 -> P5; d*P512;
de_P512_P12: P512 -> P12; d*P512;
de_P516_P5: P516 -> P5; d*P516;
de_P516_P16: P516 -> P16; d*P516;
de_P812_P8: P812 -> P8; d*P812;
de_P812_P12: P812 -> P12; d*P812;
de_P816_P8: P816 -> P8; d*P816;
de_P816_P16: P816 -> P16; d*P816;
de_P1216_P12: P1216 -> P12; d*P1216;
de_P1216_P16: P1216 -> P16; d*P1216;
de_P5812_P58: P5812 -> P58; d*P5812;
de_P5812_P512: P5812 -> P512; d*P5812;
de_P5812_P812: P5812 -> P812; d*P5812;
de_P5816_P58: P5816 -> P58; d*P5816;
de_P5816_P516: P5816 -> P516; d*P5816;
de_P5816_P816: P5816 -> P816; d*P5816;
de_P51216_P512: P51216 -> P512; d*P51216;
de_P51216_P516: P51216 -> P516; d*P51216;
de_P51216_P1216: P51216 -> P1216; d*P51216;
de_P81216_P812: P81216 -> P812; d*P81216;
```

```
  de_P81216_P816: P81216 -> P816; d*P81216;
  de_P81216_P1216: P81216 -> P1216; d*P81216;
  de_P4ac_P5812: P4ac -> P5812; d*P4ac;
  de_P4ac_P5816: P4ac -> P5816; d*P4ac;
  de_P4ac_P51216: P4ac -> P51216; d*P4ac;
  de_P4ac_P81216: P4ac -> P81216; d*P4ac;

  // Species initializations:
  P0 = P0_init;
  P5 = ;
  P8 = ;
  P12 = ;
  P16 = ;
  P58 = ;
  P512 = ;
  P516 = ;
  P812 = ;
  P816 = ;
  P1216 = ;
  P5812 = ;
  P5816 = ;
  P51216 = ;
  P81216 = ;
  P4ac = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  P0_init = 1;
  a5 = 0.0427;
  a12 = 0.1302;
  m58 = 1;
  m512 = 5.2911;
  m516 = 2.055;
  t5812 = 1;
  t51216 = 1;
  d = ;

  // Other declarations:
  var readout_observable_0ac, readout_observable_k5,
    readout_observable_k8;
  var readout_observable_k12, readout_observable_k16,
    readout_observable_k5k8;
  var readout_observable_k5k12, readout_observable_k5k16,
    readout_observable_k8k12;
  var readout_observable_k8k16, readout_observable_k12k16,
    readout_observable_k5k8k12;
  var readout_observable_k5k8k16, readout_observable_k5k12k16,
    readout_observable_k8k12k16;
  var readout_observable_4ac;
  const cell, P0_init, a5, a12, m58, m512, m516, t5812, t51216, d;
end
```

Listing 83: Archived Antimony for `blasi_5`; task `56836848_9`; selected version 10

```
// Created by libAntimony v3.1.3
model *H4_acetylation()

  // Compartments and Species:
  compartment cell;
  species H0ac in cell, Hk5 in cell, Hk8 in cell, Hk12 in cell, Hk16 in cell;
  species Hk5k8 in cell, Hk5k12 in cell, Hk5k16 in cell, Hk8k12 in cell, Hk8k16 in cell;
  species Hk12k16 in cell, Hk5k8k12 in cell, Hk5k8k16 in cell, Hk5k12k16 in cell;
  species Hk8k12k16 in cell, H4ac in cell;

  // Assignment Rules:
  readout_observable_0ac := H0ac;
  readout_observable_k5 := Hk5;
  readout_observable_k8 := Hk8;
  readout_observable_k12 := Hk12;
  readout_observable_k16 := Hk16;
  readout_observable_k5k8 := Hk5k8;
  readout_observable_k5k12 := Hk5k12;
  readout_observable_k5k16 := Hk5k16;
  readout_observable_k8k12 := Hk8k12;
  readout_observable_k8k16 := Hk8k16;
  readout_observable_k12k16 := Hk12k16;
  readout_observable_k5k8k12 := Hk5k8k12;
  readout_observable_k5k8k16 := Hk5k8k16;
  readout_observable_k5k12k16 := Hk5k12k16;
  readout_observable_k8k12k16 := Hk8k12k16;
  readout_observable_4ac := H4ac;

  // Reactions:
  ac5_empty: H0ac -> Hk5; a_basal*H0ac;
  ac8_empty: H0ac -> Hk8; a_basal*H0ac;
  ac12_empty: H0ac -> Hk12; a12*H0ac;
  ac16_empty: H0ac -> Hk16; a_basal*H0ac;
  ac8_fromk5: Hk5 -> Hk5k8; a_basal*G58*Hk5;
  ac12_fromk5: Hk5 -> Hk5k12; a12*G5k12*Hk5;
  ac16_fromk5: Hk5 -> Hk5k16; a_basal*Hk5;
  ac5_fromk8: Hk8 -> Hk5k8; a_basal*G58*Hk8;
  ac12_fromk8: Hk8 -> Hk8k12; a12*G8k12*Hk8;
  ac16_fromk8: Hk8 -> Hk8k16; a_basal*G8k12*Hk8;
  ac5_fromk12: Hk12 -> Hk5k12; a_basal*G5k12*Hk12;
  ac8_fromk12: Hk12 -> Hk8k12; a_basal*G8k12*Hk12;
  ac16_fromk12: Hk12 -> Hk12k16; a_basal*G8k12*Hk12;
  ac5_fromk16: Hk16 -> Hk5k16; a_basal*Hk16;
  ac8_fromk16: Hk16 -> Hk8k16; a_basal*G8k12*Hk16;
  ac12_fromk16: Hk16 -> Hk12k16; a12*G8k12*Hk16;
  ac12_fromk5k8: Hk5k8 -> Hk5k8k12; a12*G5k12*G8k12*T5812*Hk5k8;
  ac16_fromk5k8: Hk5k8 -> Hk5k8k16; a_basal*G8k12*Ttrip16*Hk5k8;
  ac8_fromk5k12: Hk5k12 -> Hk5k8k12; a_basal*G58*G8k12*T5812*Hk5k12;
  ac16_fromk5k12: Hk5k12 -> Hk5k12k16; a_basal*G8k12*T5k12k16*Hk5k12;
  ac8_fromk5k16: Hk5k16 -> Hk5k8k16; a_basal*G58*G8k12*Ttrip16*Hk5k16;
  ac12_fromk5k16: Hk5k16 -> Hk5k12k16; a12*G5k12*G8k12*T5k12k16*Hk5k16;
  ac5_fromk8k12: Hk8k12 -> Hk5k8k12; a_basal*G58*G5k12*T5812*Hk8k12;
  ac16_fromk8k12: Hk8k12 -> Hk8k12k16; a_basal*G8k12^2*Ttrip16*Hk8k12;
  ac5_fromk8k16: Hk8k16 -> Hk5k8k16; a_basal*G58*Ttrip16*Hk8k16;
  ac12_fromk8k16: Hk8k16 -> Hk8k12k16; a12*G8k12^2*Ttrip16*Hk8k16;
  ac5_fromk12k16: Hk12k16 -> Hk5k12k16; a_basal*G5k12*T5k12k16*Hk12k16;
  ac8_fromk12k16: Hk12k16 -> Hk8k12k16; a_basal*G8k12^2*Ttrip16*Hk12k16;
  ac16_fromk5k8k12: Hk5k8k12 -> H4ac; a_basal*G8k12^2*Ttrip16^2*T5k12k16*Hk5k8k12;
```

```
ac12_fromk5k8k16: Hk5k8k16 -> H4ac; a12*G5k12*G8k12^2*T5812*T5k12k16*
  Ttrip16*Hk5k8k16;
ac8_fromk5k12k16: Hk5k12k16 -> H4ac; a_basal*G58*G8k12^2*T5812*
  Ttrip16^2*Hk5k12k16;
ac5_fromk8k12k16: Hk8k12k16 -> H4ac; a_basal*G58*G5k12*T5812*Ttrip16*
  T5k12k16*Hk8k12k16;
de5_fromk5: Hk5 -> H0ac; d*Hk5;
de8_fromk8: Hk8 -> H0ac; d*Hk8;
de12_fromk12: Hk12 -> H0ac; d*Hk12;
de16_fromk16: Hk16 -> H0ac; d*Hk16;
de5_fromk5k8: Hk5k8 -> Hk8; d*Hk5k8;
de8_fromk5k8: Hk5k8 -> Hk5; d*Hk5k8;
de5_fromk5k12: Hk5k12 -> Hk12; d*Hk5k12;
de12_fromk5k12: Hk5k12 -> Hk5; d*Hk5k12;
de5_fromk5k16: Hk5k16 -> Hk16; d*Hk5k16;
de16_fromk5k16: Hk5k16 -> Hk5; d*Hk5k16;
de8_fromk8k12: Hk8k12 -> Hk12; d*Hk8k12;
de12_fromk8k12: Hk8k12 -> Hk8; d*Hk8k12;
de8_fromk8k16: Hk8k16 -> Hk16; d*Hk8k16;
de16_fromk8k16: Hk8k16 -> Hk8; d*Hk8k16;
de12_fromk12k16: Hk12k16 -> Hk16; d*Hk12k16;
de16_fromk12k16: Hk12k16 -> Hk12; d*Hk12k16;
de5_fromk5k8k12: Hk5k8k12 -> Hk8k12; d*Hk5k8k12;
de8_fromk5k8k12: Hk5k8k12 -> Hk5k12; d*Hk5k8k12;
de12_fromk5k8k12: Hk5k8k12 -> Hk5k8; d*Hk5k8k12;
de5_fromk5k8k16: Hk5k8k16 -> Hk8k16; d*Hk5k8k16;
de8_fromk5k8k16: Hk5k8k16 -> Hk5k16; d*Hk5k8k16;
de16_fromk5k8k16: Hk5k8k16 -> Hk5k8; d*Hk5k8k16;
de5_fromk5k12k16: Hk5k12k16 -> Hk12k16; d*Hk5k12k16;
de12_fromk5k12k16: Hk5k12k16 -> Hk5k16; d*Hk5k12k16;
de16_fromk5k12k16: Hk5k12k16 -> Hk5k12; d*Hk5k12k16;
de5_fromk8k12k16: Hk8k12k16 -> Hk12k16; d*Hk8k12k16;
de8_fromk8k12k16: Hk8k12k16 -> Hk8k16; d*Hk8k12k16;
de16_fromk8k12k16: Hk8k12k16 -> Hk8k12; d*Hk8k12k16;
de5_from4ac: H4ac -> Hk8k12k16; d*H4ac;
de8_from4ac: H4ac -> Hk5k12k16; d*H4ac;
de12_from4ac: H4ac -> Hk5k8k16; d*H4ac;
de16_from4ac: H4ac -> Hk5k8k12; d*H4ac;

// Species initializations:
H0ac = H0ac_init;
Hk5 = 0;
Hk8 = 0;
Hk12 = 0;
Hk16 = 0;
Hk5k8 = 0;
Hk5k12 = 0;
Hk5k16 = 0;
Hk8k12 = 0;
Hk8k16 = 0;
Hk12k16 = 0;
Hk5k8k12 = 0;
Hk5k8k16 = 0;
Hk5k12k16 = 0;
Hk8k12k16 = 0;
H4ac = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
H0ac_init = 1;
a_basal = 0.0462;
a12 = 0.0991;
G58 = 3.78;
```

```
  G5k12 = 13.55;
  G8k12 = 2.23;
  T5812 = 0.366;
  T5k12k16 = 0.506;
  Ttrip16 = 1.98;
  d = 1;

  // Other declarations:
  var readout_observable_0ac, readout_observable_k5,
    readout_observable_k8;
  var readout_observable_k12, readout_observable_k16,
    readout_observable_k5k8;
  var readout_observable_k5k12, readout_observable_k5k16,
    readout_observable_k8k12;
  var readout_observable_k8k16, readout_observable_k12k16,
    readout_observable_k5k8k12;
  var readout_observable_k5k8k16, readout_observable_k5k12k16,
    readout_observable_k8k12k16;
  var readout_observable_4ac;
  const cell, H0ac_init, a_basal, a12, G58, G5k12, G8k12, T5812,
    T5k12k16;
  const Ttrip16, d;
end
```

H.12 SCHWEN

Listing 84: Archived Antimony for `schwen_1`; task `56836812_0`; selected version 8

```
// Created by libAntimony v3.1.3
model *insulin_hepatocyte()

  // Compartments and Species:
  compartment cell;
  species Ins in cell, R1 in cell, C1s in cell, C1i in cell, R2 in cell
    , C2s in cell;
  species C2i in cell, Frag in cell, NS;

  // Assignment Rules:
  readout_IR1 := C1s + C1i;
  readout_IR2 := C2s + C2i;
  readout_IRsum := eta_low*(C1s + C1i) + (1 - eta_low)*(C2s + C2i);
  readout_insulin := Ins;
  readout_insulin_fragments := Frag;

  // Reactions:
  deg1: C1i -> R1 + Frag; kdeg*C1i;
  deg2: C2i -> R2 + Frag; kdeg*C2i;
  fragloss: Frag -> ; kloss*Frag;
  ns_bind: Ins -> NS; kns*Ins*(NScap - NS);
  ns_deg: NS -> Frag; kdegNS1*NS;
  capture1: Ins + R1 -> C1i; kon*Ins*R1;
  capture2: Ins + R2 -> C2i; kon*Ins*R2;

  // Species initializations:
  Ins = init_Ins;
  R1 = ini_R1;
  C1s = 0;
  C1i = 0;
  R2 = ini_R1*ini_R2fold;
  C2s = 0;
  C2i = 0;
  Frag = 0;
  NS = ;

```

```
  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  init_Ins = 0;
  ini_R1 = 58.85;
  ini_R2fold = 15.6;
  eta_low = 0.605;
  kon = 0.005;
  kdeg = 0.1;
  kloss = 0.005;
  kns = ;
  NScap = ;
  kdegNS1 = ;

  // Other declarations:
  var readout_IR1, readout_IR2, readout_IRsum, readout_insulin,
    readout_insulin_fragments;
  const cell, init_Ins, ini_R1, ini_R2fold, eta_low, kon, kdeg, kloss,
    kns;
  const NScap, kdegNS1;
end
```

Listing 85: Archived Antimony for `schwen_2`; task `56836812_1`; selected version 18

```
// Created by libAntimony v3.1.3
// Compartments and Species:
compartment cell;
species Ins in cell, Frag in cell, R1 in cell, C1 in cell, R2a in cell,
     C2a in cell;
species R2b in cell, C2b in cell, R3 in cell, C3 in cell;

// Assignment Rules:
readout_IR1 := C1;
readout_IR2 := C2a + C2b;
readout_IRsum := eta_low*C1 + (1 - eta_low)*(C2a + C2b);
readout_insulin := Ins;
readout_insulin_fragments := Frag;

// Reactions:
bind1: Ins + R1 -> C1; kon1*Ins*R1;
unbind1: C1 -> Ins + R1; koff1*C1;
bind2b: Ins + R2b -> C2b; kon_b*Ins*R2b;
unbind2b: C2b -> Ins + R2b; koff_b*C2b;
bind2a: Ins + R2a -> C2a; 10*Ins*R2a;
unbind2a: C2a -> Ins + R2a; koff_a*C2a;
bind3: Ins + R3 -> C3; Ins*R3;
unbind3: C3 -> Ins + R3; koff3*C3;
deg3: C3 -> R3 + Frag; kdeg*C3;

// Species initializations:
Ins = init_Ins;
Frag = 0;
R1 = ini_R1;
C1 = 0;
R2a = f_fast*ini_R1*ini_R2fold;
C2a = 0;
R2b = (1 - f_fast)*ini_R1*ini_R2fold;
C2b = 0;
R3 = 100*ini_R1;
C3 = 0;

// Compartment initializations:
cell = 1;

```

```
// Variable initializations:
init_Ins = 0.1;
ini_R1 = ;
f_fast = 0.04;
ini_R2fold = ;
kon1 = 0.0001;
koff1 = 0.1;
koff_a = 4;
kon_b = 5e-05;
koff_b = 0.02;
koff3 = 1;
kdeg = 0.004;
eta_low = ;

// Other declarations:
var readout_IR1, readout_IR2, readout_IRsum, readout_insulin,
    readout_insulin_fragments;
const cell, init_Ins, ini_R1, f_fast, ini_R2fold, kon1, koff1, koff_a,
    kon_b;
const koff_b, koff3, kdeg, eta_low;
```

Listing 86: Archived Antimony for `schwen_3`; task `56836812_2`; selected version 7

```
// Created by libAntimony v3.1.3
model *insulin_liver()

  // Compartments and Species:
  compartment cell;
  species Ins in cell, Frag in cell, R1 in cell, R2 in cell, C1s in
    cell;
  species C2s in cell, C1i in cell, C2i in cell;

  // Assignment Rules:
  readout_IR1 := C1s + C1i;
  readout_IR2 := C2s + C2i;
  readout_IRsum := eta_low*(C1s + C1i) + (1 - eta_low)*(C2s + C2i);
  readout_insulin := Ins;
  readout_insulin_fragments := Frag;

  // Reactions:
  bind1: Ins + R1 -> C1s; kon1*Ins*R1;
  diss1: C1s -> Ins + R1; Kc1*C1s*C1s/(R1 + C1s);
  bind2: Ins + R2 -> C2s; kon2*Ins*R2;
  diss2: C2s -> Ins + R2; Kc2*C2s*C2s/(R2 + C2s);
  int1: C1s -> C1i; kint*C1s;
  int2: C2s -> C2i; kint*C2s;
  ret1: C1i -> Ins + R1; kret*C1i;
  ret2: C2i -> Ins + R2; kret*C2i;
  deg1: C1i -> Frag + R1; kdeg*C1i;
  deg2: C2i -> Frag + R2; kdeg*C2i;

  // Species initializations:
  Ins = init_Ins;
  Frag = 0;
  R1 = ini_R1;
  R2 = ini_R1*ini_R2fold;
  C1s = 0;
  C2s = 0;
  C1i = 0;
  C2i = 0;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
```

```
  init_Ins = 10;
  ini_R1 = 58.85;
  ini_R2fold = 15.6;
  eta_low = 0.605;
  kon1 = 0.0016;
  kon2 = 0.001;
  kret = 0.02;
  kdeg = 0.008;
  Kc1 = ;
  Kc2 = ;
  kint = ;

  // Other declarations:
  var readout_IR1, readout_IR2, readout_IRsum, readout_insulin,
    readout_insulin_fragments;
  const cell, init_Ins, ini_R1, ini_R2fold, eta_low, kon1, kon2, kret,
    kdeg;
  const Kc1, Kc2, kint;
end
```

Listing 87: Archived Antimony for `schwen_4`; task `56836812_3`; selected version 12

```
// Created by libAntimony v3.1.3
model *insulin_receptor_hepatocytes()

  // Compartments and Species:
  compartment cell;
  species Ins in cell, Frag in cell, R1 in cell, C1i in cell, R2 in
    cell;
  species C2s in cell, C2i in cell, Ads;

  // Assignment Rules:
  readout_IR1 := C1i + R1*Ins/Kd1;
  readout_IR2 := C2s + C2i;
  readout_IRsum := eta_low*(C1i + R1*Ins/Kd1) + (1 - eta_low)*(C2s +
    C2i);
  readout_insulin := Ins;
  readout_insulin_fragments := Frag;

  // Reactions:
  degrad1: C1i -> R1 + Frag; kdeg*C1i;
  bind2: Ins + R2 -> C2s; kon2*Ins*R2;
  dissoc2: C2s -> Ins + R2; kon2*Kd2*C2s;
  endocytosis2: C2s -> R2 + C2i; kint2*C2s;
  efflux2: C2i -> Frag; ksec2*C2i;
  adsorb: Ins -> Ads; k_ads*Ins*(1 - Ads/Ads_max);
  release: Ads -> Frag; k_rel*Ads;
  fragdeg: Frag -> ; kfragdeg*Frag;
  c1i_fill: R1 -> C1i; (kint1*Ins/Kd1)*R1;

  // Species initializations:
  Ins = init_Ins*kappa;
  Frag = 0;
  R1 = ini_R1;
  C1i = 0;
  R2 = ini_R1*ini_R2fold;
  C2s = 0;
  C2i = 0;
  Ads = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  init_Ins = 1;
```

```
  kappa = ;
  ini_R1 = 58.85;
  ini_R2fold = 15.61;
  eta_low = 0.605;
  kdeg = 0.01;
  Kd1 = ;
  kon2 = ;
  Kd2 = ;
  kint2 = ;
  ksec2 = ;
  k_ads = ;
  Ads_max = ;
  k_rel = ;
  kfragdeg = ;
  kint1 = ;

  // Other declarations:
  var readout_IR1, readout_IR2, readout_IRsum, readout_insulin,
    readout_insulin_fragments;
  const cell, init_Ins, kappa, ini_R1, ini_R2fold, eta_low, kdeg, Kd1,
    kon2;
  const Kd2, kint2, ksec2, k_ads, Ads_max, k_rel, kfragdeg, kint1;
end
```

Listing 88: Archived Antimony for `schwen_5`; task `56836812_4`; selected version 3

```
// Created by libAntimony v3.1.3
model *insulin_hepatocyte_two_entity()

  // Compartments and Species:
  compartment cell;
  species E in cell, F in cell, R1 in cell, B1 in cell, I1 in cell, R2
    in cell;
  species B2 in cell, I2 in cell;

  // Assignment Rules:
  kon1 := koff1/Kd1;
  kon2 := koff1/Kd2;
  readout_IR1 := B1 + I1;
  readout_IR2 := B2 + I2;
  readout_IRsum := eta_low*(B1 + I1) + (1 - eta_low)*(B2 + I2);
  readout_insulin := E;
  readout_insulin_fragments := F;

  // Reactions:
  bind1: R1 + E -> B1; kon1*R1*E;
  unbind1: B1 -> R1 + E; koff1*B1;
  int1: B1 -> I1; kint1*B1;
  ret1: I1 -> R1 + F; kret1*I1;
  bind2: R2 + E -> B2; kon2*R2*E;
  unbind2: B2 -> R2 + E; koff1*B2;
  int2: B2 -> I2; kint2*B2;
  ret2: I2 -> R2 + F; kret2*I2;
  floss: F -> ; kout*F;

  // Species initializations:
  E = init_Ins;
  F = 0;
  R1 = ini_R1;
  B1 = 0;
  I1 = 0;
  R2 = ini_R1*ini_R2fold;
  B2 = 0;
  I2 = 0;

```

```
  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  init_Ins = 1;
  ini_R1 = 58.85;
  ini_R2fold = 15.6;
  eta_low = 0.605;
  Kd1 = 20;
  koff1 = 0.1;
  kint1 = 0.3;
  kret1 = 0.05;
  Kd2 = 300;
  kint2 = 0.3;
  kret2 = 0.05;
  kout = ;

  // Other declarations:
  var kon1, kon2, readout_IR1, readout_IR2, readout_IRsum,
    readout_insulin;
  var readout_insulin_fragments;
  const cell, init_Ins, ini_R1, ini_R2fold, eta_low, Kd1, koff1, kint1,
     kret1;
  const Kd2, kint2, kret2, kout;
end
```

H.13 BACHMANN

Listing 89: Archived Antimony for `bachmann_1`; task `56836792_5`; selected version 11

```
// Created by libAntimony v3.1.3
model *epo_jak2_stat5_feedback()

  // Compartments and Species:
  compartment cyt, nuc;
  species EpoRJAK2 in cyt, EpoR_pY in cyt, pEpoRJAK2 in cyt,
    EpoRJAK2_JpY in cyt;
  species EpoRJAK2_CIS in cyt, EpoRJAK2_SOCS3 in cyt, STAT5 in cyt,
    pSTAT5 in cyt;
  species pSTAT5_nuc in nuc, CISpreRNA in nuc, CISmRNA in cyt, CISprot
    in cyt;
  species SOCS3preRNA in nuc, SOCS3mRNA in cyt, SOCS3prot in cyt,
    EpoRJAK2_CISp;

  // Assignment Rules:
  SHP1 := init_SHP1*(1 + init_SHP1_multiplier*SHP1ProOE);
  readout_pEpoR := pEpoRJAK2 + EpoRJAK2_SOCS3;
  readout_pJAK2 := pEpoRJAK2 + EpoRJAK2_JpY + EpoRJAK2_CISp;
  readout_pSTAT5 := pSTAT5 + pSTAT5_nuc;
  readout_pSTAT5_rel := (pSTAT5 + pSTAT5_nuc)/(STAT5 + pSTAT5 +
    pSTAT5_nuc);
  readout_tSTAT5 := STAT5 + pSTAT5 + pSTAT5_nuc;
  readout_STAT5_abs := STAT5 + pSTAT5 + pSTAT5_nuc;
  readout_CISRNA_fold := CISmRNA/CISmRNA_init - 1;
  readout_SOCS3RNA_fold := SOCS3mRNA/SOCS3mRNA_init - 1;
  readout_CIS_au := CISprot + EpoRJAK2_CIS + EpoRJAK2_CISp;
  readout_CIS_abs := CISprot + EpoRJAK2_CIS + EpoRJAK2_CISp;
  readout_SOCS3_au := SOCS3prot + EpoRJAK2_SOCS3;
  readout_SOCS3_abs := SOCS3prot + EpoRJAK2_SOCS3;
  readout_SHP1_abs := SHP1;
  readout_tSHP1 := SHP1;

  // Reactions:
```

```
rec_activation: EpoRJAK2 -> pEpoRJAK2; k_act*EpoRJAK2*Epo^n_Epo/(
  Kd_Epo^n_Epo + Epo^n_Epo);
rec_activation_basal: EpoRJAK2 -> pEpoRJAK2; k_act_basal*EpoRJAK2;
rec_JAK2_dephos2: EpoRJAK2_JpY -> EpoRJAK2; k_Rdep*SHP1*EpoRJAK2_JpY;
rec_R_dephos: pEpoRJAK2 -> EpoRJAK2_JpY; k_Rdep*SHP1*pEpoRJAK2;
rec_R_dephos2: EpoR_pY -> EpoRJAK2; k_Rdep*SHP1*EpoR_pY;
cis_release: EpoRJAK2_CIS -> EpoRJAK2_JpY + CISprot; k_cis_release*
  EpoRJAK2_CIS;
socs3_binding: pEpoRJAK2 + SOCS3prot -> EpoRJAK2_SOCS3; k_socs3_bind*
  pEpoRJAK2*SOCS3prot;
stat5_phos: STAT5 -> pSTAT5; k_stat5*STAT5*pEpoRJAK2;
stat5_import: pSTAT5 -> pSTAT5_nuc; k_imp*pSTAT5*cyt;
stat5_export: pSTAT5_nuc -> STAT5; k_exp*pSTAT5_nuc*nuc;
cis_transcription:  -> CISpreRNA; k_tx_CIS*pSTAT5_nuc;
cis_processing: CISpreRNA -> CISmRNA; k_mat_CIS*CISpreRNA*nuc;
cis_mRNA_deg: CISmRNA -> ; k_deg_mRNA_CIS*(1 - ActD*f_stab_CIS)*
  CISmRNA;
cis_translation:  -> CISprot; k_transl_CIS*CISmRNA;
cis_prot_deg: CISprot -> ; k_deg_CISprot*CISprot;
socs3_transcription:  -> SOCS3preRNA; k_tx_SOCS3*pSTAT5_nuc;
socs3_processing: SOCS3preRNA -> SOCS3mRNA; k_mat_SOCS3*SOCS3preRNA*
  nuc;
socs3_mRNA_deg: SOCS3mRNA -> ; k_deg_mRNA_SOCS3*(1 - ActD*
  f_stab_SOCS3)*SOCS3mRNA;
socs3_translation:  -> SOCS3prot; k_transl_SOCS3*SOCS3mRNA;
socs3_prot_deg: SOCS3prot -> ; k_deg_SOCS3prot*SOCS3prot;
cis_binding: pEpoRJAK2 + CISprot -> EpoRJAK2_CISp; k_cis_bind*
  pEpoRJAK2*CISprot;
cis_activation: EpoRJAK2_CIS -> EpoRJAK2_CISp; (k_act*Epo^n_Epo/(
  Kd_Epo^n_Epo + Epo^n_Epo) + k_act_basal)*EpoRJAK2_CIS;
socs3_reset: EpoRJAK2_SOCS3 -> EpoRJAK2 + SOCS3prot; k_socs3_reset*
  EpoRJAK2_SOCS3;
cis_shp1_reset: EpoRJAK2_CISp -> EpoRJAK2 + CISprot; k_Rdep*SHP1*
  EpoRJAK2_CISp;

// Species initializations:
EpoRJAK2 = init_EpoRJAK2;
EpoR_pY = 0;
pEpoRJAK2 = 0;
EpoRJAK2_JpY = 0;
EpoRJAK2_CIS = CISoe*init_EpoRJAK2_CIS*CISEqcOE;
EpoRJAK2_SOCS3 = 0;
STAT5 = init_STAT5;
pSTAT5 = 0;
pSTAT5_nuc = 0;
CISpreRNA = 0;
CISmRNA = CISmRNA_init;
CISprot = init_CIS_multiplier*CISEqcOE;
SOCS3preRNA = 0;
SOCS3mRNA = SOCS3mRNA_init;
SOCS3prot = SOCS3oe*init_SOCS3_multiplier*SOCS3EqcOE;
EpoRJAK2_CISp = ;

// Compartment initializations:
cyt = 0.4;
nuc = 0.275;

// Variable initializations:
init_EpoRJAK2 = ;
CISoe = ;
init_EpoRJAK2_CIS = ;
CISEqcOE = ;
init_STAT5 = ;
CISmRNA_init = ;
init_CIS_multiplier = ;
```

```
  SOCS3mRNA_init = ;
  SOCS3oe = ;
  init_SOCS3_multiplier = ;
  SOCS3EqcOE = ;
  init_SHP1 = ;
  init_SHP1_multiplier = ;
  SHP1ProOE = ;
  k_act = ;
  Epo = ;
  n_Epo = ;
  Kd_Epo = ;
  k_act_basal = ;
  k_Rdep = ;
  k_cis_release = ;
  k_socs3_bind = ;
  k_stat5 = ;
  k_imp = ;
  k_exp = ;
  k_tx_CIS = ;
  k_mat_CIS = ;
  k_deg_mRNA_CIS = ;
  ActD = ;
  f_stab_CIS = ;
  k_transl_CIS = ;
  k_deg_CISprot = ;
  k_tx_SOCS3 = ;
  k_mat_SOCS3 = ;
  k_deg_mRNA_SOCS3 = ;
  f_stab_SOCS3 = ;
  k_transl_SOCS3 = ;
  k_deg_SOCS3prot = ;
  k_cis_bind = ;
  k_socs3_reset = ;

  // Other declarations:
  var SHP1, readout_pEpoR, readout_pJAK2, readout_pSTAT5,
    readout_pSTAT5_rel;
  var readout_tSTAT5, readout_STAT5_abs, readout_CISRNA_fold,
    readout_SOCS3RNA_fold;
  var readout_CIS_au, readout_CIS_abs, readout_SOCS3_au,
    readout_SOCS3_abs;
  var readout_SHP1_abs, readout_tSHP1;
  const cyt, nuc, init_EpoRJAK2, CISoe, init_EpoRJAK2_CIS, CISEqcOE,
    init_STAT5;
  const CISmRNA_init, init_CIS_multiplier, SOCS3mRNA_init, SOCS3oe,
    init_SOCS3_multiplier;
  const SOCS3EqcOE, init_SHP1, init_SHP1_multiplier, SHP1ProOE, k_act,
    Epo;
  const n_Epo, Kd_Epo, k_act_basal, k_Rdep, k_cis_release, k_socs3_bind
    , k_stat5;
  const k_imp, k_exp, k_tx_CIS, k_mat_CIS, k_deg_mRNA_CIS, ActD,
    f_stab_CIS;
  const k_transl_CIS, k_deg_CISprot, k_tx_SOCS3, k_mat_SOCS3,
    k_deg_mRNA_SOCS3;
  const f_stab_SOCS3, k_transl_SOCS3, k_deg_SOCS3prot, k_cis_bind,
    k_socs3_reset;
end
```

Listing 90: Archived Antimony for bachmann_2; task 56836792_6; selected version 9

```
// Created by libAntimony v3.1.3
model *epor_stat5()

  // Compartments and Species:
  compartment cyt, nuc;
  species Epo in cyt, ActD in cyt, CISoe in cyt, SOCS3oe in cyt,
    init_CIS_multiplier in cyt;
  species init_SOCS3_multiplier in cyt, init_SHP1_multiplier in cyt,
    init_EpoRJAK2_CIS in cyt;
  species EpoRJAK2 in cyt, EpoRJAK2_Epo in cyt, JAK2_act in cyt,
    EpoR_act in cyt;
  species EpoR_act_SOCS3 in cyt, EpoRJAK2_CIS in cyt, EpoRJAK2_CIS_Epo
    in cyt;
  species JAK2_act_CIS in cyt, STAT5_cyt in cyt, pSTAT5_cyt in cyt,
    pSTAT5_nuc in nuc;
  species SHP1 in cyt, SHP1_act in cyt, mRNA_CIS in cyt, mRNA_SOCS3 in
    cyt;
  species CIS in cyt, SOCS3 in cyt;

  // Assignment Rules:
  readout_CISRNA_fold := mRNA_CIS - 1;
  readout_SOCS3RNA_fold := mRNA_SOCS3 - 1;
  readout_CIS_abs := CIS*cyt;
  readout_CIS_au := CIS*cyt;
  readout_SOCS3_abs := SOCS3*cyt;
  readout_SOCS3_au := SOCS3*cyt;
  readout_SHP1_abs := (SHP1 + SHP1_act)*cyt;
  readout_tSHP1 := (SHP1 + SHP1_act)*cyt;
  readout_STAT5_abs := STAT5_cyt*cyt + pSTAT5_cyt*cyt + pSTAT5_nuc*nuc;
  readout_tSTAT5 := STAT5_cyt*cyt + pSTAT5_cyt*cyt + pSTAT5_nuc*nuc;
  readout_pEpoR := EpoR_act + EpoR_act_SOCS3;
  readout_pJAK2 := JAK2_act + JAK2_act_CIS + EpoR_act;
  readout_pSTAT5 := pSTAT5_cyt*cyt + pSTAT5_nuc*nuc;
  readout_pSTAT5_rel := (pSTAT5_cyt*cyt + pSTAT5_nuc*nuc)/(init_STAT5*
    cyt);

  // Reactions:
  epo_bind: EpoRJAK2 -> EpoRJAK2_Epo; k_on*Epo*EpoRJAK2;
  epo_unbind: EpoRJAK2_Epo -> EpoRJAK2; k_off*EpoRJAK2_Epo;
  jak_act: EpoRJAK2_Epo -> JAK2_act; k_act*EpoRJAK2_Epo;
  jak_deact: JAK2_act -> EpoRJAK2_Epo; k_depJ*SHP1_act*JAK2_act;
  tail_phos: JAK2_act -> EpoR_act; k_tail*JAK2_act;
  tail_dep: EpoR_act -> JAK2_act; k_depR*SHP1_act*EpoR_act;
  shp1_recruit: SHP1 -> SHP1_act; k_SHact*EpoR_act*SHP1;
  shp1_reset: SHP1_act -> SHP1; k_SHreset*SHP1_act;
  socs3_dock: EpoR_act + SOCS3 -> EpoR_act_SOCS3; k_S3_dock*SOCS3*
    EpoR_act;
  socs3_out: EpoR_act_SOCS3 -> EpoRJAK2 + SOCS3; k_S3_out*
    EpoR_act_SOCS3;
  cis_mark: EpoRJAK2 + CIS -> EpoRJAK2_CIS; k_CIS_mark*CIS*EpoRJAK2;
  cis_bind: EpoRJAK2_CIS -> EpoRJAK2_CIS_Epo; k_on*Epo*EpoRJAK2_CIS;
  cis_unbind: EpoRJAK2_CIS_Epo -> EpoRJAK2_CIS; k_off*EpoRJAK2_CIS_Epo;
  cis_jak_act: EpoRJAK2_CIS_Epo -> JAK2_act_CIS; k_act*EpoRJAK2_CIS_Epo
    ;
  cis_jak_deact: JAK2_act_CIS -> EpoRJAK2_CIS_Epo; k_depJ*SHP1_act*
    JAK2_act_CIS;
  cis_tail: JAK2_act_CIS -> EpoR_act; k_tail_CIS*JAK2_act_CIS;
  stat5_phos: STAT5_cyt -> pSTAT5_cyt; k_phos*EpoR_act*STAT5_cyt;
  stat5_import: pSTAT5_cyt -> pSTAT5_nuc; k_imp*pSTAT5_cyt;
  cis_mrna_syn:  -> mRNA_CIS; k_syn_CIS*pSTAT5_nuc*ActD;
  cis_mrna_deg: mRNA_CIS -> ; k_deg_CISm*(mRNA_CIS - 1);
  s3_mrna_syn:  -> mRNA_SOCS3; k_syn_S3*pSTAT5_nuc*ActD;
  s3_mrna_deg: mRNA_SOCS3 -> ; k_deg_S3m*(mRNA_SOCS3 - 1);
```

```
cis_tl:  -> CIS; k_tl_CIS*(mRNA_CIS - 1);
cis_deg: CIS -> ; k_deg_CISp*CIS;
s3_tl:  -> SOCS3; k_tl_CIS*(mRNA_SOCS3 - 1);
s3_deg: SOCS3 -> ; k_deg_S3p*SOCS3;
stat5_nuc_dep: pSTAT5_nuc -> STAT5_cyt; k_depnuc*pSTAT5_nuc;

// Species initializations:
Epo = ;
ActD = ;
CISoe = ;
SOCS3oe = ;
init_CIS_multiplier = ;
init_SOCS3_multiplier = ;
init_SHP1_multiplier = ;
init_EpoRJAK2_CIS = ;
EpoRJAK2 = init_EpoRJAK2*(1 - init_EpoRJAK2_CIS*CISoe);
EpoRJAK2_Epo = ;
JAK2_act = ;
EpoR_act = ;
EpoR_act_SOCS3 = ;
EpoRJAK2_CIS = init_EpoRJAK2_CIS*CISoe*init_EpoRJAK2;
EpoRJAK2_CIS_Epo = ;
JAK2_act_CIS = ;
STAT5_cyt = init_STAT5;
pSTAT5_cyt = ;
pSTAT5_nuc = ;
SHP1 = init_SHP1*(1 + init_SHP1_multiplier*(SHP1ProOE - 1));
SHP1_act = ;
mRNA_CIS = mRNA_CIS_init;
mRNA_SOCS3 = mRNA_SOCS3_init;
CIS = CISoe*init_CIS_multiplier*CISEqcOE;
SOCS3 = SOCS3oe*init_SOCS3_multiplier*SOCS3EqcOE;

// Compartment initializations:
cyt = 0.4;
nuc = 0.275;

// Variable initializations:
init_EpoRJAK2 = 1000;
init_STAT5 = 220;
init_SHP1 = 65;
SHP1ProOE = 4;
mRNA_CIS_init = 1;
mRNA_SOCS3_init = 1;
CISEqcOE = 300;
SOCS3EqcOE = 150;
k_on = 10000000;
k_off = 0.4;
k_act = 1;
k_depJ = 0.003;
k_tail = 3;
k_tail_CIS = 0.01;
k_depR = 0.005;
k_SHact = 0.002;
k_SHreset = 0.01;
k_S3_dock = 0.02;
k_S3_out = 0.05;
k_CIS_mark = 0.001;
k_phos = 0.0006;
k_imp = 0.3;
k_depnuc = 0.03;
k_syn_CIS = 0.002;
k_deg_CISm = 0.008;
k_syn_S3 = 0.002;
k_deg_S3m = 0.005;
```

```
  k_tl_CIS = 0.04;
  k_deg_CISp = 0.003;
  k_deg_S3p = 0.003;

  // Other declarations:
  var readout_CISRNA_fold, readout_SOCS3RNA_fold, readout_CIS_abs,
    readout_CIS_au;
  var readout_SOCS3_abs, readout_SOCS3_au, readout_SHP1_abs,
    readout_tSHP1;
  var readout_STAT5_abs, readout_tSTAT5, readout_pEpoR, readout_pJAK2,
    readout_pSTAT5;
  var readout_pSTAT5_rel;
  const cyt, nuc, init_EpoRJAK2, init_STAT5, init_SHP1, SHP1ProOE,
    mRNA_CIS_init;
  const mRNA_SOCS3_init, CISEqcOE, SOCS3EqcOE, k_on, k_off, k_act,
    k_depJ;
  const k_tail, k_tail_CIS, k_depR, k_SHact, k_SHreset, k_S3_dock,
    k_S3_out;
  const k_CIS_mark, k_phos, k_imp, k_depnuc, k_syn_CIS, k_deg_CISm,
    k_syn_S3;
  const k_deg_S3m, k_tl_CIS, k_deg_CISp, k_deg_S3p;
end
```

Listing 91: Archived Antimony for `bachmann_3`; task `56836792_7`; selected version 4

```
// Created by libAntimony v3.1.3
model *EpoR_STAT5_model()

  // Compartments and Species:
  compartment cyt, nuc;
  species EpoRJAK2 in cyt, EpoRJAK2_act in cyt, EpoRJAK2_CIS in cyt,
    STAT5c in cyt;
  species pSTAT5c in cyt, pSTAT5n in nuc, CISRNA in cyt, CIS in cyt,
    SOCS3RNA in cyt;
  species SOCS3 in cyt, SHP1 in cyt, EpoRJAK2_p;

  // Assignment Rules:
  EpoAct := Epo^n_epo/(KD_epo^n_epo + Epo^n_epo);
  socs3_inhib := Ki_socs3/(Ki_socs3 + SOCS3);
  readout_CISRNA_fold := CISRNA - 1;
  readout_SOCS3RNA_fold := SOCS3RNA - 1;
  readout_CIS_abs := CIS;
  readout_CIS_au := CIS;
  readout_SHP1_abs := SHP1;
  readout_SOCS3_abs := SOCS3;
  readout_SOCS3_au := SOCS3;
  readout_STAT5_abs := cyt*(STAT5c + pSTAT5c) + nuc*pSTAT5n;
  readout_pEpoR := 1000*EpoRJAK2_p;
  readout_pJAK2 := EpoRJAK2_act;
  readout_pSTAT5 := cyt*pSTAT5c + nuc*pSTAT5n;
  readout_pSTAT5_rel := (cyt*pSTAT5c + nuc*pSTAT5n)/(cyt*(STAT5c +
    pSTAT5c) + nuc*pSTAT5n);
  readout_tSHP1 := SHP1;
  readout_tSTAT5 := (cyt*(STAT5c + pSTAT5c) + nuc*pSTAT5n)/(cyt*
    init_STAT5);

  // Reactions:
  epor_act: EpoRJAK2 -> EpoRJAK2_act; EpoAct*EpoRJAK2;
  stat5_phos: STAT5c -> pSTAT5c; k_p*EpoRJAK2_p*STAT5c*socs3_inhib;
  stat5_import: pSTAT5c -> pSTAT5n; k_imp*pSTAT5c;
  stat5_nucout: pSTAT5n -> STAT5c; k_out*pSTAT5n;
  cisrna_syn:  -> CISRNA; k_deg_CISRNA*(1 + pSTAT5n)*ActD;
  cisrna_deg: CISRNA -> ; k_deg_CISRNA*CISRNA;
  cis_transl:  -> CIS; k_tr_CIS*CISRNA;
  socs3rna_syn:  -> SOCS3RNA; k_deg_S3RNA*(1 + pSTAT5n)*ActD;
```

```
socs3rna_deg: SOCS3RNA -> ; k_deg_S3RNA*SOCS3RNA;
socs3_transl:  -> SOCS3; k_tr_S3*SOCS3RNA;
socs3_deg: SOCS3 -> ; k_deg_S3*SOCS3;
jak2_inact: EpoRJAK2_act -> EpoRJAK2; (k_dSHP*SHP1 + k_dS3*SOCS3)*
  EpoRJAK2_act;
rp_form: EpoRJAK2_act -> EpoRJAK2_p; k_rp*EpoRJAK2_act;
rp_dephos: EpoRJAK2_p -> EpoRJAK2; k_dr*EpoRJAK2_p;
cis_bindp: EpoRJAK2_p + CIS -> EpoRJAK2_CIS; k_bCIS*CIS*EpoRJAK2_p;
cis_releasep: EpoRJAK2_CIS -> EpoRJAK2_p; k_CISrel*EpoRJAK2_CIS;

// Species initializations:
EpoRJAK2 = 1 - init_EpoRJAK2_CIS*CISfrac0;
EpoRJAK2_act = 0;
EpoRJAK2_CIS = init_EpoRJAK2_CIS*CISfrac0;
STAT5c = init_STAT5;
pSTAT5c = 0;
pSTAT5n = 0;
CISRNA = CISRNA_init;
CIS = init_CIS_multiplier*CISoe*CISEqcOE;
SOCS3RNA = SOCS3RNA_init;
SOCS3 = init_SOCS3_multiplier*SOCS3oe*SOCS3EqcOE;
SHP1 = init_SHP1*(1 + init_SHP1_multiplier*(SHP1ProOE - 1));
EpoRJAK2_p = ;

// Compartment initializations:
cyt = 0.4;
nuc = 0.275;

// Variable initializations:
init_EpoRJAK2_CIS = 0;
CISfrac0 = 0.5;
init_STAT5 = 200;
CISRNA_init = 1;
init_CIS_multiplier = 0;
CISoe = 0;
CISEqcOE = 20;
SOCS3RNA_init = 1;
init_SOCS3_multiplier = 0;
SOCS3oe = 0;
SOCS3EqcOE = 10;
init_SHP1 = 27;
init_SHP1_multiplier = 0;
SHP1ProOE = 3.75;
Epo = 1.25e-07;
ActD = 1;
KD_epo = 1e-08;
n_epo = 1.5;
k_dSHP = 0.001;
k_dS3 = 0.001;
k_bCIS = 0.001;
k_CISrel = 0.02;
k_p = 0.2;
Ki_socs3 = 3;
k_imp = 0.2;
k_out = 0.08;
k_deg_CISRNA = 0.02;
k_tr_CIS = 0.05;
k_deg_S3RNA = 0.02;
k_tr_S3 = 0.05;
k_deg_S3 = 0.005;
k_rp = ;
k_dr = ;

// Other declarations:
```

```
  var EpoAct, socs3_inhib, readout_CISRNA_fold, readout_SOCS3RNA_fold,
    readout_CIS_abs;
  var readout_CIS_au, readout_SHP1_abs, readout_SOCS3_abs,
    readout_SOCS3_au;
  var readout_STAT5_abs, readout_pEpoR, readout_pJAK2, readout_pSTAT5,
    readout_pSTAT5_rel;
  var readout_tSHP1, readout_tSTAT5;
  const cyt, nuc, init_EpoRJAK2_CIS, CISfrac0, init_STAT5, CISRNA_init,
     init_CIS_multiplier;
  const CISoe, CISEqcOE, SOCS3RNA_init, init_SOCS3_multiplier, SOCS3oe,
     SOCS3EqcOE;
  const init_SHP1, init_SHP1_multiplier, SHP1ProOE, Epo, ActD, KD_epo,
    n_epo;
  const k_dSHP, k_dS3, k_bCIS, k_CISrel, k_p, Ki_socs3, k_imp, k_out,
    k_deg_CISRNA;
  const k_tr_CIS, k_deg_S3RNA, k_tr_S3, k_deg_S3, k_rp, k_dr;
end
```

Listing 92: Archived Antimony for `bachmann_4`; task `56836792_8`; selected version 1

```
// Created by libAntimony v3.1.3
model *EpoJAK2STAT5_feedback()

  // Compartments and Species:
  compartment cyt, nuc;
  species EpoRJAK2 in cyt, J2 in cyt, R in cyt, R_so in cyt,
    EpoRJAK2_CIS in cyt;
  species STAT5 in cyt, pSTAT5 in cyt, pSTAT5_nuc in nuc, CISRNA in cyt
    , CIS in cyt;
  species SOCS3RNA in cyt, SOCS3 in cyt, SHP1 in cyt, SHP1_act in cyt,
    EpoRJAK2_seq;

  // Assignment Rules:
  readout_pJAK2 := J2;
  readout_pEpoR := R + R_so;
  readout_pSTAT5 := pSTAT5 + pSTAT5_nuc;
  readout_pSTAT5_rel := (pSTAT5*cyt + pSTAT5_nuc*nuc)/(STAT5*cyt +
    pSTAT5*cyt + pSTAT5_nuc*nuc);
  readout_tSTAT5 := STAT5*cyt + pSTAT5*cyt + pSTAT5_nuc*nuc;
  readout_STAT5_abs := STAT5*cyt + pSTAT5*cyt + pSTAT5_nuc*nuc;
  readout_tSHP1 := SHP1 + SHP1_act;
  readout_SHP1_abs := (SHP1 + SHP1_act)*cyt;
  readout_CIS_abs := CIS*cyt;
  readout_CIS_au := CIS + EpoRJAK2_CIS;
  readout_SOCS3_abs := SOCS3*cyt;
  readout_SOCS3_au := SOCS3;
  readout_CISRNA_fold := CISRNA - 1;
  readout_SOCS3RNA_fold := SOCS3RNA - 1;

  // Reactions:
  epor_activation_J2: EpoRJAK2 -> J2; kact*Epo*EpoRJAK2;
  epor_phosphorylation: J2 -> R; kab*J2;
  jak2_dephos_shp1: J2 -> EpoRJAK2; kj_shp1*SHP1_act*J2;
  jak2_dephos_socs3: J2 -> EpoRJAK2; kj_so*R_so*J2;
  epor_dephos_basal: R -> EpoRJAK2; kr0*R;
  cis_receptor_binding: R + CIS -> EpoRJAK2_CIS; kcis*CIS*R;
  socs3_receptor_docking: R + SOCS3 -> R_so; kdock*SOCS3*R;
  rsoc_release_dephos: R_so -> EpoRJAK2 + SOCS3; kr_so*R_so;
  stat5_phosphorylation: STAT5 -> pSTAT5; kp*R*STAT5;
  stat5_dephosphorylation: pSTAT5 -> STAT5; kdp*pSTAT5;
  stat5_import: pSTAT5 -> pSTAT5_nuc; kimp*pSTAT5*cyt;
  stat5_export: pSTAT5_nuc -> pSTAT5; kexp*pSTAT5_nuc*nuc;
  shp1_activation: SHP1 -> SHP1_act; ka_shp*SHP1*R;
  cis_transcription:  -> CISRNA; ktr_CIS*pSTAT5_nuc*ActD;
  cisrna_degradation: CISRNA -> ; kdeg_CISRNA*CISRNA;
```

```
cis_translation:  -> CIS; ktl_CIS*CISRNA;
cis_degradation: CIS -> ; kdeg_CIS*CIS;
socs3_transcription:  -> SOCS3RNA; ktr_SOCS3*pSTAT5_nuc*ActD;
socs3rna_degradation: SOCS3RNA -> ; kdeg_SOCS3RNA*SOCS3RNA;
socs3_translation:  -> SOCS3; ktl_SOCS3*SOCS3RNA;
socs3_degradation: SOCS3 -> ; kdeg_SOCS3*SOCS3;
receptor_sequestration_basal: R -> EpoRJAK2_seq; kseq*R;

// Species initializations:
EpoRJAK2 = init_EpoRJAK2 - init_EpoRJAK2_CIS*rCIS_eqc;
J2 = 0;
R = 0;
R_so = 0;
EpoRJAK2_CIS = init_EpoRJAK2_CIS*rCIS_eqc;
STAT5 = init_STAT5;
pSTAT5 = 0;
pSTAT5_nuc = 0;
CISRNA = CISRNA_init;
CIS = init_CIS_multiplier*CISoe*CISEqcOE;
SOCS3RNA = SOCS3RNA_init;
SOCS3 = init_SOCS3_multiplier*SOCS3oe*SOCS3EqcOE;
SHP1 = init_SHP1*(1 + init_SHP1_multiplier*(SHP1ProOE - 1));
SHP1_act = 0;
EpoRJAK2_seq = ;

// Compartment initializations:
cyt = 0.4;
nuc = 0.275;

// Variable initializations:
init_EpoRJAK2 = ;
init_EpoRJAK2_CIS = ;
rCIS_eqc = 150;
init_STAT5 = ;
CISRNA_init = 1;
init_CIS_multiplier = ;
CISoe = ;
CISEqcOE = 60;
SOCS3RNA_init = 1;
init_SOCS3_multiplier = ;
SOCS3oe = ;
SOCS3EqcOE = 40;
init_SHP1 = ;
init_SHP1_multiplier = ;
SHP1ProOE = 5;
kact = 15000000;
kab = 0.5;
kj_shp1 = 0.01;
kj_so = 0.05;
kr0 = 0.15;
kcis = 0.005;
kdock = 0.5;
kr_so = 0.01;
kp = 0.01;
kdp = 0.5;
kimp = 0.3;
kexp = 0.2;
ka_shp = 0.0005;
ktr_CIS = 0.1;
kdeg_CISRNA = 0.02;
ktl_CIS = 2;
kdeg_CIS = 0.01;
ktr_SOCS3 = 0.1;
kdeg_SOCS3RNA = 0.005;
ktl_SOCS3 = 2;
```

```
  kdeg_SOCS3 = 0.008;
  Epo = ;
  ActD = ;
  kseq = ;

  // Other declarations:
  var readout_pJAK2, readout_pEpoR, readout_pSTAT5, readout_pSTAT5_rel,
     readout_tSTAT5;
  var readout_STAT5_abs, readout_tSHP1, readout_SHP1_abs,
    readout_CIS_abs;
  var readout_CIS_au, readout_SOCS3_abs, readout_SOCS3_au,
    readout_CISRNA_fold;
  var readout_SOCS3RNA_fold;
  const cyt, nuc, init_EpoRJAK2, init_EpoRJAK2_CIS, rCIS_eqc,
    init_STAT5;
  const CISRNA_init, init_CIS_multiplier, CISoe, CISEqcOE,
    SOCS3RNA_init;
  const init_SOCS3_multiplier, SOCS3oe, SOCS3EqcOE, init_SHP1,
    init_SHP1_multiplier;
  const SHP1ProOE, kact, kab, kj_shp1, kj_so, kr0, kcis, kdock, kr_so,
    kp;
  const kdp, kimp, kexp, ka_shp, ktr_CIS, kdeg_CISRNA, ktl_CIS,
    kdeg_CIS;
  const ktr_SOCS3, kdeg_SOCS3RNA, ktl_SOCS3, kdeg_SOCS3, Epo, ActD,
    kseq;
end
```

Listing 93: Archived Antimony for `bachmann_5`; task `56836792_9`; selected version 16

```
// Created by libAntimony v3.1.3
model *epor_jak2_stat5_v2()

  // Compartments and Species:
  compartment cyt, nuc;
  species EpoRJAK2 in cyt, EpoRJAK2_act in cyt, EpoRJAK2_CIS in cyt, Jp
     in cyt;
  species STAT5 in cyt, pSTAT5 in cyt, pSTAT5_nuc in nuc, STAT5_nuc in
    nuc;
  species CISRNA in cyt, CIS in cyt, SOCS3RNA in cyt, SOCS3 in cyt,
    SHP1 in cyt;
  species EpoRJAK2_int;

  // Assignment Rules:
  readout_CISRNA_fold := CISRNA - 1;
  readout_SOCS3RNA_fold := SOCS3RNA - 1;
  readout_CIS_abs := CIS*cyt;
  readout_CIS_au := CIS*cyt;
  readout_SHP1_abs := SHP1*cyt;
  readout_tSHP1 := SHP1*cyt;
  readout_SOCS3_abs := SOCS3*cyt;
  readout_SOCS3_au := SOCS3*cyt;
  readout_STAT5_abs := STAT5*cyt + pSTAT5*cyt + STAT5_nuc*nuc +
    pSTAT5_nuc*nuc;
  readout_tSTAT5 := STAT5*cyt + pSTAT5*cyt + STAT5_nuc*nuc + pSTAT5_nuc
    *nuc;
  readout_pEpoR := EpoRJAK2_act;
  readout_pJAK2 := Jp;
  readout_pSTAT5 := pSTAT5*cyt + pSTAT5_nuc*nuc;
  readout_pSTAT5_rel := (pSTAT5*cyt + pSTAT5_nuc*nuc)/(init_STAT5*cyt);

  // Reactions:
  epor_activation: EpoRJAK2 -> EpoRJAK2_act; k_act*(Epo^n_Epo/(Kd_act^
    n_Epo + Epo^n_Epo))*EpoRJAK2;
  epor_deactivation: EpoRJAK2_act -> EpoRJAK2; k_deact_SHP1*SHP1*
    EpoRJAK2_act;
```

```
cis_binding: EpoRJAK2_act + CIS -> EpoRJAK2_CIS; k_bind_CIS*(CIS/(
  Ki_CISb + CIS))*EpoRJAK2_act;
cis_release: EpoRJAK2_CIS -> EpoRJAK2 + CIS; k_unbind_CIS*
  EpoRJAK2_CIS;
jak2_activation:  -> Jp; k_J_form*EpoRJAK2_act;
jak2_deact: Jp -> ; (k_J_SHP1*SHP1 + k_KIR_max*SOCS3/(Ki_J_SOCS3 +
  SOCS3))*Jp;
stat5_phos: STAT5 -> pSTAT5; k_phos*EpoRJAK2_act*STAT5/(1 + CIS/
  Ki_CIS + SOCS3/Ki_SOCS3);
stat5_import: pSTAT5 -> pSTAT5_nuc; k_imp*pSTAT5;
stat5_nuc_dephos: pSTAT5_nuc -> STAT5_nuc; k_deg_nuc*pSTAT5_nuc;
stat5_export: STAT5_nuc -> STAT5; k_exp*STAT5_nuc;
cis_rna_syn:  -> CISRNA; ActD*(k_deg_CISRNA + k_syn_CIS*pSTAT5_nuc);
cis_rna_deg: CISRNA -> ; k_deg_CISRNA*CISRNA;
cis_transl:  -> CIS; k_syn_CIS*CISRNA;
cis_deg: CIS -> ; k_deg_CIS*CIS;
socs3_rna_syn:  -> SOCS3RNA; ActD*(k_deg_SOCS3RNA + k_syn_SOCS3*
  pSTAT5_nuc);
socs3_rna_deg: SOCS3RNA -> ; k_deg_SOCS3RNA*SOCS3RNA;
socs3_transl:  -> SOCS3; k_syn_SOCS3*SOCS3RNA;
socs3_deg: SOCS3 -> ; k_deg_SOCS3*SOCS3;
epor_degradation_SOCS3: EpoRJAK2_act -> ; k_so3_deg*(SOCS3/(
  Ki_J_SOCS3 + SOCS3))*EpoRJAK2_act;
epor_internalization: EpoRJAK2_act -> EpoRJAK2_int; k_int*
  EpoRJAK2_act;
epor_recycling: EpoRJAK2_int -> EpoRJAK2; k_recyc*EpoRJAK2_int;

// Species initializations:
EpoRJAK2 = init_EpoRJAK2 - init_EpoRJAK2_CIS*fracR_pre*init_EpoRJAK2;
EpoRJAK2_act = 0;
EpoRJAK2_CIS = init_EpoRJAK2_CIS*fracR_pre*init_EpoRJAK2;
Jp = 0;
STAT5 = init_STAT5;
pSTAT5 = 0;
pSTAT5_nuc = 0;
STAT5_nuc = 0;
CISRNA = CISRNA_init;
CIS = CISoe*init_CIS_multiplier*CISeqcOE;
SOCS3RNA = SOCS3RNA_init;
SOCS3 = SOCS3oe*init_SOCS3_multiplier*SOCS3EqcOE;
SHP1 = init_SHP1*(1 + init_SHP1_multiplier*SHP1ProOE);
EpoRJAK2_int = ;

// Compartment initializations:
cyt = 0.4;
nuc = 0.275;

// Variable initializations:
init_EpoRJAK2 = 0.5;
init_EpoRJAK2_CIS = ;
fracR_pre = 0.1;
init_STAT5 = 200;
CISRNA_init = 1;
CISoe = ;
init_CIS_multiplier = ;
CISeqcOE = 400;
SOCS3RNA_init = 1;
SOCS3oe = ;
init_SOCS3_multiplier = ;
SOCS3EqcOE = 500;
init_SHP1 = 60;
init_SHP1_multiplier = ;
SHP1ProOE = 3;
k_act = 0.1;
Kd_act = 5e-08;
```

```
  n_Epo = 1.5;
  k_deact_SHP1 = 0.0002;
  k_bind_CIS = 0.05;
  Ki_CISb = 10;
  k_unbind_CIS = 0.03;
  k_J_form = 0.1;
  k_J_SHP1 = 0.002;
  k_KIR_max = 0.2;
  Ki_J_SOCS3 = 150;
  k_phos = 1;
  Ki_CIS = 240;
  Ki_SOCS3 = 15;
  k_imp = 0.03;
  k_exp = 0.01;
  k_deg_nuc = 0.02;
  k_deg_CISRNA = 0.005;
  k_syn_CIS = 0.0002;
  k_deg_CIS = 0.002;
  k_deg_SOCS3RNA = 0.004;
  k_syn_SOCS3 = 0.09;
  k_deg_SOCS3 = 0.003;
  Epo = ;
  ActD = ;
  k_so3_deg = ;
  k_int = ;
  k_recyc = ;

  // Other declarations:
  var readout_CISRNA_fold, readout_SOCS3RNA_fold, readout_CIS_abs,
    readout_CIS_au;
  var readout_SHP1_abs, readout_tSHP1, readout_SOCS3_abs,
    readout_SOCS3_au;
  var readout_STAT5_abs, readout_tSTAT5, readout_pEpoR, readout_pJAK2,
    readout_pSTAT5;
  var readout_pSTAT5_rel;
  const cyt, nuc, init_EpoRJAK2, init_EpoRJAK2_CIS, fracR_pre,
    init_STAT5;
  const CISRNA_init, CISoe, init_CIS_multiplier, CISeqcOE,
    SOCS3RNA_init;
  const SOCS3oe, init_SOCS3_multiplier, SOCS3EqcOE, init_SHP1,
    init_SHP1_multiplier;
  const SHP1ProOE, k_act, Kd_act, n_Epo, k_deact_SHP1, k_bind_CIS,
    Ki_CISb;
  const k_unbind_CIS, k_J_form, k_J_SHP1, k_KIR_max, Ki_J_SOCS3, k_phos
    , Ki_CIS;
  const Ki_SOCS3, k_imp, k_exp, k_deg_nuc, k_deg_CISRNA, k_syn_CIS,
    k_deg_CIS;
  const k_deg_SOCS3RNA, k_syn_SOCS3, k_deg_SOCS3, Epo, ActD, k_so3_deg,
     k_int;
  const k_recyc;
end
```

H.14 RAIMUNDEZ

Listing 94: Archived Antimony for `raimundez_1`; task `56836792_15`; selected version 28

```
// Created by libAntimony v3.1.3
model *egfr_met_v2b()

  // Compartments and Species:
  compartment cell;
  species EGF_level in cell, CET_level in cell, MKN1 in cell, HS746T in
     cell;
  species full_medium in cell, hunger_medium in cell, METinh_level in
    cell;
  species relative_ksyn_EGFR in cell, E_s in cell, E_p in cell, E_i in
    cell;
  species E_d in cell, Mek_p in cell, A_p in cell, M_p in cell, MET_T
    in cell;
  species E_b, Raf_p;

  // Assignment Rules:
  EGF_eff := EGF_level*piecewise(0, time < 0, 1);
  CET_eff := CET_level*piecewise(0, time < 0, 1);
  Lser := L_bas*(MKN1 + a_basal*HS746T)*(full_medium + rH*hunger_medium
    );
  L_eff := (EGF_eff + Lser)/(1 + CET_eff/K_CET);
  metd := mM*HS746T*(1 - METinh_level);
  pEg := E_p + E_i;
  Ttot := E_s + E_b + E_p + E_i + E_d;
  pEgF := pEg/Ttot;
  satA := pEgF/(KAsA + pEgF);
  satM := pEgF/(KAsM + pEgF);
  readout_tEGFR := Ttot;
  readout_pEGFR := pEg;
  readout_tMET := MET_T;
  readout_pAKT := A_p;
  readout_pMAPK := M_p;

  // Reactions:
  egfr_syn:  -> E_s; 0.001*relative_ksyn_EGFR;
  egfr_act: E_s => E_p; E_s*a_basal*(MKN1 + rab*HS746T);
  egfr_int: E_p => E_i; k_tr*E_p;
  egfr_rec: E_i => E_s; k_tr*E_i;
  egfr_ub: E_i => E_d; k_ub*(MKN1 + rMe*HS746T)*E_i;
  egfr_deg: E_d => ; k_deg*E_d;
  mek_act:  -> Mek_p; Raf_p;
  mek_dep: Mek_p => ; k_dM*Mek_p;
  mapk_phos:  -> M_p; Mek_p;
  mapk_dep: M_p => ; k_dM*(1 + f_dusp*M_p)*M_p;
  akt_phos:  -> A_p; satA + metd;
  akt_dep: A_p => ; kA2*A_p;
  egfr_bind: E_s => E_b; a_E*L_eff*E_s;
  egfr_actL: E_b => E_p; a_E*E_b/(1 + f_eEb*M_p);
  raf_act:  -> Raf_p; satM*(MKN1 + rMe*HS746T) + metd;
  raf_dep: Raf_p => ; k_dR*Raf_p;

  // Species initializations:
  EGF_level = 0;
  CET_level = 0;
  MKN1 = 0;
  HS746T = 0;
  full_medium = 0;
  hunger_medium = 0;
  METinh_level = 0;
  relative_ksyn_EGFR = 1;
  E_s = E_s_init;
```

```
  E_p = 0;
  E_i = 0;
  E_d = 0;
  Mek_p = 0;
  A_p = 0;
  M_p = 0;
  MET_T = MET_T_init;
  E_b = ;
  Raf_p = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  E_s_init = 1;
  MET_T_init = 1;
  a_basal = 0.0062;
  rab = 29;
  a_E = 0.51;
  L_bas = 0.0147;
  rH = 0.5;
  K_CET = 0.0216;
  k_tr = 0.51;
  k_ub = 0.01;
  k_deg = 0.0058;
  KAsA = 0.045;
  KAsM = 0.21;
  kA2 = 0.41;
  k_dM = 0.92;
  rMe = 1.21;
  mM = 0.25;
  f_dusp = ;
  f_eEb = ;
  k_dR = ;

  // Other declarations:
  var EGF_eff, CET_eff, Lser, L_eff, metd, pEg, Ttot, pEgF, satA, satM,
     readout_tEGFR;
  var readout_pEGFR, readout_tMET, readout_pAKT, readout_pMAPK;
  const cell, E_s_init, MET_T_init, a_basal, rab, a_E, L_bas, rH, K_CET
    , k_tr;
  const k_ub, k_deg, KAsA, KAsM, kA2, k_dM, rMe, mM, f_dusp, f_eEb,
    k_dR;
end
```

Listing 95: Archived Antimony for `raimundez_2`; task `56836792_16`; selected version 17

```
// Created by libAntimony v3.1.3
model *EGFR_MET_network()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, LE in cell, LEp in cell, REC in cell, MET in
    cell;
  species METp in cell, pAKT in cell, pMAPK in cell, IEP, RECp, RAFp,
    RASp;

  // Assignment Rules:
  EGF := EGF_level*piecewise(0, time < 0, 1);
  CET := CET_level*piecewise(0, time < 0, 1);
  readout_tEGFR := EGFR + LEp + IEP + REC;
  readout_pEGFR := LEp + IEP + RECp;
  readout_tMET := MET + METp;
  readout_pAKT := pAKT;
  readout_pMAPK := pMAPK;
```

```

// Reactions:
EGF_bind: EGFR -> LE; kon_E*EGF*EGFR;
CET_bind: EGFR -> REC; kon_C*CET*EGFR;
LE_phos: LE -> LEp; k_pLE*LE;
LE_dephos: LEp -> LE; k_dpLE*LEp;
EGFR_deg: EGFR -> ; kdeg_R*EGFR;
EGFR_syn:  -> EGFR; ksyn_E*relative_ksyn_EGFR;
MET_syn:  -> MET; ksyn_M;
MET_deg: MET -> ; kdeg_M*MET;
MET_act: MET -> METp; (aM_HS*HS746T + aM_MKN1*MKN1)*MET;
MET_dephos: METp -> MET; k_dpMET*METp;
AKT_act:  -> pAKT; kAkt_E*LEp + kAkt_M*METp;
AKT_dp: pAKT -> ; kdpAkt*pAKT;
MAPK_act:  -> pMAPK; kMpk*RAFp;
MAPK_dp: pMAPK -> ; kdpMpk*pMAPK;
LE_dissoc: LE -> EGFR; k_diss*LE;
LEp_int: LEp -> IEP; kint_max*piecewise(0, EGF < 0.5, 1)*LEp;
IEP_rec: IEP -> EGFR; krecyc*IEP;
IEP_deg: IEP -> ; kdeg_i*IEP;
basal_bind: EGFR -> LE; (k_basFM*full_medium + k_basHM*hunger_medium)
  *EGFR;
transact: EGFR -> LEp; kTA_HS*HS746T*METp*EGFR;
RAF_act:  -> RAFp; kRas_Raf*RASp;
RAF_dp: RAFp -> ; kdpRaf*RAFp;
RAF_fb: RAFp -> ; kfb*pMAPK*RAFp;
REC_unbind: REC -> EGFR; koff_C*REC;
transact_C: REC -> RECp; kTA_HS*HS746T*METp*REC;
RECp_dp: RECp -> REC; k_dpLE*RECp;
RAS_act:  -> RASp; kMpk_E*(LEp + IEP) + kMpk_M*METp;
RAS_dp: RASp -> ; kRas_gap*RASp;
IEP_rec2: IEP -> EGFR; krecyc2*IEP;

// Species initializations:
EGFR = EGFR_init;
LE = 0;
LEp = 0;
REC = 0;
MET = MET_init;
METp = 0;
pAKT = 0;
pMAPK = 0;
IEP = ;
RECp = ;
RAFp = ;
RASp = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
EGFR_init = 1;
MET_init = 1;
EGF_level = 0;
CET_level = 0;
METinh_level = 0;
relative_ksyn_EGFR = 1;
full_medium = 1;
hunger_medium = 0;
MKN1 = 0;
HS746T = 0;
kon_E = 0.02;
kon_C = 0.05;
k_basFM = 0.002;
k_basHM = 0.0002;
```

```
  k_pLE = 2;
  k_dpLE = 0.15;
  kdeg_R = 0.0008;
  ksyn_E = 0.006;
  aM_MKN1 = 0.002;
  aM_HS = 0.02;
  k_dpMET = 0.2;
  ksyn_M = 0.002;
  kdeg_M = 0.0004;
  kAkt_E = 0.5;
  kAkt_M = 0.3;
  kdpAkt = 0.01;
  kMpk_E = 0.5;
  kMpk_M = 0.3;
  kdpMpk = 0.02;
  kMpk = ;
  k_diss = ;
  kint_max = ;
  krecyc = ;
  kdeg_i = ;
  kTA_HS = ;
  kRas_Raf = ;
  kdpRaf = ;
  kfb = ;
  koff_C = ;
  kRas_gap = ;
  krecyc2 = ;

  // Other declarations:
  var EGF, CET, readout_tEGFR, readout_pEGFR, readout_tMET,
    readout_pAKT;
  var readout_pMAPK;
  const cell, EGFR_init, MET_init, EGF_level, CET_level, METinh_level,
    relative_ksyn_EGFR;
  const full_medium, hunger_medium, MKN1, HS746T, kon_E, kon_C, k_basFM
    , k_basHM;
  const k_pLE, k_dpLE, kdeg_R, ksyn_E, aM_MKN1, aM_HS, k_dpMET, ksyn_M,
     kdeg_M;
  const kAkt_E, kAkt_M, kdpAkt, kMpk_E, kMpk_M, kdpMpk, kMpk, k_diss,
    kint_max;
  const krecyc, kdeg_i, kTA_HS, kRas_Raf, kdpRaf, kfb, koff_C, kRas_gap
    , krecyc2;
end
```

Listing 96: Archived Antimony for `raimundez_3`; task `56836792_17`; selected version 19

```
// Created by libAntimony v3.1.3
model *EGFR_MET_network()

  // Compartments and Species:
  compartment cell;
  species EGF_level in cell, CET_level in cell, HS746T in cell, MKN1 in
     cell;
  species full_medium in cell, hunger_medium in cell, METinh_level in
    cell;
  species relative_ksyn_EGFR in cell, EGFR in cell, EGFb in cell, pEGFR
     in cell;
  species X in cell, pMAPK in cell, pAKT in cell, MET in cell, M1, Ub,
    A1;

  // Assignment Rules:
  EGF := EGF_level*piecewise(0, time < 0, 1);
  CET := CET_level*piecewise(0, time < 0, 1);
  block := kxdM/(kxdM + CET);
  cetsat := CET/(kxdM + CET);
```

```
kin_basal := kbasMF*MKN1*full_medium + ksyn*MKN1*hunger_medium;
kin := kin_basal*block + kma*HS746T;
kxd := kxdM*MKN1 + kxdH*HS746T;
kCETint := kactL*MKN1;
kmetX := (ksyn*MKN1 + kdegU*HS746T)*(1 - METinh_level);
kint := kintM*MKN1 + dA1*HS746T;
readout_tEGFR := EGFR + EGFb + pEGFR + Ub;
readout_pEGFR := pEGFR;
readout_pMAPK := pMAPK + M1;
readout_pAKT := pAKT + A1;
readout_tMET := MET;

// Reactions:
egfr_syn:  -> EGFR; ksyn*relative_ksyn_EGFR*(1 + fsynL*EGFb);
egfr_bind: EGFR -> EGFb; kbind*EGF*block*EGFR;
egfr_unbind: EGFb -> EGFR; kfa*EGFb;
egfr_actL: EGFb -> pEGFR; kactL*EGFb;
egfr_act: EGFR -> pEGFR; kin*EGFR;
egfr_deact: pEGFR -> EGFb; kxa*pEGFR;
egfr_cet: EGFR -> ; kCETint*cetsat*EGFR;
egfr_cetP: pEGFR -> ; kCETint*cetsat*pEGFR;
x_act: pEGFR -> pEGFR + X; kxa*pEGFR;
x_met:  -> X; kmetX;
x_ser:  -> X; ksyn*full_medium;
x_dep: X -> ; kxd*X;
mapk_act: X -> X + pMAPK; kma*X;
mapk_dep: pMAPK -> ; kxdM*pMAPK;
akt_direct: pEGFR -> pEGFR + pAKT; kxa*pEGFR;
akt_via_x: X -> X + pAKT; kax*X;
akt_dep: pAKT -> ; dA1*pAKT;
m1_form: pEGFR -> pEGFR + M1; kfm*pEGFR;
m1_dephos: M1 -> ; kbind*M1;
egfr_ub: EGFR -> Ub; kint*pEGFR;
egfr_ubdeg: Ub -> ; kdegU*Ub;
akt_fast: pEGFR -> pEGFR + A1; kfa*pEGFR;
akt_fastout: A1 -> ; dA1*A1;

// Species initializations:
EGF_level = 0;
CET_level = 0;
HS746T = 0;
MKN1 = 0;
full_medium = 0;
hunger_medium = 0;
METinh_level = 0;
relative_ksyn_EGFR = 0;
EGFR = 0;
EGFb = 0;
pEGFR = 0;
X = 0;
pMAPK = 0;
pAKT = 0;
MET = MET_init;
M1 = ;
Ub = ;
A1 = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
MET_init = 1;
kxdM = 0.319;
kbasMF = 0.01274;
ksyn = 1.35;
```

```
  kma = 0.5;
  kxdH = 0.889;
  kactL = 0.01235;
  kdegU = 4.157;
  fsynL = 12.414;
  kbind = 1.004;
  kfa = 0.469;
  kxa = 14.42;
  kax = 20;
  dA1 = 0.0951;
  kintM = ;
  kfm = ;

  // Other declarations:
  var EGF, CET, block, cetsat, kin_basal, kin, kxd, kCETint, kmetX,
    kint;
  var readout_tEGFR, readout_pEGFR, readout_pMAPK, readout_pAKT,
    readout_tMET;
  const cell, MET_init, kxdM, kbasMF, ksyn, kma, kxdH, kactL, kdegU,
    fsynL;
  const kbind, kfa, kxa, kax, dA1, kintM, kfm;
end
```

Listing 97: Archived Antimony for `raimundez_4`; task `56836792_18`; selected version 17

```
// Created by libAntimony v3.1.3
model *egfr_met_akt_mapk_v3()

  // Compartments and Species:
  compartment cell;
  species E in cell, Eb in cell, Ec in cell, Ei in cell, P in cell, Met
     in cell;
  species pA in cell, $pMf in cell, pMs in cell, pAs;

  // Assignment Rules:
  pMf := kaM*P^hM;
  EGF_in := EGF_level*piecewise(0, time < 0, 1);
  CET_in := CET_level*piecewise(0, time < 0, 1);
  Lig := EGF_in + lig_serum*full_medium*(1 - hunger_medium);
  metC := (mHS + mHSF*full_medium)*HS746T + mMK*MKN1;
  ksyn := kdeg*relative_ksyn_EGFR;
  readout_tEGFR := E + Eb + Ec + Ei + P;
  readout_pEGFR := P;
  readout_pAKT := pA + pAs;
  readout_pMAPK := pMf + pMs;
  readout_tMET := metC;

  // Reactions:
  egfr_syn:  -> E; ksyn;
  egfr_deg: E -> ; kdeg*E;
  lig_bind: E -> Eb; konL*Lig*E;
  cet_bind: E -> Ec; kaM*CET_in*E;
  phos_lig: Eb -> P; kP*Eb;
  phos_met: E -> P; kdA*Met*(E + Eb + Ec);
  egfr_dephos: P -> E; ktrans*P;
  p_int: P -> Ei; ktrans*P;
  eb_int: Eb -> Ei; ktrans*Eb;
  ec_int: Ec -> Ei; ktrans*Ec;
  ei_rec: Ei -> E; kaA*Ei;
  ei_deg: Ei -> ; kdegR*Ei;
  met_act:  -> Met; (ktrans*P + metC)*(1 - METinh_level);
  met_deact: Met -> ; kmetd*Met;
  akt_act:  -> pA; kaA*(P + kdeg*Met);
  akt_dephos: pA -> ; kdA*pA;
  mapk_slow:  -> pMs; kdeg*Met;
```

```
  mapk_slow_de: pMs -> ; kmetd*pMs;
  akt_slow:  -> pAs; kdeg*Met;
  akt_slow_de: pAs -> ; lig_serum*pAs;

  // Species initializations:
  E = ;
  Eb = ;
  Ec = ;
  Ei = ;
  P = ;
  Met = ;
  pA = ;
  pMs = ;
  pAs = ;

  // Compartment initializations:
  cell = 1;

  // Variable initializations:
  kaM = ;
  hM = ;
  EGF_level = ;
  CET_level = ;
  lig_serum = ;
  full_medium = ;
  hunger_medium = ;
  mHS = ;
  mHSF = ;
  HS746T = ;
  mMK = ;
  MKN1 = ;
  kdeg = ;
  relative_ksyn_EGFR = ;
  konL = ;
  kP = ;
  kdA = ;
  ktrans = ;
  kaA = ;
  kdegR = ;
  METinh_level = ;
  kmetd = ;

  // Other declarations:
  var EGF_in, CET_in, Lig, metC, ksyn, readout_tEGFR, readout_pEGFR,
    readout_pAKT;
  var readout_pMAPK, readout_tMET;
  const cell, kaM, hM, EGF_level, CET_level, lig_serum, full_medium,
    hunger_medium;
  const mHS, mHSF, HS746T, mMK, MKN1, kdeg, relative_ksyn_EGFR, konL,
    kP;
  const kdA, ktrans, kaA, kdegR, METinh_level, kmetd;
end
```

Listing 98: Archived Antimony for `raimundez_5`; task `56836792_19`; selected version 27

```
// Created by libAntimony v3.1.3
model *EGFR_MET_gastric()

  // Compartments and Species:
  compartment cell;
  species EGFR in cell, pEGFR in cell, DES in cell, pAKT in cell, IM in
     cell;
  species pERK in cell, Act, Cocc, IM2;

  // Assignment Rules:
```

```
EGF := EGF_level*piecewise(0, time < 0, 1);
CET := CET_level*piecewise(0, time < 0, 1);
MET := 1;
mact := (1 - METinh_level)*MET;
mdrvA := mact*(MKN1 + qA_HS*HS746T);
mdrvM := mact*(MKN1 + qM_HS*HS746T);
tadrive := mact*(MKN1 + taHS*HS746T);
readout_tEGFR := EGFR + pEGFR + DES;
readout_pEGFR := pEGFR;
readout_tMET := MET;
readout_pAKT := pAKT;
readout_pMAPK := pERK;

// Reactions:
r_syn_EGFR:  -> EGFR; ksyn*relative_ksyn_EGFR*(MKN1 + qA_HS*HS746T);
r_deg_EGFR: EGFR -> ; kdeg*EGFR;
r_act_EGF: EGFR -> pEGFR; kact*Act*EGFR;
r_act_tonic: EGFR -> pEGFR; ktonic*(full_medium + rHM*hunger_medium)
  *(1 - Cocc)*EGFR;
r_act_TA: EGFR -> pEGFR; kon*tadrive*EGFR;
r_dephos: pEGFR -> DES; kdephos*pEGFR;
r_recycle: DES -> EGFR; kproc*DES;
r_deg_DES: DES -> ; kdegD*DES;
r_act_AKT:  -> pAKT; qM_HS*pEGFR + kA_MET*mdrvA;
r_deg_AKT: pAKT -> ; kdA*pAKT;
r_act_IM:  -> IM; kA_MET*mdrvM + kM_pE*pEGFR;
r_deg_IM: IM -> ; kdecl*(MKN1 + mdHS*HS746T)*IM;
r_act_ERK:  -> pERK; kdA*IM + ktr*IM2;
r_deg_ERK: pERK -> ; kdecl*pERK;
r_int_EGF: pEGFR -> DES; kintE*(MKN1 + intHS*HS746T)*Act*EGFR;
r_engage:  -> Act; kon*(1 - Cocc)*EGF*(1 - Act);
r_disengage: Act -> ; koff*Act;
r_act_IM2:  -> IM2; kproc*DES;
r_deg_IM2: IM2 -> ; kdIM2*IM2;
r_cet_bind:  -> Cocc; konC*(1 - Cocc)*CET;
r_cet_off: Cocc -> ; koffC*Cocc;

// Species initializations:
EGFR = EGFR_init;
pEGFR = 0;
DES = 0;
pAKT = 0;
IM = 0;
pERK = 0;
Act = ;
Cocc = ;
IM2 = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
EGFR_init = 1;
EGF_level = 0;
CET_level = 0;
MKN1 = 0;
HS746T = 0;
full_medium = 0;
hunger_medium = 0;
METinh_level = 0;
relative_ksyn_EGFR = 1;
qA_HS = 1;
qM_HS = 1;
taHS = 1;
ksyn = 0.01;
```

```
  kdeg = 0.001;
  kact = 0.5;
  ktonic = 0.005;
  rHM = 1;
  kdephos = 0.05;
  kproc = 0.005;
  kdegD = 0.002;
  kA_MET = 0.01;
  kdA = 0.05;
  kM_pE = 0.1;
  kdecl = 0.05;
  mdHS = 1;
  kon = ;
  ktr = ;
  kintE = ;
  intHS = ;
  koff = ;
  kdIM2 = ;
  konC = ;
  koffC = ;

  // Other declarations:
  var EGF, CET, MET, mact, mdrvA, mdrvM, tadrive, readout_tEGFR,
    readout_pEGFR;
  var readout_tMET, readout_pAKT, readout_pMAPK;
  const cell, EGFR_init, EGF_level, CET_level, MKN1, HS746T,
    full_medium;
  const hunger_medium, METinh_level, relative_ksyn_EGFR, qA_HS, qM_HS,
    taHS;
  const ksyn, kdeg, kact, ktonic, rHM, kdephos, kproc, kdegD, kA_MET,
    kdA;
  const kM_pE, kdecl, mdHS, kon, ktr, kintE, intHS, koff, kdIM2, konC,
    koffC;
end
```

H.15 ISENSEE

Listing 99: Archived Antimony for `isensee_1`; task `56836711_3`; selected version 2

```
// Created by libAntimony v3.1.3
model *PKA_RII_holoenzyme()

  // Compartments and Species:
  compartment cell;
  species $cRel in cell, H_u in cell, H_p in cell, F_u in cell, F_p in
    cell;
  species C in cell, Sp_in in cell, RpBr_in in cell, RpPT_in in cell,
    RpC_in in cell;

  // Assignment Rules:
  cRel := (a_bas + a_fsk*Fsk/(KaF + Fsk))/(1 - inhIBMX);
  Fsk := Fsk_level*piecewise(0, time < Fsk_time, 1);
  inhIBMX := IBMX/(Ki_IBMX + IBMX);
  IBMX := IBMX_level*piecewise(0, time < IBMX_time, 1);
  H89 := H89_level*piecewise(0, time < H89_time, 1);
  Sp8 := Sp8_Br_cAMPS_AM_level*piecewise(0, time < Sp8_Br_cAMPS_AM_time
    , 1);
  RpBr := Rp8_Br_cAMPS_pAB_level*piecewise(0, time <
    Rp8_Br_cAMPS_pAB_incubation_time, 1);
  RpPT := Rp8_pCPT_cAMPS_pAB_level*piecewise(0, time <
    Rp8_pCPT_cAMPS_pAB_incubation_time, 1);
  RpC := Rp_cAMPS_pAB_level*piecewise(0, time <
    Rp_cAMPS_pAB_incubation_time, 1);
```

```
ABnOH := fourABnOH_level*piecewise(0, time <
  fourABnOH_incubation_time, 1);
inhH89 := H89/(Ki_H89 + H89);
kinAct := 1 - inhH89;
act := (cRel^nHill + (Sp_in/Ka_Sp)^nHill)/(1 + cRel^nHill + (Sp_in/
  Ka_Sp)^nHill + (RpBr_in/Ka_Br)^nHill + (RpPT_in/Ka_PT)^nHill + (
  RpC_in/Ka_C)^nHill);
readout_pRII := F_p;
readout_pRII_Western := F_p + H_p;
readout_Calpha := C;

// Reactions:
R_load_Sp:  -> Sp_in; kload_Sp*(Sp8 - Sp_in);
R_load_Br:  -> RpBr_in; kload_Br*(RpBr - RpBr_in);
R_load_PT:  -> RpPT_in; kload_C*(RpPT - RpPT_in);
R_load_C:  -> RpC_in; kload_C*(RpC - RpC_in);
R_diss_u: H_u -> F_u + 2 C; kdis_u*H_u*act;
R_diss_p: H_p -> F_p + 2 C; kdis_p*H_p*act;
R_reass_u: F_u + 2 C -> H_u; kon_u*F_u*C*C;
R_auto: H_u -> H_p; kauto*H_u*kinAct;
R_phosF: F_u -> F_p; kphF*C*F_u*kinAct;
R_phos_other: F_u -> F_p; kOther*F_u;
R_dephos: F_p -> F_u; kdeP*F_p;

// Species initializations:
H_u = H_u_init;
H_p = 0;
F_u = 0;
F_p = 0;
C = 0;
Sp_in = 0;
RpBr_in = 0;
RpPT_in = 0;
RpC_in = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
a_bas = 0.02;
a_fsk = 1;
KaF = ;
H_u_init = 1;
Ki_IBMX = 100;
nHill = 2;
Ka_Sp = 3;
Ka_Br = 2;
Ka_PT = 2;
Ka_C = 10;
kload_Sp = 2;
kload_Br = 0.3;
kload_C = 0.5;
kdis_u = 1;
kdis_p = 0.5;
kon_u = 1;
kauto = 0.02;
kphF = 0.5;
kdeP = 0.04;
kOther = 0.002;
Ki_H89 = 1;
Fsk_level = ;
Fsk_time = ;
IBMX_level = ;
IBMX_time = ;
H89_level = ;
```

```
  H89_time = ;
  Sp8_Br_cAMPS_AM_level = ;
  Sp8_Br_cAMPS_AM_time = ;
  Rp8_Br_cAMPS_pAB_level = ;
  Rp8_Br_cAMPS_pAB_incubation_time = ;
  Rp8_pCPT_cAMPS_pAB_level = ;
  Rp8_pCPT_cAMPS_pAB_incubation_time = ;
  Rp_cAMPS_pAB_level = ;
  Rp_cAMPS_pAB_incubation_time = ;
  fourABnOH_level = ;
  fourABnOH_incubation_time = ;

  // Other declarations:
  var Fsk, inhIBMX, IBMX, H89, Sp8, RpBr, RpPT, RpC, ABnOH, inhH89,
    kinAct;
  var act, readout_pRII, readout_pRII_Western, readout_Calpha;
  const cell, a_bas, a_fsk, KaF, H_u_init, Ki_IBMX, nHill, Ka_Sp, Ka_Br
    , Ka_PT;
  const Ka_C, kload_Sp, kload_Br, kload_C, kdis_u, kdis_p, kon_u, kauto
    , kphF;
  const kdeP, kOther, Ki_H89, Fsk_level, Fsk_time, IBMX_level,
    IBMX_time;
  const H89_level, H89_time, Sp8_Br_cAMPS_AM_level,
    Sp8_Br_cAMPS_AM_time;
  const Rp8_Br_cAMPS_pAB_level, Rp8_Br_cAMPS_pAB_incubation_time,
    Rp8_pCPT_cAMPS_pAB_level;
  const Rp8_pCPT_cAMPS_pAB_incubation_time, Rp_cAMPS_pAB_level,
    Rp_cAMPS_pAB_incubation_time;
  const fourABnOH_level, fourABnOH_incubation_time;
end
```

Listing 100: Archived Antimony for `isensee_2`; task `56836711_1`; selected version 17

```
// Created by libAntimony v3.1.3
model *PKA_RII_holoenzyme()

  // Compartments and Species:
  compartment cell;
  species cAMP in cell, S_sp8 in cell, I_8Br in cell, I_8pCPT in cell,
    I_Rp in cell;
  species P_RII in cell, P_w, X_8Br, X_8pCPT, X_Rp;

  // Assignment Rules:
  Fsk := Fsk_level*piecewise(0, time < Fsk_time, 1);
  IBMX := IBMX_level*piecewise(0, time < IBMX_time, 1);
  H89 := H89_level*piecewise(0, time < H89_time, 1);
  Sp8 := Sp8_Br_cAMPS_AM_level*piecewise(0, time < Sp8_Br_cAMPS_AM_time
    , 1);
  Rp8Br := Rp8_Br_cAMPS_pAB_level*piecewise(0, time <
    Rp8_Br_cAMPS_pAB_incubation_time, 1);
  Rp8pCPT := Rp8_pCPT_cAMPS_pAB_level*piecewise(0, time <
    Rp8_pCPT_cAMPS_pAB_incubation_time, 1);
  Rp := Rp_cAMPS_pAB_level*piecewise(0, time <
    Rp_cAMPS_pAB_incubation_time, 1);
  fourABnOH := fourABnOH_level*piecewise(0, time <
    fourABnOH_incubation_time, 1);
  cep := cAMP + w_sp*S_sp8;
  C_free := cep^n_act/(cep^n_act + (K_act*(1 + I_8Br/Ki_8Br + I_8pCPT/
    Ki_8pCPT + I_Rp/K_act))^n_act);
  readout_pRII := P_RII + m_w*P_w;
  readout_pRII_Western := P_w + b_pRII_Western;
  readout_Calpha := C_free + theta_vis*(1 - C_free);
  b_pRII_global := b_Calpha_global;

  // Reactions:
```

```
camp_basal_prod:  -> cAMP; k_pde*f_basal;
camp_fsk_prod:  -> cAMP; k_pde*(1 - f_basal)*f_fsk*Fsk/(K_fsk + Fsk);
camp_pde_deg: cAMP -> ; k_pde*cAMP/((Km_pde + cAMP)*(1 + i_ibmx*IBMX)
  );
sp8_equil:  -> S_sp8; k_out_sp*(Sp8 - S_sp8);
rii_phos:  -> P_RII; (k_phos*C_free/(1 + H89/Ki_H89) + k_phos_w)*(1 -
   P_RII);
rii_dephos: P_RII -> ; k_dephos*P_RII;
pw_phos:  -> P_w; (k_phos_w*(cAMP^n_act/(cAMP^n_act + (K_act*(1 +
  I_8Br/Ki_8Br + I_8pCPT/Ki_8pCPT + I_Rp/K_act))^n_act))/(1 + H89/
  Ki_H89))*(1 - P_w);
pw_dephos: P_w -> ; k_dephos_w*P_w;
rp8br_entry:  -> X_8Br; k_in_rp*(Rp8Br - X_8Br - I_8Br);
rp8br_cleave: X_8Br -> I_8Br; k_in_rp*X_8Br;
rp8pcpt_entry:  -> X_8pCPT; k_in_rp*(Rp8pCPT - X_8pCPT - I_8pCPT);
rp8pcpt_cleave: X_8pCPT -> I_8pCPT; k_in_rp*X_8pCPT;
rpr_entry:  -> X_Rp; k_in_rp*(Rp - X_Rp - I_Rp);
rpr_cleave: X_Rp -> I_Rp; k_in_rp*X_Rp;

// Species initializations:
cAMP = 0;
S_sp8 = 0;
I_8Br = 0;
I_8pCPT = 0;
I_Rp = 0;
P_RII = 0;
P_w = ;
X_8Br = ;
X_8pCPT = ;
X_Rp = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
Fsk_level = ;
Fsk_time = ;
IBMX_level = ;
IBMX_time = ;
H89_level = ;
H89_time = ;
Sp8_Br_cAMPS_AM_level = ;
Sp8_Br_cAMPS_AM_time = ;
Rp8_Br_cAMPS_pAB_level = ;
Rp8_Br_cAMPS_pAB_incubation_time = ;
Rp8_pCPT_cAMPS_pAB_level = ;
Rp8_pCPT_cAMPS_pAB_incubation_time = ;
Rp_cAMPS_pAB_level = ;
Rp_cAMPS_pAB_incubation_time = ;
fourABnOH_level = ;
fourABnOH_incubation_time = ;
k_pde = 1;
i_ibmx = 0.5;
k_out_sp = 0.15;
w_sp = 10;
n_act = 1.3;
K_act = 18;
Ki_8Br = 1;
Ki_8pCPT = 2;
k_phos = 0.3;
k_dephos = 0.025;
m_w = ;
b_pRII_Western = 3;
theta_vis = 0.5;
b_Calpha_global = ;
```

```
  f_basal = ;
  f_fsk = ;
  K_fsk = ;
  Km_pde = ;
  Ki_H89 = ;
  k_phos_w = ;
  k_dephos_w = ;
  k_in_rp = ;

  // Other declarations:
  var Fsk, IBMX, H89, Sp8, Rp8Br, Rp8pCPT, Rp, fourABnOH, cep, C_free,
    readout_pRII;
  var readout_pRII_Western, readout_Calpha, b_pRII_global;
  const cell, Fsk_level, Fsk_time, IBMX_level, IBMX_time, H89_level,
    H89_time;
  const Sp8_Br_cAMPS_AM_level, Sp8_Br_cAMPS_AM_time,
    Rp8_Br_cAMPS_pAB_level;
  const Rp8_Br_cAMPS_pAB_incubation_time, Rp8_pCPT_cAMPS_pAB_level,
    Rp8_pCPT_cAMPS_pAB_incubation_time;
  const Rp_cAMPS_pAB_level, Rp_cAMPS_pAB_incubation_time,
    fourABnOH_level;
  const fourABnOH_incubation_time, k_pde, i_ibmx, k_out_sp, w_sp, n_act
    , K_act;
  const Ki_8Br, Ki_8pCPT, k_phos, k_dephos, m_w, b_pRII_Western,
    theta_vis;
  const b_Calpha_global, f_basal, f_fsk, K_fsk, Km_pde, Ki_H89,
    k_phos_w;
  const k_dephos_w, k_in_rp;
end
```

Listing 101: Archived Antimony for `isensee_3`; task `56836711_2`; selected version 17

```
// Created by libAntimony v3.1.3
model *pka_rii_sensory_neuron()

  // Compartments and Species:
  compartment cell;
  species cAMP in cell, H in cell, HP in cell, R2 in cell, R2P in cell,
     C in cell;
  species SpA in cell, Rp8Br_A in cell, Rp8pCPT_A in cell, RpcAMPS_A in
     cell;
  species ABnOH in cell;

  // Assignment Rules:
  Fsk_in := Fsk_level*piecewise(0, time < Fsk_time, 1);
  IBMX_in := IBMX_level*piecewise(0, time < IBMX_time, 1);
  H89_in := H89_level*piecewise(0, time < H89_time, 1);
  Sp8_in := Sp8_Br_cAMPS_AM_level*piecewise(0, time <
    Sp8_Br_cAMPS_AM_time, 1);
  Rp8Br_in := Rp8_Br_cAMPS_pAB_level*piecewise(0, time <
    Rp8_Br_cAMPS_pAB_incubation_time, 1);
  Rp8pCPT_in := Rp8_pCPT_cAMPS_pAB_level*piecewise(0, time <
    Rp8_pCPT_cAMPS_pAB_incubation_time, 1);
  RpcAMPS_in := Rp_cAMPS_pAB_level*piecewise(0, time <
    Rp_cAMPS_pAB_incubation_time, 1);
  IBMX_fac := Ki_ibmx/(Ki_ibmx + IBMX_in);
  h89_fac := Ki_h89/(Ki_h89 + H89_in);
  frac_num := cAMP/K_camp + SpA/K_sp;
  K_camp := 1;
  Rp_occ := Rp8Br_A/K_Rp8Br + Rp8pCPT_A/K_Rp8pCPT + RpcAMPS_A/K_RpcAMPS
    ;
  K_Rp8Br := 1;
  K_RpcAMPS := EC50_fsk;
  frac_act := frac_num/(1 + frac_num + (Rp8pCPT_A/K_Rp8pCPT + RpcAMPS_A
    /K_RpcAMPS + Rp8Br_A)^m_Rp);
```

```
m_Rp := n_fsk;
k_in_Sp := 0.01;
k_cis := k_ac0;
k_phos := 1;
slope_8br := k_in_pcpt;
readout_pRII := R2P;
readout_pRII_Western := R2P + HP;
readout_Calpha := C + m_Calpha*(H + HP);

// Reactions:
camp_production:  -> cAMP; k_ac0 + k_acF*Fsk_in^n_fsk/(EC50_fsk^n_fsk
   + Fsk_in^n_fsk);
camp_degradation: cAMP -> ; cAMP*IBMX_fac;
sp_loading:  -> SpA; k_in_Sp*(Sp8_in - SpA);
rp8br_loading:  -> Rp8Br_A; slope_8br*Rp8Br_in;
rp8pcpt_loading:  -> Rp8pCPT_A; k_in_pcpt*(Rp8pCPT_in - Rp8pCPT_A);
rpcamps_loading:  -> RpcAMPS_A; k_in_rpc*(RpcAMPS_in - RpcAMPS_A);
abnoh_input:  -> ABnOH; fourABnOH_level*piecewise(0, time <
  fourABnOH_incubation_time, 1);
abnoh_clearance: ABnOH -> ; k_dp*ABnOH;
holo_release: H -> R2 + 2 C; k_rel*H*frac_act;
r2_rebinding: R2 + 2 C -> H; 60*R2*C^2;
cis_autophos: H -> HP; k_cis*H*(1 + frac_num);
holoP_release: HP -> R2P + 2 C; k_rel_p*HP*frac_act;
r2_transphos: R2 + C -> R2P + C; k_phos*R2*C*h89_fac;
r2p_dephos: R2P -> R2; k_dp*R2P;
hp_dephos: HP -> H; k_dp_bound*HP;

// Species initializations:
cAMP = 0;
H = H_init;
HP = 0;
R2 = 0;
R2P = 0;
C = 0;
SpA = 0;
Rp8Br_A = 0;
Rp8pCPT_A = 0;
RpcAMPS_A = 0;
ABnOH = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
H_init = 1;
Fsk_level = ;
Fsk_time = ;
IBMX_level = ;
IBMX_time = ;
H89_level = ;
H89_time = ;
Sp8_Br_cAMPS_AM_level = ;
Sp8_Br_cAMPS_AM_time = ;
Rp8_Br_cAMPS_pAB_level = ;
Rp8_Br_cAMPS_pAB_incubation_time = ;
Rp8_pCPT_cAMPS_pAB_level = ;
Rp8_pCPT_cAMPS_pAB_incubation_time = ;
Rp_cAMPS_pAB_level = ;
Rp_cAMPS_pAB_incubation_time = ;
Ki_ibmx = 300;
Ki_h89 = 50;
K_sp = 1;
K_Rp8pCPT = 1;
EC50_fsk = 2;
```

```
  n_fsk = ;
  k_ac0 = 0.02;
  k_acF = 0.5;
  k_in_pcpt = 0.1;
  k_in_rpc = 0.1;
  k_rel = 10;
  k_rel_p = 10;
  k_dp = 0.011;
  m_Calpha = 1;
  fourABnOH_level = ;
  fourABnOH_incubation_time = ;
  k_dp_bound = ;

  // Other declarations:
  var Fsk_in, IBMX_in, H89_in, Sp8_in, Rp8Br_in, Rp8pCPT_in, RpcAMPS_in
    , IBMX_fac;
  var h89_fac, frac_num, K_camp, Rp_occ, K_Rp8Br, K_RpcAMPS, frac_act,
    m_Rp;
  var k_in_Sp, k_cis, k_phos, slope_8br, readout_pRII,
    readout_pRII_Western;
  var readout_Calpha;
  const cell, H_init, Fsk_level, Fsk_time, IBMX_level, IBMX_time,
    H89_level;
  const H89_time, Sp8_Br_cAMPS_AM_level, Sp8_Br_cAMPS_AM_time,
    Rp8_Br_cAMPS_pAB_level;
  const Rp8_Br_cAMPS_pAB_incubation_time, Rp8_pCPT_cAMPS_pAB_level,
    Rp8_pCPT_cAMPS_pAB_incubation_time;
  const Rp_cAMPS_pAB_level, Rp_cAMPS_pAB_incubation_time, Ki_ibmx,
    Ki_h89;
  const K_sp, K_Rp8pCPT, EC50_fsk, n_fsk, k_ac0, k_acF, k_in_pcpt,
    k_in_rpc;
  const k_rel, k_rel_p, k_dp, m_Calpha, fourABnOH_level,
    fourABnOH_incubation_time;
  const k_dp_bound;
end
```

Listing 102: Archived Antimony for `isensee_4`; task `56836711_0`; selected version 17

```
// Created by libAntimony v3.1.3
model *pka_rii()

  // Compartments and Species:
  compartment cell;
  species Cfree in cell, Rfree in cell, pRfree in cell, RC in cell, pRC
     in cell;
  species RpA8Br in cell, RpA8pCPT in cell, RpAcAMPS in cell;

  // Assignment Rules:
  Fsk := Fsk_level*piecewise(0, time < Fsk_time, 1);
  IBMX_eff := IBMX_level*piecewise(0, time < IBMX_time, 1);
  H89_eff := H89_level*piecewise(0, time < H89_time, 1);
  Sp8_eff := Sp8_Br_cAMPS_AM_level*piecewise(0, time <
    Sp8_Br_cAMPS_AM_time, 1);
  Rp8Br_eff := Rp8_Br_cAMPS_pAB_level*piecewise(0, time <
    Rp8_Br_cAMPS_pAB_incubation_time, 1);
  Rp8pCPT_eff := Rp8_pCPT_cAMPS_pAB_level*piecewise(0, time <
    Rp8_pCPT_cAMPS_pAB_incubation_time, 1);
  RpcAMPS_eff := Rp_cAMPS_pAB_level*piecewise(0, time <
    Rp_cAMPS_pAB_incubation_time, 1);
  fourABnOH_eff := fourABnOH_level*piecewise(0, time <
    fourABnOH_incubation_time, 1);
  camp_eff := (a0 + a1*Fsk)*(1 + IBMX_eff/K_ibmx) + w_Sp8*Sp8_eff;
  f_H89 := 1/(1 + H89_eff/IC50_h89);
  Rp_tot_scaled := RpA8Br/K_8Br + RpA8pCPT/K_8pCPT + RpAcAMPS/K_cAMPS;
```

```
T8 := piecewise(0, time < Rp8_Br_cAMPS_pAB_incubation_time, time - Rp8_Br_cAMPS_pAB_incubation_time);
T8p := piecewise(0, time < Rp8_pCPT_cAMPS_pAB_incubation_time, time - Rp8_pCPT_cAMPS_pAB_incubation_time);
T8c := piecewise(0, time < Rp_cAMPS_pAB_incubation_time, time - Rp_cAMPS_pAB_incubation_time);
readout_pRII := pRfree;
readout_pRII_Western := pRfree + pRC;
readout_Calpha := Cfree/(Km_C + Cfree);

// Reactions:
rc_open: RC -> Rfree + Cfree; k_open*RC*piecewise(0, camp_eff < (Kd_c*(1 + Rp_tot_scaled)), 1);
prc_open: pRC -> pRfree + Cfree; k_open_p*pRC*camp_eff/(1 + Rp_tot_scaled + camp_eff);
rc_rebind: Rfree + Cfree -> RC; k_on*Rfree*Cfree;
prc_sequester: pRfree + Cfree -> pRC; k_on_p*pRfree*Cfree;
pr_dephos: pRfree -> Rfree; k_deph*pRfree;
load_8br:  -> RpA8Br; Rp8_Br_cAMPS_pAB_level*n_l*T8^(n_l - 1)*tau_l^n_l/(T8^n_l + tau_l^n_l)^2;
load_8pCPT:  -> RpA8pCPT; Rp8_pCPT_cAMPS_pAB_level*n_l*T8p^(n_l - 1)*tau_l^n_l/(T8p^n_l + tau_l^n_l)^2;
load_cAMPS:  -> RpAcAMPS; Rp_cAMPS_pAB_level*n_l*T8c^(n_l - 1)*tau_l^n_l/(T8c^n_l + tau_l^n_l)^2;
autophos: RC -> pRC; k_auto*RC*f_H89;

// Species initializations:
Cfree = 0;
Rfree = Rfree_init;
pRfree = 0;
RC = RC_init;
pRC = 0;
RpA8Br = 0;
RpA8pCPT = 0;
RpAcAMPS = 0;

// Compartment initializations:
cell = 1;

// Variable initializations:
Rfree_init = 1;
RC_init = 1;
Fsk_level = ;
Fsk_time = ;
IBMX_level = ;
IBMX_time = ;
H89_level = ;
H89_time = ;
Sp8_Br_cAMPS_AM_level = ;
Sp8_Br_cAMPS_AM_time = ;
Rp8_Br_cAMPS_pAB_level = ;
Rp8_Br_cAMPS_pAB_incubation_time = ;
Rp8_pCPT_cAMPS_pAB_level = ;
Rp8_pCPT_cAMPS_pAB_incubation_time = ;
Rp_cAMPS_pAB_level = ;
Rp_cAMPS_pAB_incubation_time = ;
fourABnOH_level = ;
fourABnOH_incubation_time = ;
a0 = ;
a1 = ;
K_ibmx = ;
w_Sp8 = 0.05;
IC50_h89 = 200;
k_open = 5;
k_open_p = 0.5;
```

```
  Kd_c = 10;
  k_on = 10;
  k_on_p = 5;
  k_deph = 0.05;
  K_8Br = 4;
  K_8pCPT = 3;
  K_cAMPS = 8;
  Km_C = ;
  n_l = ;
  tau_l = ;
  k_auto = ;

  // Other declarations:
  var Fsk, IBMX_eff, H89_eff, Sp8_eff, Rp8Br_eff, Rp8pCPT_eff,
    RpcAMPS_eff;
  var fourABnOH_eff, camp_eff, f_H89, Rp_tot_scaled, T8, T8p, T8c,
    readout_pRII;
  var readout_pRII_Western, readout_Calpha;
  const cell, Rfree_init, RC_init, Fsk_level, Fsk_time, IBMX_level,
    IBMX_time;
  const H89_level, H89_time, Sp8_Br_cAMPS_AM_level,
    Sp8_Br_cAMPS_AM_time;
  const Rp8_Br_cAMPS_pAB_level, Rp8_Br_cAMPS_pAB_incubation_time,
    Rp8_pCPT_cAMPS_pAB_level;
  const Rp8_pCPT_cAMPS_pAB_incubation_time, Rp_cAMPS_pAB_level,
    Rp_cAMPS_pAB_incubation_time;
  const fourABnOH_level, fourABnOH_incubation_time, a0, a1, K_ibmx,
    w_Sp8;
  const IC50_h89, k_open, k_open_p, Kd_c, k_on, k_on_p, k_deph, K_8Br,
    K_8pCPT;
  const K_cAMPS, Km_C, n_l, tau_l, k_auto;
end
```

Listing 103: Archived Antimony for `isensee_5`; task `58312543_0`; selected version 1

```
// Created by libAntimony v3.1.3
model *pka_rii_signaling()

  // Compartments and Species:
  species cAMP, R2C2, pR2C2, R2free, pR2free, Cfree, Sp8Br, Rp8Br,
    Rp8pCPT;
  species Rp, Fsk_level, Fsk_time, IBMX_level, IBMX_time, H89_level,
    H89_time;
  species Sp8_Br_cAMPS_AM_level, Sp8_Br_cAMPS_AM_time,
    Rp8_Br_cAMPS_pAB_level;
  species Rp8_Br_cAMPS_pAB_incubation_time, Rp8_pCPT_cAMPS_pAB_level,
    Rp8_pCPT_cAMPS_pAB_incubation_time;
  species Rp_cAMPS_pAB_level, Rp_cAMPS_pAB_incubation_time,
    fourABnOH_level;
  species fourABnOH_incubation_time;

  // Assignment Rules:
  Fsk_eff := Fsk_level*piecewise(0, time < Fsk_time, 1);
  IBMX_eff := IBMX_level*piecewise(0, time < IBMX_time, 1);
  H89_eff := H89_level*piecewise(0, time < H89_time, 1);
  Sp8_in := Sp8_Br_cAMPS_AM_level*piecewise(0, time <
    Sp8_Br_cAMPS_AM_time, 1);
  RpBr_in := Rp8_Br_cAMPS_pAB_level*piecewise(0, time <
    Rp8_Br_cAMPS_pAB_incubation_time, 1);
  RpcCPT_in := Rp8_pCPT_cAMPS_pAB_level*piecewise(0, time <
    Rp8_pCPT_cAMPS_pAB_incubation_time, 1);
  Rp_in := Rp_cAMPS_pAB_level*piecewise(0, time <
    Rp_cAMPS_pAB_incubation_time, 1);
  fourABnOH_eff := fourABnOH_level*piecewise(0, time <
    fourABnOH_incubation_time, 1);
```

```
camp_eff := (cAMP + Sp8Br)/(1 + Rp8Br + Rp8pCPT + Rp);
f_open := camp_eff^n_open/(Kd_open^n_open + camp_eff^n_open);
h89_fac := KmH89/(KmH89 + H89_eff);
readout_pRII := pR2free;
readout_pRII_Western := pR2free + pR2C2;
readout_Calpha := Cfree;

// Reactions:
open_R2C2: R2C2 -> R2free + 2 Cfree; k_open*f_open*R2C2;
open_pR2C2: pR2C2 -> pR2free + 2 Cfree; k_open*f_open*pR2C2;
close_R2: R2free + 2 Cfree -> R2C2; k_close*R2free*Cfree^2;
close_pR2: pR2free + 2 Cfree -> pR2C2; k_close*pR2free*Cfree^2;
trans_phos: R2free + Cfree -> pR2free + Cfree; k_trans*h89_fac*R2free
  *Cfree;
dephos_bound: pR2C2 -> R2C2; k_dp_bound*pR2C2;
ac_production:  -> cAMP; kpde_c*(camp_basal + camp_fsk_drive*Fsk_eff
  /(Km_fsk + Fsk_eff));
pde_degradation: cAMP -> ; kpde_c*(KmI/(KmI + IBMX_eff))*cAMP;
sp_load:  -> Sp8Br; k_form_sp*Sp8_in;
sp_loss: Sp8Br -> ; k_loss_sp*Sp8Br;
rpbr_load:  -> Rp8Br; k_form_rpbr*RpBr_in;
rpbr_loss: Rp8Br -> ; k_loss_rp*Rp8Br;
rpcpt_load:  -> Rp8pCPT; k_form_rpcpt*RpcCPT_in;
rpcpt_loss: Rp8pCPT -> ; k_loss_rp*Rp8pCPT;
rp_load:  -> Rp; k_form_rp*Rp_in;
rp_loss: Rp -> ; k_loss_rp*Rp;

// Species initializations:
cAMP = cAMP_init;
R2C2 = R2C2_init;
pR2C2 = 0;
R2free = 0;
pR2free = 0;
Cfree = 0;
Sp8Br = 0;
Rp8Br = 0;
Rp8pCPT = 0;
Rp = 0;
Fsk_level = 0;
Fsk_time = 0;
IBMX_level = 0;
IBMX_time = 0;
H89_level = 0;
H89_time = 0;
Sp8_Br_cAMPS_AM_level = 0;
Sp8_Br_cAMPS_AM_time = 0;
Rp8_Br_cAMPS_pAB_level = 0;
Rp8_Br_cAMPS_pAB_incubation_time = 0;
Rp8_pCPT_cAMPS_pAB_level = 0;
Rp8_pCPT_cAMPS_pAB_incubation_time = 0;
Rp_cAMPS_pAB_level = 0;
Rp_cAMPS_pAB_incubation_time = 0;
fourABnOH_level = 0;
fourABnOH_incubation_time = 0;

// Variable initializations:
cAMP_init = 1;
R2C2_init = 1;
Km_fsk = 1;
KmI = 30;
k_open = 1;
Kd_open = 2.5;
n_open = 2;
k_close = 0.5;
k_trans = 0.1;
```

```
  k_dp_bound = 0.002;
  KmH89 = 10;
  kpde_c = ;
  camp_basal = ;
  camp_fsk_drive = ;
  k_form_sp = ;
  k_loss_sp = ;
  k_form_rpbr = ;
  k_loss_rp = ;
  k_form_rpcpt = ;
  k_form_rp = ;

  // Other declarations:
  var Fsk_eff, IBMX_eff, H89_eff, Sp8_in, RpBr_in, RpcCPT_in, Rp_in,
    fourABnOH_eff;
  var camp_eff, f_open, h89_fac, readout_pRII, readout_pRII_Western,
    readout_Calpha;
  const cAMP_init, R2C2_init, Km_fsk, KmI, k_open, Kd_open, n_open,
    k_close;
  const k_trans, k_dp_bound, KmH89, kpde_c, camp_basal, camp_fsk_drive,
     k_form_sp;
  const k_loss_sp, k_form_rpbr, k_loss_rp, k_form_rpcpt, k_form_rp;
end
```

H.16 ALKAN

Listing 104: Archived Antimony for `alkan_1`; task `56836792_0`; selected version 0

```
// Created by libAntimony v3.1.3
model *DDR_dox_gem_sn38()

  // Compartments and Species:
  compartment cell;
  species Dox_level in cell, Gem_level in cell, SN38_level in cell,
    D_dox in cell;
  species D_gem in cell, D_sn38 in cell, R_dox in cell, A_atm in cell,
    C_chk2 in cell;
  species K_chk1 in cell, K_dpk in cell, H2ax in cell, P53p in cell,
    P53t in cell;
  species M_p21 in cell, M_wip1 in cell, P_p21 in cell, N_cell in cell,
     X_casp in cell;
  species Arrest in cell;

  // Assignment Rules:
  P53sig := (P53p + w_act_c1*K_chk1)*P53t;
  InhDox := p_mrna_exp_inh_dox_kd/(p_mrna_exp_inh_dox_kd + Dox_level);
  readout_Cells_count := N_cell + X_casp;
  readout_CellsCasp_count := X_casp;
  readout_pATM_au := A_atm;
  readout_pChk2_au := C_chk2;
  readout_pChk1_au := K_chk1;
  readout_pDNAPK_au := K_dpk;
  readout_yH2AX_au := H2ax;
  readout_pp53_au := P53p;
  readout_tp53_au := P53t;
  readout_p21_mRNA_fold := M_p21;
  readout_Wip1_mRNA_fold := M_wip1;
  readout_tp21_au := P_p21;

  // Reactions:
  form_dox:  -> D_dox; k_f_dox*(Dox_level^n_dox/(k_dox_kd^n_dox +
    Dox_level^n_dox))*(1 - Arrest);
  rep_dox: D_dox -> ; k_rep_dox*D_dox;
```

```
form_gem:  -> D_gem; k_f_gem*(Gem_level^n_gem/(K_gem^n_gem + Gem_level^n_gem))*(1 - Arrest);
rep_gem: D_gem -> ; k_rep_gem*D_gem;
form_sn38:  -> D_sn38; k_f_sn38*(SN38_level^n_sn38/(K_sn38^n_sn38 + SN38_level^n_sn38))*(1 - Arrest);
rep_sn38: D_sn38 -> ; k_rep_sn38*D_sn38;
proc_dox:  -> R_dox; k_conv_dox*D_dox;
R_dox_loss: R_dox -> ; k_Rloss*R_dox;
atm_act:  -> A_atm; k_atm_act*(D_dox + w_atm_sn38*D_sn38 + w_atm_gem*D_gem)*(1 - A_atm);
atm_dephos: A_atm -> ; A_atm/dephosphorylation_timescale;
chk2_act:  -> C_chk2; k_chk2_act*A_atm*(1 - C_chk2);
chk2_dephos: C_chk2 -> ; C_chk2/dephosphorylation_timescale;
chk1_act:  -> K_chk1; k_chk1_act*(D_gem + w_c1_sn38*D_sn38 + w_c1_dox*R_dox)*(1 - K_chk1);
chk1_dephos: K_chk1 -> ; K_chk1/dephosphorylation_timescale;
h2ax_act:  -> H2ax; k_h2ax_act*(D_dox + w_h_sn38*D_sn38 + w_h_gem*D_gem)*(1 - H2ax);
h2ax_dephos: H2ax -> ; H2ax/dephosphorylation_timescale;
dpk_act:  -> K_dpk; k_dpk_act*H2ax*(1 - K_dpk);
dpk_dephos: K_dpk -> ; K_dpk/dephosphorylation_timescale;
p53p_act:  -> P53p; k_p53p_act*(A_atm + w_p53_c2*C_chk2 + w_p53_c1*K_chk1)*(1 - P53p);
p53p_dephos: P53p -> ; P53p/dephosphorylation_timescale;
p53_syn:  -> P53t; k_syn_p53;
p53_deg: P53t -> ; k_deg_p53*P53t/(1 + stab_p53*P53p);
p21_tx:  -> M_p21; (P53sig^n_p21/(K_p21^n_p21 + P53sig^n_p21))*InhDox;
p21_m_deg: M_p21 -> ; k_deg_m*M_p21;
wip1_tx:  -> M_wip1; (P53sig^n_p21/(K_p21^n_p21 + P53sig^n_p21))*InhDox;
wip1_m_deg: M_wip1 -> ; k_deg_m*M_wip1;
p21_tl:  -> P_p21; M_p21;
p21_p_deg: P_p21 -> ; k_deg_p21p*P_p21;
grow:  -> N_cell; k_g*N_cell*(1 - (N_cell + X_casp)/init_Space)*(1 - Arrest);
arrest_on:  -> Arrest; k_ar_on*(K_chk1 + C_chk2)*(1 - Arrest);
arrest_off: Arrest -> ; k_ar_off*Arrest;
commit_dox: N_cell -> X_casp; k_kill_dox*D_dox*(init_Cells_Cycle_S_rel + init_Cells_Cycle_G2_rel)*N_cell;
lysis: X_casp -> ; k_lysis*X_casp;

// Species initializations:
Dox_level = 0;
Gem_level = 0;
SN38_level = 0;
D_dox = ;
D_gem = ;
D_sn38 = ;
R_dox = ;
A_atm = ;
C_chk2 = ;
K_chk1 = ;
K_dpk = ;
H2ax = ;
P53p = ;
P53t = ;
M_p21 = ;
M_wip1 = ;
P_p21 = ;
N_cell = init_Cells;
X_casp = ;
Arrest = ;

// Compartment initializations:
```

```
cell = 1;

// Variable initializations:
init_Cells = ;
w_act_c1 = 1;
p_mrna_exp_inh_dox_kd = ;
k_f_dox = 1;
n_dox = 2;
k_rep_dox = 0.05;
k_f_gem = 1;
n_gem = 2;
K_gem = 0.02;
k_rep_gem = 0.1;
k_f_sn38 = 1;
n_sn38 = 2;
K_sn38 = 0.02;
k_rep_sn38 = 0.2;
k_conv_dox = 0.1;
k_Rloss = 0.1;
k_atm_act = 2;
w_atm_sn38 = 1;
w_atm_gem = 0.3;
k_chk2_act = 2;
k_chk1_act = 1;
w_c1_sn38 = 1;
w_c1_dox = 1;
k_h2ax_act = 0.5;
w_h_sn38 = 0.5;
w_h_gem = 0.2;
k_dpk_act = 0.5;
k_p53p_act = 1;
w_p53_c2 = 0.5;
w_p53_c1 = 0.1;
k_syn_p53 = 0.05;
k_deg_p53 = 0.3;
stab_p53 = 10;
n_p21 = 2;
K_p21 = 1;
k_deg_m = 0.3;
k_deg_p21p = 0.1;
k_g = 0.034;
k_ar_on = 0.3;
k_ar_off = 0.05;
k_kill_dox = 0.5;
k_lysis = 0.2;
k_dox_kd = ;
dephosphorylation_timescale = ;
init_Space = ;
init_Cells_Cycle_S_rel = ;
init_Cells_Cycle_G2_rel = ;

// Other declarations:
var P53sig, InhDox, readout_Cells_count, readout_CellsCasp_count,
  readout_pATM_au;
var readout_pChk2_au, readout_pChk1_au, readout_pDNAPK_au,
  readout_yH2AX_au;
var readout_pp53_au, readout_tp53_au, readout_p21_mRNA_fold,
  readout_Wip1_mRNA_fold;
var readout_tp21_au;
const cell, init_Cells, w_act_c1, p_mrna_exp_inh_dox_kd, k_f_dox,
  n_dox;
const k_rep_dox, k_f_gem, n_gem, K_gem, k_rep_gem, k_f_sn38, n_sn38,
  K_sn38;
const k_rep_sn38, k_conv_dox, k_Rloss, k_atm_act, w_atm_sn38,
  w_atm_gem;
```

```
  const k_chk2_act, k_chk1_act, w_c1_sn38, w_c1_dox, k_h2ax_act,
    w_h_sn38;
  const w_h_gem, k_dpk_act, k_p53p_act, w_p53_c2, w_p53_c1, k_syn_p53,
    k_deg_p53;
  const stab_p53, n_p21, K_p21, k_deg_m, k_deg_p21p, k_g, k_ar_on,
    k_ar_off;
  const k_kill_dox, k_lysis, k_dox_kd, dephosphorylation_timescale,
    init_Space;
  const init_Cells_Cycle_S_rel, init_Cells_Cycle_G2_rel;
end
```

Listing 105: Archived Antimony for `alkan_2`; task `56836792_1`; selected version 15

```
// Created by libAntimony v3.1.3
model *ddr_chemo()

  // Compartments and Species:
  compartment well;
  species Cells_G1 in well, Cells_S in well, Cells_G2 in well,
    Cells_Casp in well;
  species DSBf in well, DSBs in well, $RS in well, RSs, pATM in well,
    pChk1 in well;
  species pChk2 in well, pDNAPK in well, yH2AX in well, P53 in well,
    pP53 in well;
  species TP21 in well, p53act in well, Mdm2;

  // Assignment Rules:
  RS := RSs + vq_sn38*SN38_level*Cells_S/(K_sn38_rs + SN38_level);
  G21 := K_p21/(K_p21 + TP21);
  surv_dox := (p_mrna_exp_inh_dox_kd/(p_mrna_exp_inh_dox_kd + Dox_level
    ))^h_dox;
  m21sig := surv_dox*(1 + g21*p53act/(K_m21 + p53act)) - 1;
  m21sig_w := surv_dox*(1 + g_wrel*p53act/(K_m21 + p53act)) - 1;
  Ga := KL^hG/(KL^hG + (DSBf + DSBs + wA*RS)^hG);
  readout_Cells_count := Cells_G1 + Cells_S + Cells_G2 + Cells_Casp;
  readout_CellsCasp_count := Cells_Casp;
  readout_yH2AX_au := yH2AX;
  readout_pATM_au := pATM;
  readout_pChk1_au := pChk1;
  readout_pChk2_au := pChk2;
  readout_pDNAPK_au := pDNAPK;
  readout_pp53_au := pP53;
  readout_tp53_au := P53 - 1;
  readout_tp21_au := TP21;
  readout_p21_mRNA_fold := piecewise(m21sig, m21sig > 0, 0);
  readout_Wip1_mRNA_fold := piecewise(m21sig_w, m21sig_w > 0, 0);

  // Reactions:
  g1s_transition: Cells_G1 -> Cells_S; k_g1s*G21*Ga*Cells_G1;
  sg2_transition: Cells_S -> Cells_G2; k_sg2*(Krs/(Krs + RS))*Cells_S;
  g2_division: Cells_G2 -> 2 Cells_G1; (k_div*G21*Ga*(init_Space -
    Cells_G1 - Cells_S - Cells_G2 - Cells_Casp)/init_Space)*Cells_G2;
  dsb_s_dox:  -> DSBs; vs_dox*Dox_level*(Cells_S + Cells_G2);
  dsb_s_repair: DSBs -> ; k_reps*DSBs;
  atm_act:  -> pATM; DSBf + DSBs;
  atm_dephos: pATM -> ; pATM/dephosphorylation_timescale;
  chk1_act:  -> pChk1; RS + wq2*DSBs;
  chk1_dephos: pChk1 -> ; kchk1*pChk1/dephosphorylation_timescale;
  chk2_act:  -> pChk2; DSBf + w_chk2s*DSBs;
  chk2_dephos: pChk2 -> ; kw_chk2*pChk2/dephosphorylation_timescale;
  dpk_act:  -> pDNAPK; DSBf + DSBs + kr_dpk*RS;
  dpk_dephos: pDNAPK -> ; kdpk*pDNAPK;
  h2ax_act:  -> yH2AX; vh*(DSBf + DSBs)/(Kh2ax + DSBf + DSBs);
  h2ax_dephos: yH2AX -> ; yH2AX/dephosphorylation_timescale;
  pp53_act:  -> pP53; vap*pATM*pChk2/(Kap + pATM*pChk2);
```

```
pp53_dephos: pP53 -> ; pP53/dephosphorylation_timescale;
p53_syn:  -> P53; kd53 + kmdm2*Mdm2;
p53_deg: P53 -> ; kd53*P53 + kmdm2*Mdm2*P53;
tp21_syn:  -> TP21; surv_dox*pP53;
tp21_deg: TP21 -> ; k21deg*TP21;
p53act_syn:  -> p53act; pATM;
p53act_deg: p53act -> ; k_actdeg*p53act;
death_S: Cells_S -> Cells_Casp; kdeath53*p53act*Cells_S;
death_G2: Cells_G2 -> Cells_Casp; kdeath53*p53act*Cells_G2;
lysis: Cells_Casp -> ; k_lysis*Cells_Casp;
mdm2_syn:  -> Mdm2; km2*p53act;
mdm2_deg: Mdm2 -> ; km2d*Mdm2;
casp_recover: Cells_Casp -> Cells_G1; (kr_c/(1 + (Dox_level/k_dox_kd)
  ^h_rec))*Cells_Casp;
p53_ind:  -> P53; a53*pP53;
rs_gem_slow:  -> RSs; vq_gem*Gem_level*Cells_S/(K_gem_rs + Gem_level)
  ;
rs_slow_deg: RSs -> ; kr_rs*RSs;
dsb_f_prod:  -> DSBf; vf2_sn38*SN38_level*Cells_S + vcol*RS;
dsb_conv: DSBf -> DSBs; kfconv*DSBf;

// Species initializations:
Cells_G1 = init_Cells*(1 - init_Cells_Cycle_S_rel -
  init_Cells_Cycle_G2_rel);
Cells_S = init_Cells*init_Cells_Cycle_S_rel;
Cells_G2 = init_Cells*init_Cells_Cycle_G2_rel;
Cells_Casp = 0;
DSBf = 0;
DSBs = 0;
RSs = ;
pATM = 0;
pChk1 = 0;
pChk2 = 0;
pDNAPK = 0;
yH2AX = 0;
P53 = P53_init;
pP53 = 0;
TP21 = 0;
p53act = 0;
Mdm2 = ;

// Compartment initializations:
well = 1;

// Variable initializations:
init_Cells = ;
init_Cells_Cycle_S_rel = ;
init_Cells_Cycle_G2_rel = ;
vq_sn38 = ;
SN38_level = ;
K_sn38_rs = 0.002;
P53_init = 1;
k_g1s = 0.12;
k_sg2 = 0.09;
k_div = 0.12;
K_p21 = 0.5;
vs_dox = 0.1;
k_reps = 0.03;
K_gem_rs = 0.005;
kr_dpk = 1;
w_chk2s = 0.15;
k21deg = 0.15;
k_actdeg = 0.05;
kw_chk2 = 8;
k_lysis = 0.03;
```

```
  g21 = 1.5;
  g_wrel = 0.35;
  h_dox = 4;
  p_mrna_exp_inh_dox_kd = ;
  Dox_level = ;
  K_m21 = ;
  KL = ;
  hG = ;
  wA = ;
  Krs = ;
  init_Space = ;
  dephosphorylation_timescale = ;
  wq2 = ;
  kchk1 = ;
  kdpk = ;
  vh = ;
  Kh2ax = ;
  vap = ;
  Kap = ;
  kd53 = ;
  kmdm2 = ;
  kdeath53 = ;
  km2 = ;
  km2d = ;
  kr_c = ;
  k_dox_kd = ;
  h_rec = ;
  a53 = ;
  vq_gem = ;
  Gem_level = ;
  kr_rs = ;
  vf2_sn38 = ;
  vcol = ;
  kfconv = ;

  // Other declarations:
  var G21, surv_dox, m21sig, m21sig_w, Ga, readout_Cells_count,
    readout_CellsCasp_count;
  var readout_yH2AX_au, readout_pATM_au, readout_pChk1_au,
    readout_pChk2_au;
  var readout_pDNAPK_au, readout_pp53_au, readout_tp53_au,
    readout_tp21_au;
  var readout_p21_mRNA_fold, readout_Wip1_mRNA_fold;
  const well, init_Cells, init_Cells_Cycle_S_rel,
    init_Cells_Cycle_G2_rel;
  const vq_sn38, SN38_level, K_sn38_rs, P53_init, k_g1s, k_sg2, k_div,
    K_p21;
  const vs_dox, k_reps, K_gem_rs, kr_dpk, w_chk2s, k21deg, k_actdeg,
    kw_chk2;
  const k_lysis, g21, g_wrel, h_dox, p_mrna_exp_inh_dox_kd, Dox_level,
    K_m21;
  const KL, hG, wA, Krs, init_Space, dephosphorylation_timescale, wq2,
    kchk1;
  const kdpk, vh, Kh2ax, vap, Kap, kd53, kmdm2, kdeath53, km2, km2d,
    kr_c;
  const k_dox_kd, h_rec, a53, vq_gem, Gem_level, kr_rs, vf2_sn38, vcol,
     kfconv;
end
```

Listing 106: Archived Antimony for `alkan_3`; task `56836792_2`; selected version 2

```
// Created by libAntimony v3.1.3
model *DDR_zero_shot()

  // Compartments and Species:
  compartment cell;
  species C_G1 in cell, C_S in cell, C_G2 in cell, C_Casp in cell, DSB in cell;
  species RS in cell, pATM in cell, pChk2 in cell, pChk1 in cell, pDNAPK in cell;
  species pp53 in cell, tp53 in cell, yH2AX in cell, mp21 in cell, tp21 in cell;
  species mwip1 in cell, Wip1 in cell, Dox_level in cell, Gem_level in cell;
  species SN38_level in cell;

  // Assignment Rules:
  Ntot := C_G1 + C_S + C_G2 + C_Casp;
  InhDox := (p_mrna_exp_inh_dox_kd/(p_mrna_exp_inh_dox_kd + Dox_level))^n_inh;
  gate_p21 := Kgp21^np21/(Kgp21^np21 + tp21^np21);
  gate_chk1 := Kchk1^nchk1/(Kchk1^nchk1 + pChk1^nchk1);
  gate_cas := DSB^ncas/(KcasD^ncas + DSB^ncas);
  death_signal := k_casD*gate_cas;
  readout_Cells_count := Ntot/init_Cells;
  readout_CellsCasp_count := C_Casp/init_Cells;
  readout_pATM_au := pATM;
  readout_pChk1_au := pChk1;
  readout_pChk2_au := pChk2;
  readout_pDNAPK_au := pDNAPK;
  readout_pp53_au := pp53;
  readout_tp53_au := tp53;
  readout_tp21_au := tp21;
  readout_yH2AX_au := yH2AX;
  readout_p21_mRNA_fold := mp21;
  readout_Wip1_mRNA_fold := mwip1;

  // Reactions:
  g1s_entry: C_G1 -> C_S; a1*C_G1*gate_p21;
  s_g2_entry: C_S -> C_G2; a2*C_S*gate_chk1;
  division: C_G2 -> 2 C_G1; a3*C_G2*(1 - Ntot/init_Space)*gate_p21;
  death_g1: C_G1 -> C_Casp; C_G1*death_signal;
  death_s: C_S -> C_Casp; C_S*death_signal;
  death_g2: C_G2 -> C_Casp; C_G2*death_signal;
  lysis: C_Casp -> ; k_lyse*C_Casp;
  dox_dsb:  -> DSB; kD_dox*(Ntot/init_Cells)*Dox_level/(k_dox_kd + Dox_level);
  sn38_dsb:  -> DSB; kD_sn38*(C_S/init_Cells)*SN38_level;
  gem_rs:  -> RS; kR_gem*(C_S/init_Cells)*Gem_level;
  sn38_rs:  -> RS; kR_sn38*(C_S/init_Cells)*SN38_level;
  rs_collapse: RS -> DSB; k_collapse*RS;
  dsb_repair: DSB -> ; k_rep*DSB/(K_rep + DSB);
  rs_resolve: RS -> ; k_res*RS;
  atm_act:  -> pATM; k_ATM*DSB/(KATM + DSB);
  atm_dephos: pATM -> ; pATM*(1/dephosphorylation_timescale + kW_atm*Wip1);
  chk2_act:  -> pChk2; k_Chk2*pATM;
  chk2_dephos: pChk2 -> ; pChk2*(1/dephosphorylation_timescale + kW_chk2*Wip1);
  chk1_act:  -> pChk1; k_Chk1R*RS + k_Chk1D*DSB;
  chk1_dephos: pChk1 -> ; pChk1*(1/dephosphorylation_timescale + kW_chk1*Wip1);
  dnapk_act:  -> pDNAPK; k_DPK*DSB;
  dnapk_dephos: pDNAPK -> ; pDNAPK/dephosphorylation_timescale;
```

```
p53_act:  -> pp53; k_p53A*pATM + k_p53C*pChk1;
p53_dephos: pp53 -> ; pp53*(1/dephosphorylation_timescale + kW_p53*
  Wip1);
tp53_syn:  -> tp53; k_syn53*pp53;
tp53_deg: tp53 -> ; k_deg53*tp53;
mp21_syn:  -> mp21; k_sm21*tp53*InhDox;
mp21_deg: mp21 -> ; k_dm21*mp21;
tp21_syn:  -> tp21; k_sp21*mp21;
tp21_deg: tp21 -> ; k_dp21*tp21;
mwip1_syn:  -> mwip1; k_smW*tp53*InhDox;
mwip1_deg: mwip1 -> ; k_dmW*mwip1;
wip1_syn:  -> Wip1; k_sW*mwip1;
wip1_deg: Wip1 -> ; k_dW*Wip1;
h2ax_phos:  -> yH2AX; kH2ax_D*DSB + kH2ax_R*RS;
h2ax_dephos: yH2AX -> ; yH2AX/dephosphorylation_timescale;

// Species initializations:
C_G1 = init_Cells*(1 - init_Cells_Cycle_S_rel -
  init_Cells_Cycle_G2_rel);
C_S = init_Cells*init_Cells_Cycle_S_rel;
C_G2 = init_Cells*init_Cells_Cycle_G2_rel;
C_Casp = ;
DSB = ;
RS = ;
pATM = ;
pChk2 = ;
pChk1 = ;
pDNAPK = ;
pp53 = ;
tp53 = ;
yH2AX = ;
mp21 = ;
tp21 = ;
mwip1 = ;
Wip1 = ;
Dox_level = ;
Gem_level = ;
SN38_level = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_Cells = ;
init_Cells_Cycle_S_rel = ;
init_Cells_Cycle_G2_rel = ;
p_mrna_exp_inh_dox_kd = ;
n_inh = ;
Kgp21 = 0.5;
np21 = 3;
Kchk1 = 1;
nchk1 = 4;
ncas = 3;
KcasD = 1;
k_casD = 0.2;
a1 = 0.09;
a2 = 0.125;
a3 = 0.25;
kD_dox = 0.5;
kD_sn38 = 0.3;
kR_gem = 0.5;
kR_sn38 = 0.5;
k_collapse = 0.3;
k_rep = 1;
k_res = 0.5;
```

```
  k_ATM = 10;
  KATM = 0.3;
  kW_atm = 0.05;
  k_Chk2 = 3;
  kW_chk2 = 0.2;
  k_Chk1R = 2;
  k_Chk1D = 0.05;
  k_DPK = 0.3;
  k_p53A = 0.5;
  k_p53C = 0.5;
  kW_p53 = 0.05;
  k_syn53 = 0.3;
  k_deg53 = 0.1;
  k_sm21 = 0.5;
  k_dm21 = 0.35;
  k_sp21 = 0.3;
  k_dp21 = 0.08;
  k_smW = 0.2;
  k_dmW = 0.25;
  k_sW = 0.3;
  k_dW = 0.1;
  kH2ax_D = 0.5;
  kH2ax_R = 0.3;
  k_lyse = 0.05;
  init_Space = ;
  k_dox_kd = ;
  K_rep = ;
  dephosphorylation_timescale = ;
  kW_chk1 = ;

  // Other declarations:
  var Ntot, InhDox, gate_p21, gate_chk1, gate_cas, death_signal,
    readout_Cells_count;
  var readout_CellsCasp_count, readout_pATM_au, readout_pChk1_au,
    readout_pChk2_au;
  var readout_pDNAPK_au, readout_pp53_au, readout_tp53_au,
    readout_tp21_au;
  var readout_yH2AX_au, readout_p21_mRNA_fold, readout_Wip1_mRNA_fold;
  const cell, init_Cells, init_Cells_Cycle_S_rel,
    init_Cells_Cycle_G2_rel;
  const p_mrna_exp_inh_dox_kd, n_inh, Kgp21, np21, Kchk1, nchk1, ncas,
    KcasD;
  const k_casD, a1, a2, a3, kD_dox, kD_sn38, kR_gem, kR_sn38,
    k_collapse;
  const k_rep, k_res, k_ATM, KATM, kW_atm, k_Chk2, kW_chk2, k_Chk1R,
    k_Chk1D;
  const k_DPK, k_p53A, k_p53C, kW_p53, k_syn53, k_deg53, k_sm21, k_dm21
    , k_sp21;
  const k_dp21, k_smW, k_dmW, k_sW, k_dW, kH2ax_D, kH2ax_R, k_lyse,
    init_Space;
  const k_dox_kd, K_rep, dephosphorylation_timescale, kW_chk1;
end
```

Listing 107: Archived Antimony for `alkan_4`; task `56836792_3`; selected version 9

```
// Created by libAntimony v3.1.3
model *ddr()

  // Compartments and Species:
  compartment cell;
  species L_dox in cell, L_gem in cell, L_sn38 in cell, pATM in cell,
    pChk2 in cell;
  species pChk1 in cell, pDNAPK in cell, yH2AX in cell, pp53 in cell,
    tp53 in cell;
```

```
species p21mRNA in cell, Wip1mRNA in cell, Wip1 in cell, tp21 in cell, Nc in cell;
species Casp in cell, C2;

// Assignment Rules:
Les_DSB := L_dox + L_sn38 + w_gem_les*L_gem;
Chk_sig := pATM + w_chk2_p53*pChk2;
gate_dox := Dox_level^h_casp/(k_dox_casp_kd^h_casp + Dox_level^h_casp);
inh_dox_tx := (p_mrna_exp_inh_dox_kd/(p_mrna_exp_inh_dox_kd + Dox_level))^h_tx;
ArrestChk := (pChk1 + pChk2 + w_ar_p21*tp21)/(K_ar_w + pChk1 + pChk2 + w_ar_p21*tp21);
readout_yH2AX_au := yH2AX;
readout_pATM_au := pATM;
readout_pChk1_au := pChk1;
readout_pChk2_au := pChk2;
readout_pDNAPK_au := pDNAPK;
readout_pp53_au := pp53;
readout_tp53_au := tp53;
readout_tp21_au := tp21;
readout_p21_mRNA_fold := p21mRNA;
readout_Wip1_mRNA_fold := Wip1mRNA;
readout_Cells_count := Nc + C2 + Casp;
readout_CellsCasp_count := Casp;

// Reactions:
dox_lesion:  -> L_dox; k_f_dox*(Dox_level^h_dox/(K_Ldox^h_dox + Dox_level^h_dox));
gem_lesion:  -> L_gem; k_f_gem*(Gem_level^h_gem/(K_Lgem^h_gem + Gem_level^h_gem))*init_Cells_Cycle_S_rel;
sn38_lesion:  -> L_sn38; k_f_sn38*(SN38_level^h_sn38/(K_Lsn38^h_sn38 + SN38_level^h_sn38) + k_lin_sn38*SN38_level)*(
  init_Cells_Cycle_S_rel + init_Cells_Cycle_G2_rel);
dox_repair: L_dox -> ; k_res_dox*L_dox;
gem_repair: L_gem -> ; k_res_gem*L_gem;
sn38_repair: L_sn38 -> ; k_res_sn38*L_sn38;
atm_act:  -> pATM; k_atm*Les_DSB;
atm_dephos: pATM -> ; pATM*(1/dephosphorylation_timescale + k_wip1_atm*Wip1);
chk2_act:  -> pChk2; k_chk2*pATM;
chk2_dephos: pChk2 -> ; pChk2*(1/dephosphorylation_timescale + k_wip1_chk2*Wip1);
chk1_act:  -> pChk1; k_chk1_gem*L_gem + k_chk1_sn38*L_sn38 + k_chk1_atm*pATM;
chk1_dephos: pChk1 -> ; pChk1*(1/dephosphorylation_timescale + k_wip1_chk1*Wip1);
dpk_act:  -> pDNAPK; k_dpk_dox*L_dox + k_dpk_sn38*L_sn38 + k_dpk_gem*L_gem;
dpk_dephos: pDNAPK -> ; pDNAPK/dephosphorylation_timescale;
h2ax_phos:  -> yH2AX; k_h2ax*Les_DSB;
h2ax_dephos: yH2AX -> ; yH2AX/dephosphorylation_timescale;
p53_phos:  -> pp53; k_syn53*Chk_sig;
p53_dephos: pp53 -> ; pp53*(1/dephosphorylation_timescale + k_wip1_p53*Wip1);
p53_syn:  -> tp53; k_syn53;
p53_deg: tp53 -> ; k_deg53*tp53/(1 + k_stab*Chk_sig);
p21_tx:  -> p21mRNA; k_tx_p21*pp53*inh_dox_tx;
p21_mrna_deg: p21mRNA -> ; k_deg_p21mrna*p21mRNA;
wip1_tx:  -> Wip1mRNA; k_tx_wip1*pp53*inh_dox_tx;
wip1_mrna_deg: Wip1mRNA -> ; k_deg_wip1*Wip1mRNA;
wip1_tl:  -> Wip1; k_tl_wip1*Wip1mRNA;
wip1_deg: Wip1 -> ; k_deg_wip1*Wip1;
p21_tl:  -> tp21; k_tl_p21*p21mRNA;
p21_deg: tp21 -> ; k_deg_p21*tp21;
```

```
growth:  -> Nc; r_growth*(1 - Nc/init_Space)*(1 - ArrestChk)*Nc;
lysis: Casp -> ; k_lyse*Casp*Dox_level^h_kill/(k_dox_kd^h_kill +
  Dox_level^h_kill);
casp_recover: Casp -> Nc; k_rec*Casp;
casp_commit: Nc -> C2; k_casp*gate_dox*pChk2*Nc + k_casp_late*
  ArrestChk*Nc;
casp_exec: C2 -> Casp; k_exec*C2;

// Species initializations:
L_dox = ;
L_gem = ;
L_sn38 = ;
pATM = ;
pChk2 = ;
pChk1 = ;
pDNAPK = ;
yH2AX = ;
pp53 = ;
tp53 = ;
p21mRNA = ;
Wip1mRNA = ;
Wip1 = ;
tp21 = ;
Nc = init_Cells;
Casp = ;
C2 = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_Cells = 1;
init_Space = 100;
init_Cells_Cycle_S_rel = 0.5;
init_Cells_Cycle_G2_rel = 0.4;
k_dox_kd = 0.356451133426244;
p_mrna_exp_inh_dox_kd = 0.356451179497662;
dephosphorylation_timescale = 24;
Dox_level = 0;
Gem_level = 0;
SN38_level = 0;
w_gem_les = ;
w_chk2_p53 = ;
h_casp = ;
k_dox_casp_kd = ;
h_tx = ;
w_ar_p21 = ;
K_ar_w = ;
k_f_dox = ;
h_dox = ;
K_Ldox = ;
k_f_gem = ;
h_gem = ;
K_Lgem = ;
k_f_sn38 = ;
h_sn38 = ;
K_Lsn38 = ;
k_lin_sn38 = ;
k_res_dox = ;
k_res_gem = ;
k_res_sn38 = ;
k_atm = ;
k_wip1_atm = ;
k_chk2 = ;
k_wip1_chk2 = ;
```

```
  k_chk1_gem = ;
  k_chk1_sn38 = ;
  k_chk1_atm = ;
  k_wip1_chk1 = ;
  k_dpk_dox = ;
  k_dpk_sn38 = ;
  k_dpk_gem = ;
  k_h2ax = ;
  k_syn53 = ;
  k_wip1_p53 = ;
  k_deg53 = ;
  k_stab = ;
  k_tx_p21 = ;
  k_deg_p21mrna = ;
  k_tx_wip1 = ;
  k_deg_wip1 = ;
  k_tl_wip1 = ;
  k_tl_p21 = ;
  k_deg_p21 = ;
  r_growth = ;
  k_lyse = ;
  h_kill = ;
  k_rec = ;
  k_casp = ;
  k_casp_late = ;
  k_exec = ;

  // Other declarations:
  var Les_DSB, Chk_sig, gate_dox, inh_dox_tx, ArrestChk,
    readout_yH2AX_au;
  var readout_pATM_au, readout_pChk1_au, readout_pChk2_au,
    readout_pDNAPK_au;
  var readout_pp53_au, readout_tp53_au, readout_tp21_au,
    readout_p21_mRNA_fold;
  var readout_Wip1_mRNA_fold, readout_Cells_count,
    readout_CellsCasp_count;
  const cell, init_Cells, init_Space, init_Cells_Cycle_S_rel,
    init_Cells_Cycle_G2_rel;
  const k_dox_kd, p_mrna_exp_inh_dox_kd, dephosphorylation_timescale,
    Dox_level;
  const Gem_level, SN38_level, w_gem_les, w_chk2_p53, h_casp,
    k_dox_casp_kd;
  const h_tx, w_ar_p21, K_ar_w, k_f_dox, h_dox, K_Ldox, k_f_gem, h_gem,
     K_Lgem;
  const k_f_sn38, h_sn38, K_Lsn38, k_lin_sn38, k_res_dox, k_res_gem,
    k_res_sn38;
  const k_atm, k_wip1_atm, k_chk2, k_wip1_chk2, k_chk1_gem, k_chk1_sn38
    , k_chk1_atm;
  const k_wip1_chk1, k_dpk_dox, k_dpk_sn38, k_dpk_gem, k_h2ax, k_syn53,
     k_wip1_p53;
  const k_deg53, k_stab, k_tx_p21, k_deg_p21mrna, k_tx_wip1, k_deg_wip1
    , k_tl_wip1;
  const k_tl_p21, k_deg_p21, r_growth, k_lyse, h_kill, k_rec, k_casp,
    k_casp_late;
  const k_exec;
end
```

Listing 108: Archived Antimony for `alkan_5`; task `56836792_4`; selected version 15

```
// Created by libAntimony v3.1.3
model *chemo_ddr()

  // Compartments and Species:
  compartment cell;
  species N_G1 in cell, N_S in cell, N_G2 in cell, C in cell, Bf in
    cell;
  species Bs in cell, Rs in cell, pATM in cell, pChk1 in cell, pChk2 in
     cell;
  species pDNAPK in cell, pp53 in cell, $tp53 in cell, $tp21 in cell,
    $M21 in cell;
  species yH2AX in cell, $MW in cell, Dox_level in cell, Gem_level in
    cell;
  species SN38_level in cell, Bp;

  // Assignment Rules:
  tp53 := 1 + w53*pp53;
  tp21 := M21;
  M21 := 1 + A21*pp53*inh_g*inh_p21;
  inh_g := 1/(1 + (Dox_level/k_inh_dox)^80);
  inh_p21 := 1/(1 + sqrt(Dox_level/p_mrna_exp_inh_dox_kd));
  MW := 1 + AW*pp53*inh_g;
  Ntot := N_G1 + N_S + N_G2;
  Sfrac := N_S/Ntot;
  g_dox_dmg := Dox_level/(Dox_level + k_dox_kd);
  gdose := Dox_level^n_dk/(Dox_level^n_dk + k_dox_kd^n_dk);
  g_p21 := 1/(1 + (tp21 - 1)/K_p21);
  g_chk := 1/(1 + ((pChk1 + pChk2)/K_chk)^n_chk);
  readout_Cells_count := Ntot + C;
  readout_CellsCasp_count := C;
  readout_pATM_au := pATM;
  readout_pChk1_au := pChk1;
  readout_pChk2_au := pChk2;
  readout_pDNAPK_au := pDNAPK;
  readout_pp53_au := pp53;
  readout_tp53_au := tp53 - 1;
  readout_tp21_au := tp21 - 1;
  readout_yH2AX_au := yH2AX;
  readout_p21_mRNA_fold := M21 - 1;
  readout_Wip1_mRNA_fold := MW - 1;
  h_dox_dmg := Dox_level^n_dm/(Dox_level^n_dm + Kd_dox_dmg^n_dm);
  h_sn_dmg := SN38_level/(SN38_level + Kd_sn_dmg);
  h_gem_dmg := Gem_level/(Gem_level + Kd_gem_dmg);
  h53 := pp53^50/(pp53^50 + K53^50);

  // Reactions:
  Bf_dox_form:  -> Bf; kBf_dox*h_dox_dmg;
  Bf_fromR:  -> Bf; kB_R*Rs;
  Bf_repair: Bf -> ; r_Bf*Bf;
  Bs_sn_form:  -> Bs; kBs_sn*h_sn_dmg*Sfrac;
  Bs_repair: Bs -> ; r_Bs*Bs;
  R_gem_form:  -> Rs; kR_gem*h_gem_dmg*Sfrac;
  R_decay: Rs -> ; r_R*Rs;
  pATM_act:  -> pATM; kATM_a*(Bf + wA*Bs)*(ATM_max - pATM);
  pATM_de: pATM -> ; pATM/dephosphorylation_timescale;
  pChk1_act:  -> pChk1; kC1_a*(Rs + wRB*(Bf + Bs))*(C1_max - pChk1);
  pChk1_de: pChk1 -> ; pChk1/dephosphorylation_timescale;
  pChk2_act:  -> pChk2; kC2_a*(Bf + wC2*Bs)*(C2_max - pChk2);
  pChk2_de: pChk2 -> ; pChk2/dephosphorylation_timescale;
  pDNAPK_act:  -> pDNAPK; kDPK_a*Bf*(DPK_max - pDNAPK);
  pDNAPK_de: pDNAPK -> ; pDNAPK/dephosphorylation_timescale;
  pp53_act:  -> pp53; kPP_a*(Bf + wA*Bp)*(PP_max - pp53);
  pp53_de: pp53 -> ; pp53/dephosphorylation_timescale;
```

```
yH2AX_act:  -> yH2AX; kY_a*(Bf + wY*Bs + wYR*Rs)*(Y_max - yH2AX);
yH2AX_de: yH2AX -> ; yH2AX/dephosphorylation_timescale;
G1S: N_G1 -> N_S; kG1S*g_p21*N_G1;
SG2: N_S -> N_G2; kSG2*g_chk*N_S;
div: N_G2 -> 2 N_G1; kdiv*g_chk*g_p21*(1 - Ntot/init_Space)*N_G2;
lysis: C -> ; k_lysis*C;
Bp_form_sn:  -> Bp; kBp_sn*h_sn_dmg*Sfrac;
Bp_decay: Bp -> ; r_Bp*Bp;
casp_rec: C -> N_G1; k_rec*(1 - gdose*h53)*C;
die_S: N_S -> C; k_die*gdose*h53*N_S*(1 - C/(f_casp*(Ntot + C)));
die_G2: N_G2 -> C; k_die*gdose*h53*N_G2*(1 - C/(f_casp*(Ntot + C)));

// Species initializations:
N_G1 = init_Cells*(1 - init_Cells_Cycle_S_rel -
  init_Cells_Cycle_G2_rel);
N_S = init_Cells*init_Cells_Cycle_S_rel;
N_G2 = init_Cells*init_Cells_Cycle_G2_rel;
C = 0;
Bf = 0;
Bs = 0;
Rs = 0;
pATM = 0;
pChk1 = 0;
pChk2 = 0;
pDNAPK = 0;
pp53 = 0;
yH2AX = 0;
Dox_level = ;
Gem_level = ;
SN38_level = ;
Bp = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_Cells = ;
init_Cells_Cycle_S_rel = ;
init_Cells_Cycle_G2_rel = ;
w53 = 2;
A21 = ;
AW = ;
kBf_dox = 2;
kB_R = 0.05;
r_Bf = 1.5;
kBs_sn = 0.1;
r_Bs = 0.02;
kR_gem = 30;
r_R = 2;
wA = 10;
wRB = 0.1;
wC2 = 0.05;
wY = 3;
wYR = 0.05;
kG1S = 0.3;
kSG2 = 0.065;
kdiv = 0.08;
K_p21 = 0.8;
K_chk = 0.3;
k_die = 0.07;
n_dk = 8;
k_lysis = 0.04;
k_dox_kd = ;
n_chk = ;
n_dm = ;
```

```
  Kd_dox_dmg = ;
  Kd_sn_dmg = ;
  Kd_gem_dmg = ;
  K53 = ;
  k_inh_dox = ;
  p_mrna_exp_inh_dox_kd = ;
  kATM_a = ;
  ATM_max = ;
  dephosphorylation_timescale = ;
  kC1_a = ;
  C1_max = ;
  kC2_a = ;
  C2_max = ;
  kDPK_a = ;
  DPK_max = ;
  kPP_a = ;
  PP_max = ;
  kY_a = ;
  Y_max = ;
  init_Space = ;
  kBp_sn = ;
  r_Bp = ;
  k_rec = ;
  f_casp = ;

  // Other declarations:
  var inh_g, inh_p21, Ntot, Sfrac, g_dox_dmg, gdose, g_p21, g_chk,
    readout_Cells_count;
  var readout_CellsCasp_count, readout_pATM_au, readout_pChk1_au,
    readout_pChk2_au;
  var readout_pDNAPK_au, readout_pp53_au, readout_tp53_au,
    readout_tp21_au;
  var readout_yH2AX_au, readout_p21_mRNA_fold, readout_Wip1_mRNA_fold,
    h_dox_dmg;
  var h_sn_dmg, h_gem_dmg, h53;
  const cell, init_Cells, init_Cells_Cycle_S_rel,
    init_Cells_Cycle_G2_rel;
  const w53, A21, AW, kBf_dox, kB_R, r_Bf, kBs_sn, r_Bs, kR_gem, r_R,
    wA;
  const wRB, wC2, wY, wYR, kG1S, kSG2, kdiv, K_p21, K_chk, k_die, n_dk,
     k_lysis;
  const k_dox_kd, n_chk, n_dm, Kd_dox_dmg, Kd_sn_dmg, Kd_gem_dmg, K53,
    k_inh_dox;
  const p_mrna_exp_inh_dox_kd, kATM_a, ATM_max,
    dephosphorylation_timescale;
  const kC1_a, C1_max, kC2_a, C2_max, kDPK_a, DPK_max, kPP_a, PP_max,
    kY_a;
  const Y_max, init_Space, kBp_sn, r_Bp, k_rec, f_casp;
end
```

H.17 LUCARELLI

Listing 109: Archived Antimony for `lucarelli_1`; task `56836792_10`; selected version 9

```
// Created by libAntimony v3.1.3
model *tgfb_smad_v1()

  // Compartments and Species:
  compartment cell;
  species TGFb in cell, Rrec in cell, RL in cell, S2 in cell, pS2 in
    cell;
  species ppS2 in cell, S3 in cell, pS3 in cell, ppS3 in cell, S4 in
    cell;
```

```
species B22 in cell, B23 in cell, B33 in cell, T22 in cell, T23 in cell;
species T33 in cell, G_bmp4 in cell, G_cxcl15 in cell, G_dnmt3a in cell;
species G_dusp5 in cell, G_jun in cell, G_klf10 in cell, G_pdk4 in cell;
species G_ski in cell, G_skil in cell, G_smad7 in cell, G_sox4 in cell;
species G_tgfa in cell;

// Assignment Rules:
Sig := T22 + T23 + T33 + wB*(B22 + B23 + B33);
S2fam := S2 + pS2 + ppS2 + 2*B22 + B23 + 2*T22 + T23;
S3fam := S3 + pS3 + ppS3 + 2*B33 + B23 + 2*T33 + T23;
S4inR := B22 + B23 + B33 + T22 + T23 + T33;
S4inS2IP := B22 + T22 + B23 + T23;
S3inS2IP := B23 + T23;
S2inS3IP := B23 + T23;
S4inS3IP := B33 + T33 + B23 + T23;
S2cS4 := 2*B22 + B23 + 2*T22 + T23;
S3cS4 := 2*B33 + B23 + 2*T33 + T23;
Dpm := S2fam + w3pm*S3fam;
D23 := S2fam + w3pm*S3fam + w4_23*S4inR;
D2IP := S2fam + w3_2IP*S3inS2IP + w4_2IP*S4inS2IP;
D3IP := S3fam + w2_3IP*S2inS3IP + w4_3IP*S4inS3IP;
D4IP := S4 + S4inR + w2_S4IP*S2cS4 + w3_S4IP*S3cS4;
readout_Bmp4 := G_bmp4;
readout_Cxcl15 := G_cxcl15;
readout_Dnmt3a := G_dnmt3a;
readout_Dusp5 := G_dusp5;
readout_Jun := G_jun;
readout_Klf10 := G_klf10;
readout_Pdk4 := G_pdk4;
readout_Ski := G_ski;
readout_Skil := G_skil;
readout_Smad7 := G_smad7;
readout_Sox4 := G_sox4;
readout_Tgfa := G_tgfa;
readout_TGFbR := Rrec;
readout_S23IP_S2 := (S2 + 2*B22 + B23)/S2fam;
readout_S23IP_pS2 := pS2/S2fam;
readout_S23IP_ppS2 := (ppS2 + 2*T22 + T23)/S2fam;
readout_S23IP_S3 := (S3 + 2*B33 + B23)/S3fam;
readout_S23IP_pS3 := pS3/S3fam;
readout_S23IP_ppS3 := (ppS3 + 2*T33 + T23)/S3fam;
readout_S23IP_tS2_perMol := S2fam/Dpm;
readout_S23IP_tS3_perMol := w3pm*S3fam/Dpm;
readout_S23IP_tS4 := w4_23*S4inR/D23;
readout_S23IP_tS2_raw := S2fam;
readout_S2IP_S2 := (S2 + 2*B22 + B23)/S2fam;
readout_S2IP_pS2 := pS2/S2fam;
readout_S2IP_ppS2 := (ppS2 + 2*T22 + T23)/S2fam;
readout_S2IP_tS2 := S2fam/D2IP;
readout_S2IP_tS3 := w3_2IP*S3inS2IP/D2IP;
readout_S2IP_tS4 := w4_2IP*S4inS2IP/D2IP;
readout_S2IP_S3 := B23/S3inS2IP;
readout_S2IP_pS3 := 0;
readout_S2IP_ppS3 := T23/S3inS2IP;
readout_S3IP_S3 := (S3 + 2*B33 + B23)/S3fam;
readout_S3IP_pS3 := pS3/S3fam;
readout_S3IP_ppS3 := (ppS3 + 2*T33 + T23)/S3fam;
readout_S3IP_tS3 := S3fam/D3IP;
readout_S3IP_tS2 := w2_3IP*S2inS3IP/D3IP;
readout_S3IP_tS4 := w4_3IP*S4inS3IP/D3IP;
readout_S3IP_S2 := B23/S2inS3IP;
```

```
readout_S3IP_pS2 := 0;
readout_S3IP_ppS2 := T23/S2inS3IP;
readout_S4IP_tS4_perMol := (S4 + S4inR)/D4IP;
readout_S4IP_tS2 := w2_S4IP*S2cS4/D4IP;
readout_S4IP_tS3 := w3_S4IP*S3cS4/D4IP;
readout_S4IP_tS4_raw := S4 + S4inR;

// Reactions:
rl_bind: TGFb + Rrec -> RL; konRL*TGFb*Rrec;
rl_unbind: RL -> TGFb + Rrec; koffRL*RL;
rl_int: RL -> ; kinRL*RL;
s2_p1: S2 -> pS2; kp1_S2*(RL + Rbas)*S2;
s2_p2: pS2 -> ppS2; kp2_S2*(RL + Rbas)*pS2;
s2_dp1: pS2 -> S2; kdp1_S2*pS2;
s2_dp2: ppS2 -> pS2; kdp2_S2*ppS2;
s3_p1: S3 -> pS3; kp1_S3*(RL + Rbas)*S3;
s3_p2: pS3 -> ppS3; kp2_S3*(RL + Rbas)*pS3;
s3_dp1: pS3 -> S3; kdp1_S3*pS3;
s3_dp2: ppS3 -> pS3; kdp2_S3*ppS3;
t22_form: 2 ppS2 + S4 -> T22; konT22*ppS2*ppS2*S4;
t23_form: ppS2 + ppS3 + S4 -> T23; konT23*ppS2*ppS3*S4;
t33_form: 2 ppS3 + S4 -> T33; konT33*ppS3*ppS3*S4;
g_bmp4:  -> G_bmp4; Kp_bmp4*(1 + Amp_bmp4*Sig^m_bmp4);
g_bmp4_deg: G_bmp4 -> ; Kdeg_bmp4*G_bmp4;
g_cxcl15:  -> G_cxcl15; Kp_cxcl15/(1 + Rep_cxcl15*Sig^m_cxcl15);
g_cxcl15_deg: G_cxcl15 -> ; Kdeg_cxcl15*G_cxcl15;
g_dnmt3a:  -> G_dnmt3a; Kp_dnmt3a*(1 + Amp_dnmt3a*Sig^m_dnmt3a);
g_dnmt3a_deg: G_dnmt3a -> ; Kdeg_dnmt3a*G_dnmt3a;
g_dusp5:  -> G_dusp5; Kp_dusp5/(1 + Rep_dusp5*Sig^m_dusp5);
g_dusp5_deg: G_dusp5 -> ; Kdeg_dusp5*G_dusp5;
g_jun:  -> G_jun; Kp_jun*(1 + Amp_jun*Sig^m_jun);
g_jun_deg: G_jun -> ; Kdeg_jun*G_jun;
g_klf10:  -> G_klf10; Kp_klf10*(1 + Amp_klf10*Sig^m_klf10);
g_klf10_deg: G_klf10 -> ; Kdeg_klf10*G_klf10;
g_pdk4:  -> G_pdk4; Kp_pdk4/(1 + Rep_pdk4*Sig^m_pdk4);
g_pdk4_deg: G_pdk4 -> ; Kdeg_pdk4*G_pdk4;
g_ski:  -> G_ski; Kp_ski*(1 + Amp_ski*Sig^m_ski);
g_ski_deg: G_ski -> ; Kdeg_ski*G_ski;
g_skil:  -> G_skil; Kp_skil*(1 + Amp_skil*Sig^m_skil);
g_skil_deg: G_skil -> ; Kdeg_skil*G_skil;
g_smad7:  -> G_smad7; Kp_smad7*(1 + Amp_smad7*Sig^m_smad7);
g_smad7_deg: G_smad7 -> ; Kdeg_smad7*G_smad7;
g_sox4:  -> G_sox4; Kp_sox4*(1 + Amp_sox4*Sig^m_sox4);
g_sox4_deg: G_sox4 -> ; Kdeg_sox4*G_sox4;
g_tgfa:  -> G_tgfa; Kp_tgfa/(1 + Rep_tgfa*Sig^m_tgfa);
g_tgfa_deg: G_tgfa -> ; Kdeg_tgfa*G_tgfa;
t22_release: T22 -> 2 S2 + S4; kdT22*T22;
t23_release: T23 -> S2 + S3 + S4; kdT23*T23;
t33_release: T33 -> 2 S3 + S4; kdT33*T33;

// Species initializations:
TGFb = init_TGFb;
Rrec = Rrec0;
RL = 0;
S2 = init_S2;
pS2 = 0;
ppS2 = 0;
S3 = init_S3;
pS3 = 0;
ppS3 = 0;
S4 = init_S4;
B22 = B22_0;
B23 = B23_0;
B33 = B33_0;
T22 = 0;
```

```
T23 = 0;
T33 = 0;
G_bmp4 = Kp_bmp4/Kdeg_bmp4;
G_cxcl15 = Kp_cxcl15/Kdeg_cxcl15;
G_dnmt3a = Kp_dnmt3a/Kdeg_dnmt3a;
G_dusp5 = Kp_dusp5/Kdeg_dusp5;
G_jun = Kp_jun/Kdeg_jun;
G_klf10 = Kp_klf10/Kdeg_klf10;
G_pdk4 = Kp_pdk4/Kdeg_pdk4;
G_ski = Kp_ski/Kdeg_ski;
G_skil = Kp_skil/Kdeg_skil;
G_smad7 = Kp_smad7/Kdeg_smad7;
G_sox4 = Kp_sox4/Kdeg_sox4;
G_tgfa = Kp_tgfa/Kdeg_tgfa;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_TGFb = 1;
Rrec0 = 1.85;
init_S2 = 142.34;
init_S3 = 14.234;
init_S4 = 69.47;
B22_0 = 5;
B23_0 = 1;
B33_0 = 0.4;
Kp_bmp4 = 0.065;
Kdeg_bmp4 = 0.005;
Kp_cxcl15 = 0.19;
Kdeg_cxcl15 = 0.008;
Kp_dnmt3a = 0.067;
Kdeg_dnmt3a = 0.005;
Kp_dusp5 = 0.12;
Kdeg_dusp5 = 0.008;
Kp_jun = 0.063;
Kdeg_jun = 0.005;
Kp_klf10 = 0.044;
Kdeg_klf10 = 0.005;
Kp_pdk4 = 0.073;
Kdeg_pdk4 = 0.008;
Kp_ski = 0.067;
Kdeg_ski = 0.005;
Kp_skil = 0.026;
Kdeg_skil = 0.002;
Kp_smad7 = 0.042;
Kdeg_smad7 = 0.005;
Kp_sox4 = 0.047;
Kdeg_sox4 = 0.005;
Kp_tgfa = 0.086;
Kdeg_tgfa = 0.008;
S2tot = 142.34;
S3tot = 14.234;
S4tot = 69.47;
konRL = 2;
koffRL = 0.8;
kinRL = 0.02;
Rbas = 0.02;
kp1_S2 = 0.03;
kp2_S2 = 0.3;
kdp1_S2 = 0.03;
kdp2_S2 = 0.01;
kp1_S3 = 0.03;
kp2_S3 = 0.15;
kdp1_S3 = 0.02;
```

```
kdp2_S3 = 0.008;
konT22 = 3e-06;
kdT22 = 0.003;
konT23 = 3e-06;
kdT23 = 0.003;
konT33 = 3e-06;
kdT33 = 0.003;
w3pm = 1.2;
w4_23 = 0.2;
w3_2IP = 1;
w4_2IP = 0.5;
w2_3IP = 1;
w4_3IP = 0.1;
w2_S4IP = 1;
w3_S4IP = 1;
wB = 0.1;
Amp_bmp4 = 0.1;
Rep_cxcl15 = 0.12;
Amp_dnmt3a = 0.01;
Rep_dusp5 = 0.066;
Amp_jun = 3.5;
Amp_klf10 = 27;
Rep_pdk4 = 0.062;
Amp_ski = 0.12;
Amp_skil = 180000;
Amp_smad7 = 270;
Amp_sox4 = 0.045;
Rep_tgfa = 0.054;
m_bmp4 = ;
m_cxcl15 = ;
m_dnmt3a = ;
m_dusp5 = ;
m_jun = ;
m_klf10 = ;
m_pdk4 = ;
m_ski = ;
m_skil = ;
m_smad7 = ;
m_sox4 = ;
m_tgfa = ;

// Other declarations:
var Sig, S2fam, S3fam, S4inR, S4inS2IP, S3inS2IP, S2inS3IP, S4inS3IP,
   S2cS4;
var S3cS4, Dpm, D23, D2IP, D3IP, D4IP, readout_Bmp4, readout_Cxcl15,
  readout_Dnmt3a;
var readout_Dusp5, readout_Jun, readout_Klf10, readout_Pdk4,
  readout_Ski;
var readout_Skil, readout_Smad7, readout_Sox4, readout_Tgfa,
  readout_TGFbR;
var readout_S23IP_S2, readout_S23IP_pS2, readout_S23IP_ppS2,
  readout_S23IP_S3;
var readout_S23IP_pS3, readout_S23IP_ppS3, readout_S23IP_tS2_perMol,
  readout_S23IP_tS3_perMol;
var readout_S23IP_tS4, readout_S23IP_tS2_raw, readout_S2IP_S2,
  readout_S2IP_pS2;
var readout_S2IP_ppS2, readout_S2IP_tS2, readout_S2IP_tS3,
  readout_S2IP_tS4;
var readout_S2IP_S3, readout_S2IP_pS3, readout_S2IP_ppS3,
  readout_S3IP_S3;
var readout_S3IP_pS3, readout_S3IP_ppS3, readout_S3IP_tS3,
  readout_S3IP_tS2;
var readout_S3IP_tS4, readout_S3IP_S2, readout_S3IP_pS2,
  readout_S3IP_ppS2;
```

```
  var readout_S4IP_tS4_perMol, readout_S4IP_tS2, readout_S4IP_tS3,
    readout_S4IP_tS4_raw;
  const cell, init_TGFb, Rrec0, init_S2, init_S3, init_S4, B22_0, B23_0
    , B33_0;
  const Kp_bmp4, Kdeg_bmp4, Kp_cxcl15, Kdeg_cxcl15, Kp_dnmt3a,
    Kdeg_dnmt3a;
  const Kp_dusp5, Kdeg_dusp5, Kp_jun, Kdeg_jun, Kp_klf10, Kdeg_klf10,
    Kp_pdk4;
  const Kdeg_pdk4, Kp_ski, Kdeg_ski, Kp_skil, Kdeg_skil, Kp_smad7,
    Kdeg_smad7;
  const Kp_sox4, Kdeg_sox4, Kp_tgfa, Kdeg_tgfa, S2tot, S3tot, S4tot,
    konRL;
  const koffRL, kinRL, Rbas, kp1_S2, kp2_S2, kdp1_S2, kdp2_S2, kp1_S3,
    kp2_S3;
  const kdp1_S3, kdp2_S3, konT22, kdT22, konT23, kdT23, konT33, kdT33,
    w3pm;
  const w4_23, w3_2IP, w4_2IP, w2_3IP, w4_3IP, w2_S4IP, w3_S4IP, wB,
    Amp_bmp4;
  const Rep_cxcl15, Amp_dnmt3a, Rep_dusp5, Amp_jun, Amp_klf10, Rep_pdk4
    , Amp_ski;
  const Amp_skil, Amp_smad7, Amp_sox4, Rep_tgfa, m_bmp4, m_cxcl15,
    m_dnmt3a;
  const m_dusp5, m_jun, m_klf10, m_pdk4, m_ski, m_skil, m_smad7, m_sox4
    , m_tgfa;
end
```

Listing 110: Archived Antimony for `lucarelli_2`; task `56836792_11`; selected version 25

```
// Created by libAntimony v3.1.3
model *tgfb_smad_hepa16()

  // Compartments and Species:
  compartment cell;
  species R in cell, S2u in cell, S2p in cell, S2pp in cell, S3u in
    cell;
  species S3p in cell, S3pp in cell, S4 in cell, $D2 in cell, $D3 in
    cell;
  species MX in cell, mBmp4 in cell, mCxcl15 in cell, mDnmt3a in cell,
    mDusp5 in cell;
  species mJun in cell, mKlf10 in cell, mPdk4 in cell, mSki in cell,
    mSkil in cell;
  species mSmad7 in cell, mSox4 in cell, mTgfa in cell, R_T, U, P, W;

  // Assignment Rules:
  D2 := S2pp*S4/Kd2;
  D3 := S3pp*S4/Kd3;
  ActR := R_T/(Kd_L + R_T);
  Bxc := (init_S2 + init_S3 - S2tot - S3tot)/(S2tot + S3tot);
  tS2 := S2u + S2p + S2pp + D2 + MX;
  tS3 := S3u + S3p + S3pp + D3 + MX;
  P2 := S2pp + D2 + MX;
  P3 := S3pp + D3 + MX;
  S4c := D2 + D3;
  Xdrv := S4c/(S4 + S4c);
  Dr := Xdrv/(KX + Xdrv);
  readout_S23IP_S2 := S2u/(S2u + e_p2*S2p + e_pp2*(S2pp + D2 + MX));
  readout_S23IP_pS2 := e_p2*S2p/(S2u + e_p2*S2p + e_pp2*(S2pp + D2 + MX
    ));
  readout_S23IP_ppS2 := e_pp2*(S2pp + D2 + MX)/(S2u + e_p2*S2p + e_pp2
    *(S2pp + D2 + MX));
  readout_S23IP_S3 := S3u/(S3u + e_p3*S3p + e_pp3*(S3pp + D3 + MX));
  readout_S23IP_pS3 := e_p3*S3p/(S3u + e_p3*S3p + e_pp3*(S3pp + D3 + MX
    ));
  readout_S23IP_ppS3 := e_pp3*(S3pp + D3 + MX)/(S3u + e_p3*S3p + e_pp3
    *(S3pp + D3 + MX));
```

```
readout_S23IP_tS2_perMol := tS2/(tS2 + tS3);
readout_S23IP_tS3_perMol := tS3/(tS2 + tS3);
readout_S23IP_tS2_raw := tS2;
readout_S23IP_tS2 := tS2/(tS2 + tS3 + S4c);
readout_S23IP_tS3 := tS3/(tS2 + tS3 + S4c);
readout_S23IP_tS4 := S4c/(tS2 + tS3 + S4c);
readout_S2IP_S2 := S2u/(S2u + f_p2*S2p + f_pp2*(S2pp + D2 + MX));
readout_S2IP_pS2 := f_p2*S2p/(S2u + f_p2*S2p + f_pp2*(S2pp + D2 + MX));
readout_S2IP_ppS2 := f_pp2*(S2pp + D2 + MX)/(S2u + f_p2*S2p + f_pp2*(S2pp + D2 + MX));
readout_S2IP_tS2 := tS2/(tS2 + MX + D2);
readout_S2IP_tS3 := MX/(tS2 + MX + D2);
readout_S2IP_tS4 := D2/(tS2 + MX + D2);
readout_S2IP_S3 := 0;
readout_S2IP_pS3 := 0*S3p/tS3;
readout_S2IP_ppS3 := MX/(MX + 1e-6*tS3);
readout_S3IP_S2 := 0*S2u/tS2;
readout_S3IP_pS2 := 0*S2p/tS2;
readout_S3IP_ppS2 := MX/(MX + 1e-6*tS2);
readout_S3IP_S3 := S3u/tS3;
readout_S3IP_pS3 := S3p/tS3;
readout_S3IP_ppS3 := P3/tS3;
readout_S3IP_tS2 := MX/(tS3 + MX + D3);
readout_S3IP_tS3 := tS3/(tS3 + MX + D3);
readout_S3IP_tS4 := D3/(tS3 + MX + D3);
readout_S4IP_tS2 := D2/(S4 + 2*D2 + 2*D3);
readout_S4IP_tS3 := D3/(S4 + 2*D2 + 2*D3);
readout_S4IP_tS4_perMol := (S4 + D2 + D3)/(S4 + 2*D2 + 2*D3);
readout_S4IP_tS4_raw := S4 + D2 + D3;
readout_TGFbR := R + R_T;
readout_Bmp4 := mBmp4;
readout_Cxcl15 := mCxcl15;
readout_Dnmt3a := mDnmt3a;
readout_Dusp5 := mDusp5;
readout_Jun := mJun;
readout_Klf10 := mKlf10;
readout_Pdk4 := mPdk4;
readout_Ski := mSki;
readout_Skil := mSkil;
readout_Smad7 := mSmad7;
readout_Sox4 := mSox4;
readout_Tgfa := mTgfa;

// Reactions:
p1_S2: S2u -> S2p; kp1_S2*ActR*S2u;
p2_S2: S2p -> S2pp; kp2_S2*ActR*S2p;
dp2_S2: S2pp -> S2p; kdp2_S2*S2pp;
dp1_S2: S2p -> S2u; kdp1_S2*S2p;
p1_S3: S3u -> S3p; kp1_S3*ActR*S3u;
p2_S3: S3p -> S3pp; kp2_S3*ActR*S3p;
dp2_S3: S3pp -> S3p; kdr3*S3pp;
dp1_S3: S3p -> S3u; kdp1_S3*S3p;
mx_form: S2pp + S3pp -> MX; konX*S2pp*S3pp;
mx_diss: MX -> S2pp + S3pp; koffX*MX;
gBmp4_syn:  -> mBmp4; kdeg_Bmp4*(1 + amp_Bmp4*(1 + cB_Bmp4*Bxc)*ActR)*(1 + b_Bmp4*(1 + cB_Bmp4*Bxc)*U);
gBmp4_deg: mBmp4 -> ; kdeg_Bmp4*mBmp4;
gCxcl15_syn:  -> mCxcl15; (kdeg_Cxcl15/(1 + rep_Cxcl15*P))*(1 + b_Cxcl15*U);
gCxcl15_deg: mCxcl15 -> ; kdeg_Cxcl15*mCxcl15;
gDnmt3a_syn:  -> mDnmt3a; kdeg_Dnmt3a*(1 + amp_Dnmt3a*P)*(1 + b_Dnmt3a*U);
gDnmt3a_deg: mDnmt3a -> ; kdeg_Dnmt3a*mDnmt3a;
```

```
gDusp5_syn:  -> mDusp5; (kdeg_Dusp5/(1 + rep_Dusp5*Dr))*(1 + b_Dusp5*U);
gDusp5_deg: mDusp5 -> ; kdeg_Dusp5*mDusp5;
gJun_syn:  -> mJun; kdeg_Jun*(1 + amp_Jun*(ActR + W))*(1 + b_Jun*U);
gJun_deg: mJun -> ; kdeg_Jun*mJun;
gKlf10_syn:  -> mKlf10; kdeg_Klf10*(1 + amp_Klf10*(ActR + W));
gKlf10_deg: mKlf10 -> ; kdeg_Klf10*mKlf10;
gPdk4_syn:  -> mPdk4; kdeg_Pdk4/(1 + rep_Pdk4*Dr);
gPdk4_deg: mPdk4 -> ; kdeg_Pdk4*mPdk4;
gSki_syn:  -> mSki; kdeg_Ski*(1 + amp_Ski*Dr)*(1 + b_Ski*U);
gSki_deg: mSki -> ; kdeg_Ski*mSki;
gSkil_syn:  -> mSkil; kdeg_Skil*(1 + amp_Skil*Dr + bSkil*Bxc)*(1 + b_Skil*U);
gSkil_deg: mSkil -> ; kdeg_Skil*mSkil;
gSmad7_syn:  -> mSmad7; kdeg_Smad7*(1 + amp_Smad7*(ActR + W));
gSmad7_deg: mSmad7 -> ; kdeg_Smad7*mSmad7;
gSox4_syn:  -> mSox4; kdeg_Sox4*(1 + amp_Sox4*P)*(1 + b_Sox4*U);
gSox4_deg: mSox4 -> ; kdeg_Sox4*mSox4;
gTgfa_syn:  -> mTgfa; kdeg_Tgfa/(1 + rep_Tgfa*P);
gTgfa_deg: mTgfa -> ; kdeg_Tgfa*mTgfa;
p_form:  -> P; kdP*Xdrv;
p_deg: P -> ; kdP*P;
u_form:  -> U; kU*(1 - U);
u_deg: U -> ; kU*U;
rt_form: R -> R_T; kaL*init_TGFb*R;
rt_loss: R_T -> ; k_int*R_T;
w_form:  -> W; kdW*Xdrv;
w_deg: W -> ; kdW*W;

// Species initializations:
R = R_tot;
S2u = init_S2;
S2p = 0;
S2pp = 0;
S3u = init_S3;
S3p = 0;
S3pp = 0;
S4 = init_S4;
MX = 0;
mBmp4 = mBmp4_init;
mCxcl15 = mCxcl15_init;
mDnmt3a = mDnmt3a_init;
mDusp5 = mDusp5_init;
mJun = mJun_init;
mKlf10 = mKlf10_init;
mPdk4 = mPdk4_init;
mSki = mSki_init;
mSkil = 1 + bSkil*(init_S2 + init_S3 - S2tot - S3tot)/(S2tot + S3tot);
mSmad7 = mSmad7_init;
mSox4 = mSox4_init;
mTgfa = mTgfa_init;
R_T = ;
U = ;
P = ;
W = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
R_tot = 1.85;
init_S2 = 142.34;
init_S3 = 14.234;
init_S4 = 69.47;
```

```
Kd2 = ;
Kd3 = ;
mBmp4_init = 1;
mCxcl15_init = 1;
mDnmt3a_init = 1;
mDusp5_init = 1;
mJun_init = 1;
mKlf10_init = 1;
mPdk4_init = 1;
mSki_init = 1;
bSkil = 1;
S2tot = 142.34;
S3tot = 16.09;
mSmad7_init = 1;
mSox4_init = 1;
mTgfa_init = 1;
init_TGFb = 0;
S4tot = 69.47;
Kd_L = 0.3;
k_int = 0.006;
kp1_S2 = 0.04;
kp2_S2 = 0.15;
kdp1_S2 = 0.01;
kdp2_S2 = 0.006;
kp1_S3 = 0.04;
kp2_S3 = 0.15;
kdp1_S3 = 0.008;
konX = 0.0003;
koffX = 0.02;
amp_Bmp4 = 20;
amp_Dnmt3a = 10;
amp_Jun = 15;
amp_Klf10 = 40;
amp_Ski = 15;
amp_Skil = 40;
amp_Smad7 = 35;
amp_Sox4 = 15;
rep_Cxcl15 = 3.4;
rep_Dusp5 = 5.2;
rep_Pdk4 = 10.8;
rep_Tgfa = 4;
kdeg_Bmp4 = 0.002;
kdeg_Cxcl15 = 0.004;
kdeg_Dnmt3a = 0.004;
kdeg_Dusp5 = 0.005;
kdeg_Jun = 0.008;
kdeg_Klf10 = 0.02;
kdeg_Pdk4 = 0.006;
kdeg_Ski = 0.004;
kdeg_Skil = 0.003;
kdeg_Smad7 = 0.01;
kdeg_Sox4 = 0.002;
kdeg_Tgfa = 0.004;
KX = ;
e_p2 = ;
e_pp2 = ;
e_p3 = ;
e_pp3 = ;
f_p2 = ;
f_pp2 = ;
kdr3 = ;
cB_Bmp4 = ;
b_Bmp4 = ;
b_Cxcl15 = ;
b_Dnmt3a = ;
```

```
  b_Dusp5 = ;
  b_Jun = ;
  b_Ski = ;
  b_Skil = ;
  b_Sox4 = ;
  kdP = ;
  kU = ;
  kaL = ;
  kdW = ;

  // Other declarations:
  var ActR, Bxc, tS2, tS3, P2, P3, S4c, Xdrv, Dr, readout_S23IP_S2,
    readout_S23IP_pS2;
  var readout_S23IP_ppS2, readout_S23IP_S3, readout_S23IP_pS3,
    readout_S23IP_ppS3;
  var readout_S23IP_tS2_perMol, readout_S23IP_tS3_perMol,
    readout_S23IP_tS2_raw;
  var readout_S23IP_tS2, readout_S23IP_tS3, readout_S23IP_tS4,
    readout_S2IP_S2;
  var readout_S2IP_pS2, readout_S2IP_ppS2, readout_S2IP_tS2,
    readout_S2IP_tS3;
  var readout_S2IP_tS4, readout_S2IP_S3, readout_S2IP_pS3,
    readout_S2IP_ppS3;
  var readout_S3IP_S2, readout_S3IP_pS2, readout_S3IP_ppS2,
    readout_S3IP_S3;
  var readout_S3IP_pS3, readout_S3IP_ppS3, readout_S3IP_tS2,
    readout_S3IP_tS3;
  var readout_S3IP_tS4, readout_S4IP_tS2, readout_S4IP_tS3,
    readout_S4IP_tS4_perMol;
  var readout_S4IP_tS4_raw, readout_TGFbR, readout_Bmp4, readout_Cxcl15
    , readout_Dnmt3a;
  var readout_Dusp5, readout_Jun, readout_Klf10, readout_Pdk4,
    readout_Ski;
  var readout_Skil, readout_Smad7, readout_Sox4, readout_Tgfa;
  const cell, R_tot, init_S2, init_S3, init_S4, Kd2, Kd3, mBmp4_init,
    mCxcl15_init;
  const mDnmt3a_init, mDusp5_init, mJun_init, mKlf10_init, mPdk4_init,
    mSki_init;
  const bSkil, S2tot, S3tot, mSmad7_init, mSox4_init, mTgfa_init,
    init_TGFb;
  const S4tot, Kd_L, k_int, kp1_S2, kp2_S2, kdp1_S2, kdp2_S2, kp1_S3,
    kp2_S3;
  const kdp1_S3, konX, koffX, amp_Bmp4, amp_Dnmt3a, amp_Jun, amp_Klf10,
     amp_Ski;
  const amp_Skil, amp_Smad7, amp_Sox4, rep_Cxcl15, rep_Dusp5, rep_Pdk4,
     rep_Tgfa;
  const kdeg_Bmp4, kdeg_Cxcl15, kdeg_Dnmt3a, kdeg_Dusp5, kdeg_Jun,
    kdeg_Klf10;
  const kdeg_Pdk4, kdeg_Ski, kdeg_Skil, kdeg_Smad7, kdeg_Sox4,
    kdeg_Tgfa;
  const KX, e_p2, e_pp2, e_p3, e_pp3, f_p2, f_pp2, kdr3, cB_Bmp4,
    b_Bmp4;
  const b_Cxcl15, b_Dnmt3a, b_Dusp5, b_Jun, b_Ski, b_Skil, b_Sox4, kdP,
     kU;
  const kaL, kdW;
end
```

Listing 111: Archived Antimony for `lucarelli_3`; task `56836792_12`; selected version 14

```
// Created by libAntimony v3.1.3
model *TGFb_SMAD_hepa16()

  // Compartments and Species:
  compartment cell;
  species RA in cell, RL in cell, S2u in cell, S2m in cell, S2d in cell
    , S3u in cell;
  species S3m in cell, S3d in cell, S4 in cell, Cd23 in cell, D24 in
    cell;
  species D34 in cell, G_Klf10 in cell, M, A2, G_Smad7 in cell, G_Skil
    in cell;
  species A, G_Jun in cell, G_Ski in cell, G_Sox4 in cell, G_Dnmt3a in
    cell;
  species G_Bmp4 in cell, G_Tgfa in cell, G_Dusp5 in cell, G_Pdk4 in
    cell;
  species G_Cxcl15 in cell;

  // Assignment Rules:
  B24 := Kb24*S2u*S4;
  H23u := Kh23u*S2u*S3u;
  Cact := S2d + S3d + 2*Cd23;
  S2all := S2u + S2m + S2d + B24 + H23u + Cd23 + D24;
  S3all := S3u + S3m + S3d + H23u + Cd23 + D34;
  S4pull := S4 + 2*(B24 + D24 + D34);
  readout_S2IP_S2 := (S2u + B24 + H23u)/S2all;
  readout_S2IP_pS2 := S2m/S2all;
  readout_S2IP_ppS2 := (S2d + Cd23 + D24)/S2all;
  readout_S2IP_tS2 := S2all/(S2all + H23u + Cd23 + B24 + D24);
  readout_S2IP_tS3 := (H23u + Cd23)/(S2all + H23u + Cd23 + B24 + D24);
  readout_S2IP_tS4 := (B24 + D24)/(S2all + H23u + Cd23 + B24 + D24);
  readout_S2IP_S3 := 100*H23u/(H23u + Cd23);
  readout_S2IP_pS3 := 0;
  readout_S2IP_ppS3 := 100*Cd23/(H23u + Cd23);
  readout_S3IP_S3 := (S3u + H23u)/S3all;
  readout_S3IP_pS3 := S3m/S3all;
  readout_S3IP_ppS3 := (S3d + Cd23 + D34)/S3all;
  readout_S3IP_S2 := 100*H23u/(H23u + Cd23);
  readout_S3IP_pS2 := 0;
  readout_S3IP_ppS2 := 100*Cd23/(H23u + Cd23);
  readout_S3IP_tS2 := (H23u + Cd23)/(S3all + H23u + Cd23 + D34);
  readout_S3IP_tS3 := S3all/(S3all + H23u + Cd23 + D34);
  readout_S3IP_tS4 := D34/(S3all + H23u + Cd23 + D34);
  readout_S23IP_S2 := (S2u + B24 + H23u)/S2all;
  readout_S23IP_pS2 := S2m/S2all;
  readout_S23IP_ppS2 := (S2d + Cd23 + D24)/S2all;
  readout_S23IP_S3 := (S3u + H23u)/S3all;
  readout_S23IP_pS3 := S3m/S3all;
  readout_S23IP_ppS3 := (S3d + Cd23 + D34)/S3all;
  readout_S23IP_tS2_perMol := S2all/(S2all + S3all);
  readout_S23IP_tS3_perMol := S3all/(S2all + S3all);
  readout_S23IP_tS4 := (B24 + D24 + D34)/(S2all + S3all + B24 + D24 +
    D34);
  readout_S23IP_tS2_raw := S2all;
  readout_S4IP_tS2 := (B24 + D24)/S4pull;
  readout_S4IP_tS3 := D34/S4pull;
  readout_S4IP_tS4_perMol := (S4 + B24 + D24 + D34)/S4pull;
  readout_S4IP_tS4_raw := S4 + B24 + D24 + D34;
  readout_TGFbR := Rtot - RL;
  readout_Bmp4 := G_Bmp4;
  readout_Cxcl15 := G_Cxcl15;
  readout_Dnmt3a := G_Dnmt3a;
  readout_Dusp5 := G_Dusp5;
  readout_Jun := G_Jun;
```

```
readout_Klf10 := G_Klf10;
readout_Pdk4 := G_Pdk4;
readout_Ski := G_Ski;
readout_Skil := G_Skil;
readout_Smad7 := G_Smad7;
readout_Sox4 := G_Sox4;
readout_Tgfa := G_Tgfa;

// Reactions:
rec_act:  -> RA; (ka*init_TGFb + kbas)*(Rtot - RA - RL);
rec_inact: RA -> ; ki*RA;
rec_int: RA -> RL; kint*RA;
S2_p1: S2u -> S2m; kp21*RA*S2u;
S2_p2: S2m -> S2d; kp22*RA*S2m;
S2_dp2: S2d -> S2m; kd21*S2d;
S2_dp1: S2m -> S2u; kd20*S2m;
S3_p1: S3u -> S3m; kp31*RA*S3u;
S3_p2: S3m -> S3d; kp32*RA*S3m;
S3_dp2: S3d -> S3m; kd31*S3d;
S3_dp1: S3m -> S3u; kd30*S3m;
C23_form: S2d + S3d -> Cd23; kon23d*S2d*S3d;
C23_diss: Cd23 -> S2d + S3d; koff23d*Cd23;
C24_form: S2d + S4 -> D24; kon24*S2d*S4;
C24_diss: D24 -> S2d + S4; koff24*D24;
C34_form: S3d + S4 -> D34; kon34*S3d*S4;
C34_diss: D34 -> S3d + S4; koff34*D34;
gene_Klf10:  -> G_Klf10; delta_Klf10*(1 + aM_Klf10*M/(KM_Klf10 + M) +
   w_Klf10*A2/(K_Klf10 + A2) + b1_Klf10*time/600 - G_Klf10);
gene_Smad7:  -> G_Smad7; delta_Smad7*(1 + aM_Smad7*M/(KM_Smad7 + M) +
   w_Smad7*A2/(K_Smad7 + A2) + b1_Smad7*time/600 - G_Smad7);
gene_Skil:  -> G_Skil; delta_Skil*(1 + a_Skil*Cact/(K_Skil + Cact) +
  w_Skil*A/(K_Skil + A) + b1_Skil*time/600 - G_Skil);
gene_Jun:  -> G_Jun; delta_Jun*(1 + a_Jun*Cact/(K_Jun + Cact) + w_Jun
  *A/(K_Jun + A) + b1_Jun*time/600 - G_Jun);
gene_Ski:  -> G_Ski; delta_Ski*(1 + aM_Ski*M/(KM_Ski + M) + w_Ski*A2
  /(K_Ski + A2) + b1_Ski*time/600 - G_Ski);
gene_Sox4:  -> G_Sox4; delta_Sox4*(1 + w_Sox4*A/(K_Sox4 + A) +
  b1_Sox4*time/600 - G_Sox4);
gene_Dnmt3a:  -> G_Dnmt3a; delta_Dnmt3a*(1 + w_Dnmt3a*A2/(K_Dnmt3a +
  A2) + b1_Dnmt3a*time/600 - G_Dnmt3a);
gene_Bmp4:  -> G_Bmp4; delta_Bmp4*(1 + a_Bmp4*Cact/(K_Bmp4 + Cact) +
  w_Bmp4*A/(K_Bmp4 + A) + b1_Bmp4*time/600 + b2_Bmp4*(time/600)^2 -
  G_Bmp4);
gene_Tgfa:  -> G_Tgfa; delta_Tgfa*((1 - r_Tgfa*A/(K_Tgfa + A)) +
  b1_Tgfa*time/600 - G_Tgfa);
gene_Dusp5:  -> G_Dusp5; delta_Dusp5*((1 - r_Dusp5*Cact/(K_Dusp5 +
  Cact)) + b1_Dusp5*time/600 - G_Dusp5);
gene_Pdk4:  -> G_Pdk4; delta_Pdk4*((1 - r_Pdk4*Cact/(K_Pdk4 + Cact))
  + b1_Pdk4*time/600 - G_Pdk4);
gene_Cxcl15:  -> G_Cxcl15; delta_Cxcl15*((1 - r_Cxcl15*A/(K_Cxcl15 +
  A)) + b1_Cxcl15*time/600 - G_Cxcl15);
A_prod:  -> A; lam*Cact;
A_deg: A -> ; lam*A;
rec_recyc: RL -> RA; krec*RL;
A2_prod:  -> A2; lam2*Cact;
A2_deg: A2 -> ; lam2*A2;
M_prod:  -> M; RA;
M_deg: M -> ; koffM*M;

// Species initializations:
RA = 0;
RL = 0;
S2u = init_S2;
S2m = 0;
S2d = 0;
```

```
S3u = init_S3;
S3m = 0;
S3d = 0;
S4 = init_S4;
Cd23 = 0;
D24 = 0;
D34 = 0;
G_Klf10 = G_Klf10_init;
M = ;
A2 = ;
G_Smad7 = G_Smad7_init;
G_Skil = G_Skil_init;
A = ;
G_Jun = G_Jun_init;
G_Ski = G_Ski_init;
G_Sox4 = G_Sox4_init;
G_Dnmt3a = G_Dnmt3a_init;
G_Bmp4 = G_Bmp4_init;
G_Tgfa = G_Tgfa_init;
G_Dusp5 = G_Dusp5_init;
G_Pdk4 = G_Pdk4_init;
G_Cxcl15 = G_Cxcl15_init;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_S2 = 142.34;
init_S3 = 14.234;
init_S4 = 69.47;
G_Klf10_init = 1;
G_Smad7_init = 1;
G_Skil_init = 1;
G_Jun_init = 1;
G_Ski_init = 1;
G_Sox4_init = 1;
G_Dnmt3a_init = 1;
G_Bmp4_init = 1;
G_Tgfa_init = 1;
G_Dusp5_init = 1;
G_Pdk4_init = 1;
G_Cxcl15_init = 1;
init_TGFb = 1;
S2tot = 142.78;
S3tot = 16.26;
S4tot = 67.02;
Rtot = 1.85;
Kb24 = ;
Kh23u = ;
ka = ;
kbas = ;
ki = ;
kint = ;
kp21 = ;
kp22 = ;
kd21 = ;
kd20 = ;
kp31 = ;
kp32 = ;
kd31 = ;
kd30 = ;
kon23d = ;
koff23d = ;
kon24 = ;
koff24 = ;
```

```
kon34 = ;
koff34 = ;
delta_Klf10 = ;
aM_Klf10 = ;
KM_Klf10 = ;
w_Klf10 = ;
K_Klf10 = ;
b1_Klf10 = ;
delta_Smad7 = ;
aM_Smad7 = ;
KM_Smad7 = ;
w_Smad7 = ;
K_Smad7 = ;
b1_Smad7 = ;
delta_Skil = ;
a_Skil = ;
K_Skil = ;
w_Skil = ;
b1_Skil = ;
delta_Jun = ;
a_Jun = ;
K_Jun = ;
w_Jun = ;
b1_Jun = ;
delta_Ski = ;
aM_Ski = ;
KM_Ski = ;
w_Ski = ;
K_Ski = ;
b1_Ski = ;
delta_Sox4 = ;
w_Sox4 = ;
K_Sox4 = ;
b1_Sox4 = ;
delta_Dnmt3a = ;
w_Dnmt3a = ;
K_Dnmt3a = ;
b1_Dnmt3a = ;
delta_Bmp4 = ;
a_Bmp4 = ;
K_Bmp4 = ;
w_Bmp4 = ;
b1_Bmp4 = ;
b2_Bmp4 = ;
delta_Tgfa = ;
r_Tgfa = ;
K_Tgfa = ;
b1_Tgfa = ;
delta_Dusp5 = ;
r_Dusp5 = ;
K_Dusp5 = ;
b1_Dusp5 = ;
delta_Pdk4 = ;
r_Pdk4 = ;
K_Pdk4 = ;
b1_Pdk4 = ;
delta_Cxcl15 = ;
r_Cxcl15 = ;
K_Cxcl15 = ;
b1_Cxcl15 = ;
lam = ;
krec = ;
lam2 = ;
koffM = ;
```

```
  // Other declarations:
  var B24, H23u, Cact, S2all, S3all, S4pull, readout_S2IP_S2,
    readout_S2IP_pS2;
  var readout_S2IP_ppS2, readout_S2IP_tS2, readout_S2IP_tS3,
    readout_S2IP_tS4;
  var readout_S2IP_S3, readout_S2IP_pS3, readout_S2IP_ppS3,
    readout_S3IP_S3;
  var readout_S3IP_pS3, readout_S3IP_ppS3, readout_S3IP_S2,
    readout_S3IP_pS2;
  var readout_S3IP_ppS2, readout_S3IP_tS2, readout_S3IP_tS3,
    readout_S3IP_tS4;
  var readout_S23IP_S2, readout_S23IP_pS2, readout_S23IP_ppS2,
    readout_S23IP_S3;
  var readout_S23IP_pS3, readout_S23IP_ppS3, readout_S23IP_tS2_perMol,
    readout_S23IP_tS3_perMol;
  var readout_S23IP_tS4, readout_S23IP_tS2_raw, readout_S4IP_tS2,
    readout_S4IP_tS3;
  var readout_S4IP_tS4_perMol, readout_S4IP_tS4_raw, readout_TGFbR,
    readout_Bmp4;
  var readout_Cxcl15, readout_Dnmt3a, readout_Dusp5, readout_Jun,
    readout_Klf10;
  var readout_Pdk4, readout_Ski, readout_Skil, readout_Smad7,
    readout_Sox4;
  var readout_Tgfa;
  const cell, init_S2, init_S3, init_S4, G_Klf10_init, G_Smad7_init,
    G_Skil_init;
  const G_Jun_init, G_Ski_init, G_Sox4_init, G_Dnmt3a_init, G_Bmp4_init
    , G_Tgfa_init;
  const G_Dusp5_init, G_Pdk4_init, G_Cxcl15_init, init_TGFb, S2tot,
    S3tot;
  const S4tot, Rtot, Kb24, Kh23u, ka, kbas, ki, kint, kp21, kp22, kd21,
     kd20;
  const kp31, kp32, kd31, kd30, kon23d, koff23d, kon24, koff24, kon34,
    koff34;
  const delta_Klf10, aM_Klf10, KM_Klf10, w_Klf10, K_Klf10, b1_Klf10,
    delta_Smad7;
  const aM_Smad7, KM_Smad7, w_Smad7, K_Smad7, b1_Smad7, delta_Skil,
    a_Skil;
  const K_Skil, w_Skil, b1_Skil, delta_Jun, a_Jun, K_Jun, w_Jun, b1_Jun
    , delta_Ski;
  const aM_Ski, KM_Ski, w_Ski, K_Ski, b1_Ski, delta_Sox4, w_Sox4,
    K_Sox4;
  const b1_Sox4, delta_Dnmt3a, w_Dnmt3a, K_Dnmt3a, b1_Dnmt3a,
    delta_Bmp4;
  const a_Bmp4, K_Bmp4, w_Bmp4, b1_Bmp4, b2_Bmp4, delta_Tgfa, r_Tgfa,
    K_Tgfa;
  const b1_Tgfa, delta_Dusp5, r_Dusp5, K_Dusp5, b1_Dusp5, delta_Pdk4,
    r_Pdk4;
  const K_Pdk4, b1_Pdk4, delta_Cxcl15, r_Cxcl15, K_Cxcl15, b1_Cxcl15,
    lam;
  const krec, lam2, koffM;
end
```

Listing 112: Archived Antimony for `lucarelli_4`; task `56836792_13`; selected version 25

```
// Created by libAntimony v3.1.3
model *hepa16_tgfb_smad()

  // Compartments and Species:
  compartment cell;
  species init_S2 in cell, init_S3 in cell, init_S4 in cell, init_TGFb
    in cell;
  species $Ract in cell, S2U in cell, S2M in cell, S2D in cell, S3U in
    cell;
```

```
  species S3M in cell, S3D in cell, S4 in cell, C2u in cell, C3u in
    cell;
  species C2d in cell, C3d in cell, C22 in cell, C33 in cell, C23_DD in
     cell;
  species $G_Bmp4 in cell, G_Cxcl15 in cell, G_Dnmt3a in cell, G_Dusp5
    in cell;
  species G_Jun in cell, $G_Klf10 in cell, G_Pdk4 in cell, G_Ski in
    cell;
  species G_Skil in cell, $G_Smad7 in cell, G_Sox4 in cell, G_Tgfa in
    cell;
  species C2s, C3s;

  // Assignment Rules:
  Ract := (1 - exp(-k_act*init_TGFb*time))*exp(-k_in*time);
  G_Bmp4 := b_Bmp4 + a_Bmp4*SatE + aL_Bmp4*SatL;
  SatE := (C2d + C3d + C22 + C33 + C23_DD)/(KD_g + C2d + C3d + C22 +
    C33 + C23_DD);
  SatL := (C2s + C3s)/(KD_g + C2s + C3s);
  G_Klf10 := b_Klf10 + a_Klf10*SatE + aL_Klf10*SatL;
  G_Smad7 := b_Smad7 + a_Smad7*SatE + aL_Smad7*SatL;
  T2 := S2U + S2M + S2D + C2u + C2d + 2*C22 + C23_DD + C2s;
  T3 := S3U + S3M + S3D + C3u + C3d + 2*C33 + C23_DD + C3s;
  U2p := S2U + C2u + C2s;
  D2p := S2D + C2d + 2*C22 + C23_DD;
  U3p := S3U + C3u + C3s;
  D3p := S3D + C3d + 2*C33 + C23_DD;
  S4bp := C2u + C3u + C2d + C3d + C22 + C33 + C23_DD + C2s + C3s;
  S4tp := S4 + S4bp;
  S2in4 := C2u + C2d + 2*C22 + C23_DD + C2s;
  S3in4 := C3u + C3d + 2*C33 + C23_DD + C3s;
  S4in2 := C2u + C2d + C22 + C23_DD + C2s;
  S4in3 := C3u + C3d + C33 + C23_DD + C3s;
  readout_S23IP_S2 := f5*U2p/(D2p + f5*(U2p + S2M));
  readout_S23IP_pS2 := f5*S2M/(D2p + f5*(U2p + S2M));
  readout_S23IP_ppS2 := D2p/(D2p + f5*(U2p + S2M));
  readout_S23IP_S3 := f5*U3p/(D3p + f5*(U3p + S3M));
  readout_S23IP_pS3 := f5*S3M/(D3p + f5*(U3p + S3M));
  readout_S23IP_ppS3 := D3p/(D3p + f5*(U3p + S3M));
  readout_S23IP_tS2_perMol := T2/(T2 + T3);
  readout_S23IP_tS3_perMol := T3/(T2 + T3);
  readout_S23IP_tS4 := S4bp/(T2 + T3 + S4bp);
  readout_S23IP_tS2_raw := T2;
  readout_S2IP_S2 := U2p/T2;
  readout_S2IP_pS2 := S2M/T2;
  readout_S2IP_ppS2 := D2p/T2;
  readout_S2IP_tS2 := T2/(T2 + C23_DD + S4in2);
  readout_S2IP_tS3 := C23_DD/(T2 + C23_DD + S4in2);
  readout_S2IP_tS4 := S4in2/(T2 + C23_DD + S4in2);
  readout_S2IP_S3 := bg_f/(C23_DD + bg_f);
  readout_S2IP_pS3 := bg_f/(C23_DD + bg_f);
  readout_S2IP_ppS3 := C23_DD/(C23_DD + bg_f);
  readout_S3IP_S3 := U3p/T3;
  readout_S3IP_pS3 := S3M/T3;
  readout_S3IP_ppS3 := D3p/T3;
  readout_S3IP_tS3 := T3/S3Pel;
  S3Pel := T3 + S4in3 + C23_DD + q_x3*(C2d + C22 + C2s) + w32f*S2U;
  readout_S3IP_tS2 := (C23_DD + q_x3*(C2d + C22 + C2s) + w32f*S2U)/
    S3Pel;
  readout_S3IP_tS4 := S4in3/S3Pel;
  readout_S3IP_S2 := bg_f/(C23_DD + bg_f);
  readout_S3IP_pS2 := bg_f/(C23_DD + bg_f);
  readout_S3IP_ppS2 := C23_DD/(C23_DD + bg_f);
  readout_S4IP_tS2 := S2in4/(S4tp + S2in4 + S3in4);
  readout_S4IP_tS3 := S3in4/(S4tp + S2in4 + S3in4);
  readout_S4IP_tS4_perMol := S4tp/(S4tp + S2in4 + S3in4);
```

```
readout_S4IP_tS4_raw := S4tp;
readout_TGFbR := Rtot;
readout_Bmp4 := G_Bmp4;
readout_Cxcl15 := G_Cxcl15;
readout_Dnmt3a := G_Dnmt3a;
readout_Dusp5 := G_Dusp5;
readout_Jun := G_Jun;
readout_Klf10 := G_Klf10;
readout_Pdk4 := G_Pdk4;
readout_Ski := G_Ski;
readout_Skil := G_Skil;
readout_Smad7 := G_Smad7;
readout_Sox4 := G_Sox4;
readout_Tgfa := G_Tgfa;

// Reactions:
s2_p1: S2U -> S2M; kp1_2*Ract*S2U;
s2_p2: S2M -> S2D; kp2_2*Ract*S2M;
s2_dpM: S2M -> S2U; kdp_2*S2M;
s2_dpD: S2D -> S2M; kdp_2*S2D;
s3_p1: S3U -> S3M; kp1_3*Ract*S3U;
s3_p2: S3M -> S3D; kp2_3*Ract*S3M;
s3_dpM: S3M -> S3U; kdp_3*S3M;
s3_dpD: S3D -> S3M; kdp_D3*S3D;
d2u: S2U + S4 -> C2u; kaD*S2U*S4 - kdD*C2u;
d2d: S2D + S4 -> C2d; kaD*S2D*S4 - kdD*C2d;
d3u: S3U + S4 -> C3u; kaD*S3U*S4 - kdD*C3u;
d3d: S3D + S4 -> C3d; kaD*S3D*S4 - kdD*C3d;
t22: C2d + S2D -> C22; kaT*C2d*S2D - kdT*C22;
t33: C3d + S3D -> C33; kaT*C3d*S3D - kdT*C33;
t23: C2d + S3D -> C23_DD; kaT*C2d*S3D - kdT*C23_DD;
g_Cxcl15:  -> G_Cxcl15; ddeg_Cxcl15*(b_Cxcl15/(1 + a_Cxcl15*SatE +
  aL_Cxcl15*SatL) - G_Cxcl15);
g_Dnmt3a:  -> G_Dnmt3a; ddeg_Dnmt3a*(b_Dnmt3a + a_Dnmt3a*SatE -
  G_Dnmt3a);
g_Dusp5:  -> G_Dusp5; ddeg_Dusp5*(b_Dusp5/(1 + a_Dusp5*SatE +
  aL_Dusp5*SatL) - G_Dusp5);
g_Jun:  -> G_Jun; ddeg_Jun*(b_Jun + a_Jun*SatE + aL_Jun*SatL - G_Jun)
  ;
g_Pdk4:  -> G_Pdk4; ddeg_Pdk4*(b_Pdk4/(1 + a_Pdk4*SatE + aL_Pdk4*SatL
  ) - G_Pdk4);
g_Ski:  -> G_Ski; ddeg_Ski*(b_Ski + a_Ski*SatE - G_Ski);
g_Skil:  -> G_Skil; ddeg_Skil*(b_Skil + a_Skil*SatE + aL_Skil*SatL -
  G_Skil);
g_Sox4:  -> G_Sox4; ddeg_Sox4*(b_Sox4 + a_Sox4*SatE + aL_Sox4*SatL -
  G_Sox4);
g_Tgfa:  -> G_Tgfa; ddeg_Tgfa*(b_Tgfa/(1 + a_Tgfa*SatE) - G_Tgfa);
s2_stab: S2U + S4 -> C2s; ka_s*Ract*S2U*S4;
s2_rel: C2s -> S2U + S4; kd_s*C2s;
s3_stab: S3U + S4 -> C3s; ka_s*Ract*S3U*S4;
s3_rel: C3s -> S3U + S4; kd_s*C3s;

// Species initializations:
init_S2 = 142.34;
init_S3 = 14.234;
init_S4 = 69.47;
init_TGFb = 0;
S2U = init_S2;
S2M = 0;
S2D = 0;
S3U = init_S3;
S3M = 0;
S3D = 0;
S4 = init_S4;
C2u = 0;
```

```
C3u = 0;
C2d = 0;
C3d = 0;
C22 = 0;
C33 = 0;
C23_DD = 0;
G_Cxcl15 = G_Cxcl15_init;
G_Dnmt3a = G_Dnmt3a_init;
G_Dusp5 = G_Dusp5_init;
G_Jun = G_Jun_init;
G_Pdk4 = G_Pdk4_init;
G_Ski = G_Ski_init;
G_Skil = G_Skil_init;
G_Sox4 = G_Sox4_init;
G_Tgfa = G_Tgfa_init;
C2s = ;
C3s = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
k_act = 0.1;
k_in = 0.012;
b_Bmp4 = 1;
a_Bmp4 = 0.5;
aL_Bmp4 = ;
G_Cxcl15_init = 1;
G_Dnmt3a_init = 1;
G_Dusp5_init = 1;
G_Jun_init = 1;
b_Klf10 = 1;
a_Klf10 = 0.7;
aL_Klf10 = ;
G_Pdk4_init = 1;
G_Ski_init = 1;
G_Skil_init = 1;
b_Smad7 = 1;
a_Smad7 = 0.5;
aL_Smad7 = ;
G_Sox4_init = 1;
G_Tgfa_init = 1;
S2tot = 142.34;
S3tot = 16.26;
S4tot = 69.47;
Rtot = 1.85;
kp1_2 = 0.02;
kp2_2 = 0.05;
kdp_2 = 0.01;
kp1_3 = 0.02;
kp2_3 = 0.03;
kdp_3 = 0.0035;
kaD = 2e-05;
kdD = 0.02;
kaT = 3e-05;
kdT = 0.003;
b_Cxcl15 = 1;
a_Cxcl15 = 0.5;
ddeg_Cxcl15 = 0.003;
b_Dnmt3a = 1;
a_Dnmt3a = 0.3;
ddeg_Dnmt3a = 0.01;
b_Dusp5 = 1;
a_Dusp5 = 0.5;
ddeg_Dusp5 = 0.01;
```

```
b_Jun = 1;
a_Jun = 0.5;
ddeg_Jun = 0.005;
b_Pdk4 = 1;
a_Pdk4 = 0.5;
ddeg_Pdk4 = 0.01;
b_Ski = 1;
a_Ski = 0.3;
ddeg_Ski = 0.005;
b_Skil = 1;
a_Skil = 0.6;
ddeg_Skil = 0.004;
b_Sox4 = 1;
a_Sox4 = 0.3;
ddeg_Sox4 = 0.005;
b_Tgfa = 1;
a_Tgfa = 0.5;
ddeg_Tgfa = 0.001;
KD_g = ;
f5 = ;
bg_f = ;
q_x3 = ;
w32f = ;
kdp_D3 = ;
aL_Cxcl15 = ;
aL_Dusp5 = ;
aL_Jun = ;
aL_Pdk4 = ;
aL_Skil = ;
aL_Sox4 = ;
ka_s = ;
kd_s = ;

// Other declarations:
var SatE, SatL, T2, T3, U2p, D2p, U3p, D3p, S4bp, S4tp, S2in4, S3in4,
   S4in2;
var S4in3, readout_S23IP_S2, readout_S23IP_pS2, readout_S23IP_ppS2,
  readout_S23IP_S3;
var readout_S23IP_pS3, readout_S23IP_ppS3, readout_S23IP_tS2_perMol,
  readout_S23IP_tS3_perMol;
var readout_S23IP_tS4, readout_S23IP_tS2_raw, readout_S2IP_S2,
  readout_S2IP_pS2;
var readout_S2IP_ppS2, readout_S2IP_tS2, readout_S2IP_tS3,
  readout_S2IP_tS4;
var readout_S2IP_S3, readout_S2IP_pS3, readout_S2IP_ppS3,
  readout_S3IP_S3;
var readout_S3IP_pS3, readout_S3IP_ppS3, readout_S3IP_tS3, S3Pel,
  readout_S3IP_tS2;
var readout_S3IP_tS4, readout_S3IP_S2, readout_S3IP_pS2,
  readout_S3IP_ppS2;
var readout_S4IP_tS2, readout_S4IP_tS3, readout_S4IP_tS4_perMol,
  readout_S4IP_tS4_raw;
var readout_TGFbR, readout_Bmp4, readout_Cxcl15, readout_Dnmt3a,
  readout_Dusp5;
var readout_Jun, readout_Klf10, readout_Pdk4, readout_Ski,
  readout_Skil;
var readout_Smad7, readout_Sox4, readout_Tgfa;
const cell, k_act, k_in, b_Bmp4, a_Bmp4, aL_Bmp4, G_Cxcl15_init,
  G_Dnmt3a_init;
const G_Dusp5_init, G_Jun_init, b_Klf10, a_Klf10, aL_Klf10,
  G_Pdk4_init;
const G_Ski_init, G_Skil_init, b_Smad7, a_Smad7, aL_Smad7,
  G_Sox4_init;
const G_Tgfa_init, S2tot, S3tot, S4tot, Rtot, kp1_2, kp2_2, kdp_2,
  kp1_3;
```

```
  const kp2_3, kdp_3, kaD, kdD, kaT, kdT, b_Cxcl15, a_Cxcl15,
    ddeg_Cxcl15;
  const b_Dnmt3a, a_Dnmt3a, ddeg_Dnmt3a, b_Dusp5, a_Dusp5, ddeg_Dusp5,
    b_Jun;
  const a_Jun, ddeg_Jun, b_Pdk4, a_Pdk4, ddeg_Pdk4, b_Ski, a_Ski,
    ddeg_Ski;
  const b_Skil, a_Skil, ddeg_Skil, b_Sox4, a_Sox4, ddeg_Sox4, b_Tgfa,
    a_Tgfa;
  const ddeg_Tgfa, KD_g, f5, bg_f, q_x3, w32f, kdp_D3, aL_Cxcl15,
    aL_Dusp5;
  const aL_Jun, aL_Pdk4, aL_Skil, aL_Sox4, ka_s, kd_s;
end
```

Listing 113: Archived Antimony for `lucarelli_5`; task `56836792_14`; selected version 24

```
// Created by libAntimony v3.1.3
model *tgfbsmad_hepa16()

  // Compartments and Species:
  compartment cell;
  species S2 in cell, pS2 in cell, ppS2 in cell, S3 in cell, pS3 in
    cell;
  species ppS3 in cell, S4 in cell, Cb in cell, T22 in cell, D23 in
    cell;
  species Ra in cell, M_Bmp4 in cell, M_Cxcl15 in cell, M_Dnmt3a in
    cell;
  species M_Dusp5 in cell, M_Jun in cell, M_Klf10 in cell, M_Pdk4 in
    cell;
  species M_Ski in cell, M_Skil in cell, M_Smad7 in cell, M_Sox4 in
    cell;
  species M_Tgfa in cell, T33;

  // Assignment Rules:
  A_p3 := ppS3/(K_b + ppS3);
  A_S4 := (T22 + T33)/(K_A + T22 + T33);
  A_k := ppS3*ppS3*S4/(K_f + ppS3*ppS3*S4);
  A2 := ppS2/(K2 + ppS2);
  A3 := T33/(K3 + T33);
  AD := D23/(KD + D23);
  A_f := k1_s3*Ra*S3/(K_F + k1_s3*Ra*S3);
  readout_Bmp4 := M_Bmp4;
  readout_Cxcl15 := M_Cxcl15;
  readout_Dnmt3a := M_Dnmt3a;
  readout_Dusp5 := M_Dusp5;
  readout_Jun := M_Jun;
  readout_Klf10 := M_Klf10;
  readout_Pdk4 := M_Pdk4;
  readout_Ski := M_Ski;
  readout_Skil := M_Skil;
  readout_Smad7 := M_Smad7;
  readout_Sox4 := M_Sox4;
  readout_Tgfa := M_Tgfa;
  readout_S23IP_S2 := (S2 + Cb)/S2tot;
  readout_S23IP_pS2 := pS2/S2tot;
  readout_S23IP_ppS2 := (ppS2 + 2*T22 + D23)/S2tot;
  readout_S23IP_S3 := S3/S3tot;
  readout_S23IP_pS3 := pS3/S3tot;
  readout_S23IP_ppS3 := (ppS3 + D23 + 2*T33)/S3tot;
  readout_S23IP_tS2_perMol := S2tot/(S2tot + S3tot);
  readout_S23IP_tS3_perMol := S3tot/(S2tot + S3tot);
  readout_S23IP_tS2_raw := S2tot;
  readout_S23IP_tS4 := (Cb + T22 + T33)/(S2tot + S3tot + Cb + T22 + T33
    );
  den3_2ip := rec_2ip*S3tot + D23;
  readout_S2IP_S2 := (S2 + Cb)/S2tot;
```

```
readout_S2IP_pS2 := pS2/S2tot;
readout_S2IP_ppS2 := (ppS2 + 2*T22 + D23)/S2tot;
readout_S2IP_S3 := 100*rec_2ip*S3/den3_2ip;
readout_S2IP_pS3 := 100*rec_2ip*pS3/den3_2ip;
readout_S2IP_ppS3 := (rec_2ip*ppS3 + D23)/den3_2ip;
readout_S2IP_tS2 := S2tot/(S2tot + den3_2ip + Cb + T22);
readout_S2IP_tS3 := den3_2ip/(S2tot + den3_2ip + Cb + T22);
readout_S2IP_tS4 := (Cb + T22)/(S2tot + den3_2ip + Cb + T22);
den2_3ip := rec_3ip*S2tot + D23;
readout_S3IP_S3 := S3/S3tot;
readout_S3IP_pS3 := pS3/S3tot;
readout_S3IP_ppS3 := (ppS3 + D23 + 2*T33)/S3tot;
readout_S3IP_S2 := 100*rec_3ip*S2/den2_3ip;
readout_S3IP_pS2 := 100*rec_3ip*pS2/den2_3ip;
readout_S3IP_ppS2 := (rec_3ip*ppS2 + D23)/den2_3ip;
readout_S3IP_tS2 := den2_3ip/(S3tot + den2_3ip + T33);
readout_S3IP_tS3 := S3tot/(S3tot + den2_3ip + T33);
readout_S3IP_tS4 := T33/(S3tot + den2_3ip + T33);
readout_S4IP_tS2 := (Cb + 2*T22)/(S4tot + Cb + 2*T22 + 2*T33);
readout_S4IP_tS3 := 2*T33/(S4tot + Cb + 2*T22 + 2*T33);
readout_S4IP_tS4_perMol := S4tot/(S4tot + Cb + 2*T22 + 2*T33);
readout_S4IP_tS4_raw := S4 + Cb + T22 + T33;
readout_TGFbR := Rtot;

// Reactions:
ra_act:  -> Ra; ka*(init_TGFb + Ra0)*(Rtot - Ra);
ra_deact: Ra -> ; ki*Ra;
ra_feedback: Ra -> ; k7*M_Smad7*Ra;
s2_p: S2 -> pS2; k1_s2*Ra*S2;
s2_pp: pS2 -> ppS2; k2_s2*Ra*pS2;
s2_dp: ppS2 -> S2; dp_s2*ppS2;
s3_p: S3 -> pS3; k1_s3*Ra*S3;
s3_pp: pS3 -> ppS3; k2_s3*Ra*pS3;
s3_dp: ppS3 -> S3; dp_s3*ppS3;
cb_form: S2 + S4 -> Cb; kon_b*S2*S4;
cb_off: Cb -> S2 + S4; koff_b*Cb;
t22_form: 2 ppS2 + S4 -> T22; kco*ppS2*ppS2*S4;
t22_off: T22 -> 2 ppS2 + S4; koff_t*T22;
d23_form: ppS2 + ppS3 -> D23; kon_d*ppS2*ppS3;
d23_off: D23 -> ppS2 + ppS3; koff_d*D23;
turn_Bmp4: M_Bmp4 -> ; kdeg_Bmp4*(M_Bmp4 - 1 - amp_Bmp4*A_p3);
turn_Cxcl15: M_Cxcl15 -> ; kdeg_Cxcl15*((1 + rep_Cxcl15*A_S4)*
  M_Cxcl15 - 1 - ampC_Cxcl15*A_k);
turn_Dnmt3a: M_Dnmt3a -> ; kdeg_Dnmt3a*(M_Dnmt3a - 1 - amp_Dnmt3a*A2)
  ;
turn_Dusp5: M_Dusp5 -> ; kdeg_Dusp5*((1 + rep_Dusp5*A_S4)*M_Dusp5 - 1
   - amp_Dusp5*A_S4);
turn_Jun: M_Jun -> ; kdeg_Jun*(M_Jun - 1 - amp_Jun*A3);
turn_Klf10: M_Klf10 -> ; kdeg_Klf10*(M_Klf10 - 1 - amp_Klf10*A_k -
  amp2_Klf10*A2);
turn_Pdk4: M_Pdk4 -> ; kdeg_Pdk4*((1 + rep_Pdk4*A_S4)*M_Pdk4 - 1 -
  amp_Pdk4*A_S4);
turn_Ski: M_Ski -> ; kdeg_Ski*(M_Ski - 1 - amp_Ski*AD);
turn_Skil: M_Skil -> ; kdeg_Skil*(M_Skil - 1 - amp_Skil*AD);
turn_Smad7: M_Smad7 -> ; kdeg_Smad7*(M_Smad7 - 1 - amp_Smad7*A_f -
  amp2_Smad7*A2);
turn_Sox4: M_Sox4 -> ; kdeg_Sox4*(M_Sox4 - 1 - amp_Sox4*A2);
turn_Tgfa: M_Tgfa -> ; kdeg_Tgfa*((1 + rep_Tgfa*A_S4)*M_Tgfa - 1 -
  amp_Tgfa*A_S4);
t33_form: 2 ppS3 + S4 -> T33; kco3*ppS3*ppS3*S4;
t33_off: T33 -> 2 ppS3 + S4; koff_t3*T33;

// Species initializations:
S2 = init_S2;
pS2 = 0;
```

```
ppS2 = 0;
S3 = init_S3;
pS3 = 0;
ppS3 = 0;
S4 = init_S4;
Cb = Cb_init;
T22 = 0;
D23 = 0;
Ra = 0;
M_Bmp4 = M_Bmp4_init;
M_Cxcl15 = M_Cxcl15_init;
M_Dnmt3a = M_Dnmt3a_init;
M_Dusp5 = M_Dusp5_init;
M_Jun = M_Jun_init;
M_Klf10 = M_Klf10_init;
M_Pdk4 = M_Pdk4_init;
M_Ski = M_Ski_init;
M_Skil = M_Skil_init;
M_Smad7 = M_Smad7_init;
M_Sox4 = M_Sox4_init;
M_Tgfa = M_Tgfa_init;
T33 = ;

// Compartment initializations:
cell = 1;

// Variable initializations:
init_S2 = 142.8;
init_S3 = 16.3;
init_S4 = 69.5;
Cb_init = 9;
M_Bmp4_init = 1;
M_Cxcl15_init = 1;
M_Dnmt3a_init = 1;
M_Dusp5_init = 1;
M_Jun_init = 1;
M_Klf10_init = 1;
M_Pdk4_init = 1;
M_Ski_init = 1;
M_Skil_init = 1;
M_Smad7_init = 1;
M_Sox4_init = 1;
M_Tgfa_init = 1;
init_TGFb = 0;
S2tot = 142.8;
S3tot = 16.3;
S4tot = 69.5;
Rtot = 1.85;
ka = 0.02;
ki = 0.01;
k7 = 0.003;
k1_s2 = 0.03;
k2_s2 = 0.4;
dp_s2 = 0.008;
k1_s3 = 0.03;
k2_s3 = 0.15;
dp_s3 = 0.004;
kon_b = 0.0012;
koff_b = 1;
kco = 1.7e-07;
koff_t = 0.01;
kon_d = 0.0023;
koff_d = 1;
rec_2ip = 0.004;
rec_3ip = 0.0006;
```

```
  K_A = 3;
  K_b = ;
  K_f = ;
  K2 = ;
  K3 = ;
  KD = ;
  K_F = ;
  Ra0 = ;
  kdeg_Bmp4 = ;
  amp_Bmp4 = ;
  kdeg_Cxcl15 = ;
  rep_Cxcl15 = ;
  ampC_Cxcl15 = ;
  kdeg_Dnmt3a = ;
  amp_Dnmt3a = ;
  kdeg_Dusp5 = ;
  rep_Dusp5 = ;
  amp_Dusp5 = ;
  kdeg_Jun = ;
  amp_Jun = ;
  kdeg_Klf10 = ;
  amp_Klf10 = ;
  amp2_Klf10 = ;
  kdeg_Pdk4 = ;
  rep_Pdk4 = ;
  amp_Pdk4 = ;
  kdeg_Ski = ;
  amp_Ski = ;
  kdeg_Skil = ;
  amp_Skil = ;
  kdeg_Smad7 = ;
  amp_Smad7 = ;
  amp2_Smad7 = ;
  kdeg_Sox4 = ;
  amp_Sox4 = ;
  kdeg_Tgfa = ;
  rep_Tgfa = ;
  amp_Tgfa = ;
  kco3 = ;
  koff_t3 = ;

  // Other declarations:
  var A_p3, A_S4, A_k, A2, A3, AD, A_f, readout_Bmp4, readout_Cxcl15,
    readout_Dnmt3a;
  var readout_Dusp5, readout_Jun, readout_Klf10, readout_Pdk4,
    readout_Ski;
  var readout_Skil, readout_Smad7, readout_Sox4, readout_Tgfa,
    readout_S23IP_S2;
  var readout_S23IP_pS2, readout_S23IP_ppS2, readout_S23IP_S3,
    readout_S23IP_pS3;
  var readout_S23IP_ppS3, readout_S23IP_tS2_perMol,
    readout_S23IP_tS3_perMol;
  var readout_S23IP_tS2_raw, readout_S23IP_tS4, den3_2ip,
    readout_S2IP_S2;
  var readout_S2IP_pS2, readout_S2IP_ppS2, readout_S2IP_S3,
    readout_S2IP_pS3;
  var readout_S2IP_ppS3, readout_S2IP_tS2, readout_S2IP_tS3,
    readout_S2IP_tS4;
  var den2_3ip, readout_S3IP_S3, readout_S3IP_pS3, readout_S3IP_ppS3,
    readout_S3IP_S2;
  var readout_S3IP_pS2, readout_S3IP_ppS2, readout_S3IP_tS2,
    readout_S3IP_tS3;
  var readout_S3IP_tS4, readout_S4IP_tS2, readout_S4IP_tS3,
    readout_S4IP_tS4_perMol;
  var readout_S4IP_tS4_raw, readout_TGFbR;
```

```
  const cell, init_S2, init_S3, init_S4, Cb_init, M_Bmp4_init,
    M_Cxcl15_init;
  const M_Dnmt3a_init, M_Dusp5_init, M_Jun_init, M_Klf10_init,
    M_Pdk4_init;
  const M_Ski_init, M_Skil_init, M_Smad7_init, M_Sox4_init, M_Tgfa_init
    , init_TGFb;
  const S2tot, S3tot, S4tot, Rtot, ka, ki, k7, k1_s2, k2_s2, dp_s2,
    k1_s3;
  const k2_s3, dp_s3, kon_b, koff_b, kco, koff_t, kon_d, koff_d,
    rec_2ip;
  const rec_3ip, K_A, K_b, K_f, K2, K3, KD, K_F, Ra0, kdeg_Bmp4,
    amp_Bmp4;
  const kdeg_Cxcl15, rep_Cxcl15, ampC_Cxcl15, kdeg_Dnmt3a, amp_Dnmt3a,
    kdeg_Dusp5;
  const rep_Dusp5, amp_Dusp5, kdeg_Jun, amp_Jun, kdeg_Klf10, amp_Klf10,
     amp2_Klf10;
  const kdeg_Pdk4, rep_Pdk4, amp_Pdk4, kdeg_Ski, amp_Ski, kdeg_Skil,
    amp_Skil;
  const kdeg_Smad7, amp_Smad7, amp2_Smad7, kdeg_Sox4, amp_Sox4,
    kdeg_Tgfa;
  const rep_Tgfa, amp_Tgfa, kco3, koff_t3;
end
```

I AGENT PROMPTS AND BENCHMARK INPUTS

This appendix reproduces all 66 prompt files supplied with the manuscript: the lead and specialist instructions, control templates, and task, constraint and entity inputs for the 18 benchmark problems. The text is preserved from the source export at revision `722f259d359da12b0cd5fcef61e6de7aba7aaa9c`; provenance and per-prompt hashes are retained in `paper/prompts/PROVENANCE.md` and `paper/prompts/manifest.json`. The hashes describe prompt strings: where export adds a final newline, that newline is excluded from the digest. The manifest's `prompt_version` is unset. This export therefore documents source instructions, rather than establishing the exact prompt text of every historical run; runtime tool schemas, delegated requests and conversations are not reproduced here. For typesetting, derived display copies wrap long lines and render mathematical symbols; the original files and their hashes are unchanged.

Configuration. The panel prompts were exported for the baseline configuration, with structured edits, free exponents and the biocurator enabled. Templates for the evaluated bare-agent (Listings 120–123) and biology-blinded (Listing 124) controls are reproduced separately below.

I.1 MODELLING LEAD

The lead constructs and revises Antimony models through the MCP server, using fit diagnostics (Appendix B.7) and BIC to guide an iterative build–fit–diagnose loop. It delegates independent analyses, reconciles their recommendations, and retains responsibility for model changes.

Listing 114: Lead agent system prompt (baseline)

```
You are the MODELING LEAD building a mechanistic ODE model for a PEtab
 v1 problem through an MCP server. The problem ships WITHOUT a model —
 only the READ-ONLY experimental design (measurements, observables,
 conditions). The problem is exposed ONLY through these structured MCP
 tools; the filesystem is EMPTY by design — do NOT use ls/glob/read to
 hunt for it, there is nothing on disk to find. Your job is to BUILD,
 from scratch, a model that explains the data. You do NOT read the
 measured data or query the fit yourself — you have four independent
 analysts via the `task` tool (pass `subagent_type`): `mathematician`,
 `systems_scientist`, `signal_engineer`, `biocurator`. A subagent's
 ENTIRE input is the `description` string you pass to `task` — put your
 question and any framing THERE; `task` takes no other content argument.
 (The biocurator is the exception: its biological entities are provided
 to it automatically, so you need not name them.) You directly read only
 the model, its SCORES (which `make_changes` itself returns — metrics,
 bic, the noise-sigma verdict and the residual_structure verdict), and a
 COMPACT fit diagnosis (describe_problem, get_model_structure,
 list_observables, get_fit_diagnosis, list_versions) — get_fit_diagnosis
 covers the residual detail and, as
 get_fit_diagnosis(sections='parameters'), the parameter identifiability;
 for the raw DATA and any DEEP diagnosis (a full residual breakdown, the
 optimizer/fitting log) you DELEGATE to the analysts — they read the
 bulky data in their OWN context and hand you back summaries, so your
 context stays lean.

Work as an explicit model-building LOOP:
0. Read both skills once, before your first `make_changes`:
 `get_skill('petab-model-building')` and
 `get_skill('antimony-authoring')`. They are short, and they carry domain
 knowledge this prompt does not.
1. Orient — CAP this at ~3 tool rounds: describe_problem,
 get_model_structure, list_observables, list_conditions. Do not skip
 `list_conditions`: what it reports is an experimental INPUT your model
 has to reproduce by name, not something to fit. Once you have the
 observables and the current structure you MUST call `make_changes`
 BEFORE any further inspection — do not re-propose the same read-only
 batch every step or keep probing; a first fit is the fastest way to
```

```
learn what the model is missing. On the first round, delegate to all
three analysts to characterise the DATA — its dynamics, the plausible
mechanism, and the signal structure — giving each only the objective
facts (the observables and that measured data exist), NOT your own
hypothesis, so their reads stay independent. And for a
molecular/biochemical problem, consult the biocurator on this SAME first
round for the KNOWN biology — the interactions, cooperativity, and
unmeasured regulators/intermediates among the entities — and treat what
it reports as the mechanism's expected TOPOLOGY, to be built unless the
data contradict it. Do this regardless of how many conditions the
problem has: condition count reflects experimental DESIGN, not
mechanistic difficulty, and a single-condition problem can still be a
multi-species, non-monotonic time course that needs the mechanism +
numerics reads as much as any multi-condition one.
1b. RECONCILE the panel before you build: state the mechanism/structure
the analysts recommend, adopt what they AGREE on, and if you deviate say
why. Their guidance is the primary input to your model — do not quietly
ignore it (e.g. if they report asymmetric amplitudes, per-species
parameters; if they name a topology, build that topology). But do not
swallow a single analyst's number uncritically: if a claim contradicts
the others or is physically impossible, sanity-check it (a fitted
parameter is a LINEAR value — check it against its reported bounds/scale
before treating it as pathological).
2. Build the WHOLE model in ONE `make_changes` call: declare the
compartments, species (with their initial conditions), parameters, and
reactions in Antimony. `make_changes` is the ONLY way to change anything
— a rate-law form, a reaction, a parameter's bounds or scale — and it
ALWAYS fits, so one call is one coherent change scored as ONE version.
Give `antimony` to author the FIRST model, `model_edits` (structured
operations) to change it thereafter, `parameters` to retune the search
space, or a model route AND `parameters` when a structural change needs
bounds; at least one is required. Your step budget is the scarce
resource, so put a whole coherent change in one call rather than
dribbling it out. The result IS the fit (`bic`, `neg_log_likelihood`,
`k` = `n_estimated` + `hardcoded_constants`: a numeric literal in your
model's math is a parameter too, bar a few structural ones, and the
result lists the ones you are being charged for. So freezing a value you
fitted costs what estimating it costs and buys none of the flexibility,
an ultra-narrow box does not dodge it either, and a constant that is
genuinely known is one the problem ships FIXED — reference it by name).
When a call does NOT produce a version, READ the result before retrying
— never resubmit the same failing model; the result says whether the
change is GONE (`applied: false` — rejected, or reverted because it
could not be scored, so fix what the error names) or in place but
unscored because this structure already IS a version (make a genuinely
DIFFERENT change). THE ONLY SUCCESS TARGET IS BIC (2·nll + k·ln(n)); a
lower bic is a better model and is EXACTLY what you are scored on
(against the true mechanism). The fit also reports reduced_chi2 and
chi2_gof_pvalue, but do NOT treat them as a success gate: they only mean
something with a KNOWN noise level, and WHENEVER the observable noise
sigmas are ESTIMATED (check get_parameters — a sigma_* with estimate=1;
usually the case here) chi2 pins near the number of data points by
construction, so reduced_chi2 ≈ 1 and its p-value are UNINFORMATIVE —
often stuck at the noise floor however wrong the mechanism is. Do NOT
chase chi2_gof_pvalue ≥ 0.05, and do NOT restructure or trade away
nll/bic to raise it. NEVER free or widen an estimated noise sigma to
lower reduced_chi2. Nothing stops you — widening one past its
data-driven cap only WARNS — which is exactly why it is on you: it hides
a missing mechanism behind inflated noise, and the finish gate refuses
a run resting on one. Judge whether the mechanism is RIGHT from the
residual STRUCTURE — get_fit_diagnosis's per-observable bias + worst-fit
points, or a signal_engineer time-trend read: SYSTEMATIC (non-random)
residuals mean a mechanism is still missing; clean random scatter means
it isn't. INITIAL CONDITIONS — the server AUTO-GENERATES an
`<species>_init` parameter for most species you declare, ESTIMATED by
```

```
 default, whether or not you write one: every species you add silently
 adds a free IC parameter to k. Two initial VALUES are treated specially:
 exactly 0 (an empty pool, which is also what leaving the initial
 unwritten means) gets NO `<species>_init` at all — the state simply
 starts empty — and exactly 1 (a unit normalization) gets one held FIXED.
 Any other declared initial becomes an estimated `<species>_init`. So an
 estimated IC the DATA CANNOT CONSTRAIN — a starting state the
 measurement does not depend on — is a non-identifiable free parameter
 inflating k for no gain in fit (and the finish gate REFUSES a
 rank-deficient fit on exactly these). It is on YOU to decide which
 initials the experiment actually pins. The ONLY lever is the VALUE: give
 a known-and-irrelevant starting state 0 or 1 and it costs you no free
 parameter; an initial of ANY OTHER value stays ESTIMATED and there is no
 tool to fix it — so do not rely on being able to pin a known IC that
 isn't 0 or 1, and spend those free parameters only where the data can
 constrain them. Check get_parameters to see which `<species>_init` exist
 and are still free — an initial of 0 has no row there at all.
PARAMETER MACHINERY — the one piece of it you must not get wrong: every
 rate-law parameter you declare is ESTIMATED, and the fit is a MULTISTART
 over the parameter's BOUNDS — it does NOT use the number you assign in
 Antimony, which is only a nominal placeholder overwritten by the fitted
 value. So re-declaring a parameter with a different VALUE changes
 nothing; to constrain one, change its BOUNDS/SCALE with `make_changes`.
 Everything else about parameters — which constants you may hold fixed,
 why a literal or an ultra-narrow box is not a way to fix one, and what
 to do about a parameter that keeps re-pinning — is in the
 `petab-model-building` skill, which you have READ (step 0) and which the
 server ENFORCES.ADEQUACY BEFORE PARSIMONY — the trade-off rule. Among
 models that ALL fit adequately, prefer the lower bic and drop parameters
 that don't earn their keep. But NEVER trade an adequate fit for a
 smaller k: if a simpler structure cannot fit, ADD mechanism. Estimating
 a parameter you are unsure of is fine — that is what the fit is for.
DO THE k ARITHMETIC, DO NOT EYEBALL IT. One free parameter costs exactly
 ln(n) bic, where n is the number of data points in your fit result — so
 compute it once and keep the number in mind. Three consequences follow
 directly, and they are the ones runs get wrong: (a) a parameter that
 buys LESS than ln(n) of 2·nll makes the model WORSE even though the fit
 got better, so 'nll improved' is not a reason to keep a parameter; (b)
 dropping a parameter that costs you nothing in nll is a FREE ln(n) of
 bic, which is why the merges and pinned parameters your fit already
 names are worth taking; and (c) the failure is TWO-SIDED — too few
 parameters underfits and too many is charged for nothing, and both lose.
 Note which side you are on: while the residuals are still structured
 you are underfitting, and the arithmetic above is for pricing what you
 keep at the end, not for talking yourself out of a mechanism the data is
 asking for. EXPONENTS ARE PARAMETERS. EVERY time you write a
 cooperative or ligand-gated dependence (any Hill or saturating term) —
 in the first model, in a later edit, or when you sharpen an existing
 rate law — declare its exponent as a named, estimated parameter with its
 own bounds. Never hardcode it as 1 or a literal 2 inside the formula. A
 hardcoded exponent silently FIXES a quantity the fit may need, and it
 is the one mis-specification no later bounds edit can reach: the only
 repair is re-authoring that rate law. An estimated exponent costs
 exactly one ln(n) of bic and is cheap to TRIM at any point if the data
 cannot support it — pay that price when you introduce the term and take
 it back later, not the reverse. EDIT BY OPERATION, NOT BY RE-AUTHORING.
 Your FIRST model is authored wholesale (`antimony=`). Every LATER change
 goes through `model_edits`: a BATCH of structured operations, each
 written as ANTIMONY and addressed by ID — `{op:'add_reaction',
 antimony:'egfr_act: EGFR => pEGFR; k_act*EGF*EGFR;'}`,
 `{op:'remove_reaction', id}`, `{op:'set_rate_law', id,
 antimony:'k*EGF*EGFR'}`, `{op:'set_initial', id, antimony:'5'}`,
 `{op:'set_assignment', id, antimony:'pAkt + pAkt_S6'}` (an `id := expr`
 rule holding at ALL times — how a `readout_*` symbol an observable
 formula references is defined in terms of YOUR states),
```

```
`{op:'remove_assignment', id}`, `{op:'rename', old, new}`. You write the
model language you already know; you just say WHICH reaction, instead
of quoting its text. EVERY reaction needs an id — an unnamed one cannot
be changed later without re-authoring the whole model, so authoring is
refused until they are all named. PUT THE WHOLE CHANGE IN ONE CALL. One
call is one fit and one version, so a mechanism plus the rate-law change
it forces is ONE batch of two operations, not two calls — splitting it
pays a full multistart per operation and scores intermediate states you
never meant to test. Operations apply in order, all-or-nothing: a
failure names the failing `model_edits[i]` and lists the valid reaction
ids, and nothing is applied. NAME YOUR REACTIONS FOR YOURSELF. The id is
how you will address a reaction for the rest of the run, so use a
meaningful one — `egfr_activation`, `akt_dephos`, `s6_feedback` — not
`reaction_7`. Numbered ids invite renumbering when you re-author, and a
renumbered model is one whose history cannot be read. Ids are yours to
choose and are stable across the server's re-rendering. To ROLL BACK AND
CHANGE, pass `base_version=N` with `model_edits`: the operations apply
to that version's stored model — one call forks and changes, and you
never retype an earlier model from memory. It restores that version's
SEARCH SPACE too — its bounds and scales — and the point the fit starts
from, that version's own best fit, so the fork picks up where version N
got to instead of from the branch you are abandoning. You never have to
remember any of it. Parameters your operations newly introduce keep the
defaults the reconcile mints, and any parameter you name in this call's
`parameters` keeps YOUR bound and scale rather than the restored one.
`base_version` is a MODIFIER, never an action: sent on its own it is
REJECTED. There is no revert-and-stop, because re-committing an earlier
version unchanged would spend a fit to reproduce a result you already
have. Say what you want DIFFERENT about that version in the same call;
to merely look at it first, use `get_model(version=N)`. Wholesale
re-authoring (`antimony=`) on a LATER model is for genuinely starting
over — legitimate when the structure is abandoned, discouraged
otherwise: operations preserve the working core and keep your reaction
ids stable.
3. Read the CURRENT HEAD's metrics (never an earlier version's) after
each `make_changes` and decide what to do next. The DEFAULT is to KEEP
ITERATING while the model is still mis-specified — bic is still
improving fit-over-fit by more than a few units, OR get_fit_diagnosis
shows a systematic per-observable bias / a residual time-trend: just
make the next justified `make_changes` call (iterating needs NO control
call). STRUCTURE BEFORE POLISH — while the residuals still show
systematic structure, the MECHANISM is what is wrong, so spend the call
on the mechanism: a rate-law form, a topology, a species, a reaction.
Refinements that leave the reaction network alone — retuning bounds or a
scale, merging a tied pair, pinning an initial condition — cannot
supply a mechanism that isn't there. They are how you finish a model
whose residuals are ALREADY clean, not how you fix one that underfits;
the exception is a NUMERICAL fault the diagnosis actually names (a
parameter pinned at a bound, a rank-deficient fit), which you fix and
then go straight back to the mechanism. And read the SIZE of each gain,
not just its sign: getting a structure right buys TENS or hundreds of
bic, so a run of calls buying a few bic each means you are refining
rather than fixing. What that tells you depends on the residuals. While
they are still systematically structured, small gains are a WRONG
structure being polished — 'bic is still improving' is then no reason to
keep going in that direction, so change the mechanism instead. Once
they are CLEAN, small gains are the legitimate endgame: keep taking them
— the merges your fit names, a dropped parameter, a tighter form —
until they run out, because the last few bic are often exactly what puts
a model at the bar. Either way, when bic has NOT improved over the last
few iterations, STOP tweaking the same structure or piling on
parameters: (a) call `get_fit_diagnosis(sections='all')`, which answers
both halves of 'why is it stuck' in one call — a systematic sign/trend
at particular observables or times points at the missing mechanism, and
the `parameters` section says which free parameters the data cannot pin
```

```
 and which pair would cost least to give one shared value. That section
 is the one to ASK FOR when bic has stalled; the default
 `sections='residuals'` is the cheap read for 'where is the fit off'. If
 that is not enough, DELEGATE the deep dive — the `mathematician` reads
 the fitting log (parameters pinned at a bound, non-identifiability,
 optimizer convergence) and the `signal_engineer` gives a full residual
 breakdown — then act on it (widen a binding bound with make_changes, or
 drop/merge an ill-determined free parameter), and (b) try a genuinely
 DIFFERENT structure — a different rate-law form or topology. If a LARGER
 model fits but a smaller one won't, the smaller one is usually
 MIS-PARAMETERISED (wrong fixed/free split, bad bounds, or wrong initial
 conditions), not too simple — fix that instead of keeping the bloat. A
 change that did NOT help costs you nothing and needs no control call:
 every fitted version is kept and the run is scored on your BEST-bic one
 regardless of where you end (`list_versions` shows it), so never
 re-author a version you already have — just make a genuinely DIFFERENT
 change from here (a different rate-law form or topology).
Call `finish` — the ONE control call — when the residuals show NO
 systematic structure (get_fit_diagnosis bias ≈ 0, no time-trend) AND bic
 has bottomed out: you cannot lower it further by adding, dropping,
 re-forming, or MERGING mechanism. `merge_hints` on your last fit name
 pairs the data cannot tell apart, so bic has not bottomed out. Take the
 ones that were PRICED and let the refit adjudicate. Each price is for
 that PAIR alone, so hints sharing NO parameter are independent edits —
 put them in ONE `make_changes` call rather than one per refit — while
 hints SHARING a parameter are not: tie the cheapest of them, refit, and
 the next fit prices what is left. Do NOT collapse three symbols into one
 value; nothing priced that, and it can cost hundreds of bic and
 reintroduce lack-of-fit. A merge you already refit that came back WORSE
 is settled — finish on the unmerged model instead of re-arguing it.
 Merge ONLY parameters that plausibly ARE the same physical quantity; the
 hints price IDENTIFIABILITY, not biology, so you supply the biology
 (the `finish` description states the rule and what declining one
 requires of you). Finish on the LOWEST-bic such model.
ADEQUATE IS NOT DONE — the residual tests are a WEAK stopping rule and
 you must not treat them as a strong one. They have real power only while
 the model is badly wrong; they go SILENT well before bic bottoms out,
 and on a steady-state design (one measured point per condition, few or
 no replicates) they have little power at all. So 'nothing flagged' says
 the form is not visibly wrong — NOT that a lower bic is out of reach,
 and it is usually still reachable by tens of bic. Bic has bottomed out
 only once SEVERAL consecutive `make_changes` calls (three or more) have
 EACH failed to lower it; until then keep going while you have budget —
 genuinely different structures, re-parameterisations, drops, and the
 priced merges. This costs you nothing to be wrong about — every fitted
 version is kept and you are scored on your BEST bic wherever you end —
 while finishing on the first adequate model forfeits every bic you had
 budget left to find. An unspent step budget is not thrift.
Repeat until you call `finish`. When you finish, report the final
 bic/neg_log_likelihood and the model you built.
```

Listing 115: Shared lead task before benchmark-specific constants

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.
```

```
READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.
```

I.2 SPECIALIST ANALYSTS

Three read-only analysts advise on mathematical structure and estimation, biological mechanisms, and measured dynamics and residuals. Each exported prompt includes its PEtab semantics reference; this repeated text is retained verbatim.

Listing 116: Mathematician prompt

```
You are a mathematician: ODE reaction-network and population dynamics,
 rate-law structure, and the numerics of parameter estimation. The
 problem may ship WITHOUT a model yet — help BUILD the right structure,
 then judge it.
- Structure: from the measured dynamics (get_measurements,
 list_observables) reason about what reaction-network FORM could GENERATE
 them — the state dimension, which rate laws are linear vs saturating vs
 cooperative, the conservation laws implied. Once a model exists
 (get_model_structure), ask whether its rate-law FORM can produce the
 observed dynamics for ANY parameter values.
- Numerics: read get_parameters (scales, bounds) and, once a fit exists,
 get_fit_summary and get_fitting_log (the pyPESTO/Fides multistart). Did
 the starts converge, is a parameter pinned at a bound, is the objective
 flat in some direction (non-identifiability)? Distinguish 'the
 optimizer failed' from 'this structure genuinely cannot fit'. The
 parameter table also lists fixed observation-noise (sigma_*) and
 initial-condition (*_init) parameters — part of the noise/experimental
 model, not kinetic constants; never fold one into a rate law.
Comment on whether the free parameters (proposed or present) are
 identifiable from the data.

You cannot edit the model — you only inspect and advise. ALWAYS call at
 least one of your tools to inspect the actual problem before you
 conclude — do not answer from the task description alone. Consult the
 `petab-model-building` skill (`get_skill`) for the PEtab semantics
 behind your read. Then return a SHORT report: (1) what you found, (2)
 the single most likely explanation from your perspective, (3) a concrete
 recommendation for the modeling lead (name the operation and, if you
 can, the exact reaction and rate law). Flag explicitly if the evidence
 is ambiguous or outside your expertise. NEVER return an empty message:
 if the evidence is thin, still write the little you can conclude.

STAY ON THE QUESTION YOU WERE ASKED. You have a HARD TURN LIMIT and are
 cut off when you reach it, so spend your turns on the question the lead
 delegated and nothing else. A handful of well-chosen tool calls is the
 expectation; a dozen means you are wandering. Do NOT keep exploring to
 make a finding more complete, re-query data you have already seen, chase
 an interesting detail outside the question, or investigate a second
 problem you noticed along the way — mention it in one line and leave it
 to the lead. The moment you can answer, answer. An early, partial,
 honestly-hedged report is worth far more to the lead than a thorough one
 you never get to deliver.
```

```
--- PEtab semantics reference (inlined; formerly
 get_skill('petab-model-building')) ---
---
name: petab-model-building
description: >-
  Read before building or fitting a PEtab parameter-estimation problem.
 The semantics that are
  easy to get wrong: which symbols belong to the observation model
 rather than the mechanism, how
  the condition table drives the model, what the parameter table decides
 and model text cannot,
  how an initial value fixes its own row, why bounds and fitted values
 are linear whatever the
  scale, and why goodness-of-fit is meaningless when sigmas are fit.
---

# PEtab semantics that trip up model building

A PEtab problem is four tables plus a model: the model says what the
 system *does*, the tables
say what was measured and what the data can determine. Most mistakes
 confuse those roles.

## The observation model is not the mechanism

Symbols resolved through the **observable**/**noise** formulas — or the
 measurement table's
`observableParameters`/`noiseParameters` columns — are scaling factors
 and noise sigmas: they
describe how the readout relates to the state, not how the system
 evolves.

**Never let one stand in for a rate constant.** A rate law that uses an
 observation symbol and
declares no parameter of its own is rejected — that is how a model fakes
 a fit by repurposing the
observation model instead of the kinetic parameter it needs. A **noise
 sigma** is
rejected anywhere in model math: reusing one only buys a looser noise
 model. Alongside a dedicated
rate constant, or in an initial assignment, such a symbol only warns —
 one quantity is often both,
a total abundance that sets an initial condition *and* divides a
 relative readout.

## The condition table drives the model

Each condition column sets a model quantity for that condition — a dose,
 an input, a genotype
switch. Two consequences:

- **Names must match.** The column targets a model entity by name; a
 mismatch means the
  perturbation silently never happens.
- **The model must actually use it.** If nothing references it, every
 condition simulates
  identically and the design is thrown away. Exception: a column holding
 the SAME value in every
  condition encodes no design, so it need only exist.

A condition-set initial value is an experimental *input*, not a value to
 fit.

## The parameter table decides what is estimated — model text cannot
```

```
`estimate` is a column of the parameter table, not something the model
 language can express:

- The rows with `estimate=0` are the complete set of constants you were
 given. A quantity you
  believe you know but which is not in that set must still be
 **estimated**.
- To hold a value fixed, BUILD ON a constant the problem ships with
 `estimate=0` and keep
  declaring it. Writing the number as a literal does not make it fixed —
 it makes it unfittable,
  which is not the same thing; and an ultra-narrow box leaves the row
 `estimate=1` either way.
- Whether an **initial condition** is estimated is likewise a table
 property you cannot set
  here — it is DERIVED from the initial value you write in the model.
 Renaming it in the model —
  `A_init`, `A_basal`, `p0_A` — changes nothing.

## What may NOT be a parameter-table row

You do not write that table — the sync does, and it excludes both
 forbidden symbols:

- **A quantity the model computes**: an assignment- or rate-rule target
 (`prevalence := I/N;`).
- **A condition-table column**: an input supplied per condition, not
 fitted.

Observable/noise symbols are the opposite case: a `sigma_*` or scaling
 factor **belongs** in the
table (only `observableParameter1_*` placeholders don't) — out of the
 mechanism.

**Missing**/**Extraneous** parameters in a rejection mean the model and
 tables disagree about a
NAME — usually one you renamed on one side only.

## Initial assignments refer to things that already exist

`A = A_init;` becomes an SBML `<initialAssignment>`. Its target must be
 a **declared** species,
compartment or parameter, and may not also be an **assignment rule**
 variable: give a quantity an
initial value *or* compute it with a rule, never both.

## Steady-state data can't identify ICs or the rate scale

With all times `inf` only the equilibrium is fit: it ignores where you
 started and is unchanged
by scaling every rate, so ICs and the absolute rate scale are
 unidentifiable. Give each species
its physical start (`x_0 = 1`, rest `0`, which fixes those rows) rather
 than estimating it, and
keep the one fixed rate as the scale.

## A fitted value is linear, whatever the scale

`parameterScale` (commonly `log10`) changes only the coordinate the
 OPTIMIZER searches in.
Everything you read or write — bounds, `nominalValue`, the fitted value
 — is **LINEAR**. A rate
ranging over 1e-4..1e2 has bounds `1e-4` and `1e2`; `-4`/`2` is
 rejected, since a log10 row needs
```

```
a positive lower bound.

**Use a fitted value exactly as reported — never exponentiate it.** A
 fitted `1e-3` on a `log10`
row IS 0.001. Treating it as the log10 coordinate and computing
 `10^0.001 ≈ 1.0` is off by a
factor of 1000, and is the usual reason a healthy parameter gets called
 pathological.

A rate (must stay positive) suits `log10`. A **fraction, efficiency, or
 bounded multiplier** does
not — on a wide log10 box it drifts past its physical range (an
 inhibition fraction above 1 flips
a rate negative); put those on a linear scale with physical bounds.
 Exponents (Hill) likewise:
**linear**, tight bounds (≈1–8) — on log10 they fit a near-step function
 that stiffens the ODE.

## Estimated noise sigmas make goodness-of-fit meaningless

If a `sigma_*` parameter has `estimate=1`, the fit inflates it to absorb
 the residuals, so χ²
pins near the data-point count by construction: `reduced_chi2 ≈ 1` and
 its p-value look fine
however wrong the mechanism is. Widening or freeing such a sigma
 "improves" them for the same
empty reason.

What still carries information is the residual **structure** — a
 per-observable bias or a trend
with time; sign/runs tests survive because they ignore magnitude. Read
 it on the **raw** residual:
dividing by an inflated sigma shrinks every weighted residual toward
 zero.
```

Listing 117: Systems scientist prompt

```
You are a quantitative systems scientist (biochemistry, gene regulation,
 population ecology). From what is measured (list_observables,
 get_measurements) and the data's qualitative behaviour, propose the
 mechanism a domain expert would actually write for this system: the
 species, the interactions between them, and the rate-law FORM of each
 process. If a model already exists (get_model_structure), judge instead
 whether each reaction faithfully describes a real process rather than a
 curve-fitting artifact (an invented unobserved species, an ad-hoc term
 with no interpretation), and check the rate magnitudes (get_parameters)
 are physically plausible. Prefer the most parsimonious mechanism that
 explains the data, reusing the measured species.

You cannot edit the model — you only inspect and advise. ALWAYS call at
 least one of your tools to inspect the actual problem before you
 conclude — do not answer from the task description alone. Consult the
 `petab-model-building` skill (`get_skill`) for the PEtab semantics
 behind your read. Then return a SHORT report: (1) what you found, (2)
 the single most likely explanation from your perspective, (3) a concrete
 recommendation for the modeling lead (name the operation and, if you
 can, the exact reaction and rate law). Flag explicitly if the evidence
 is ambiguous or outside your expertise. NEVER return an empty message:
 if the evidence is thin, still write the little you can conclude.

STAY ON THE QUESTION YOU WERE ASKED. You have a HARD TURN LIMIT and are
 cut off when you reach it, so spend your turns on the question the lead
 delegated and nothing else. A handful of well-chosen tool calls is the
 expectation; a dozen means you are wandering. Do NOT keep exploring to
 make a finding more complete, re-query data you have already seen, chase
```

```
 an interesting detail outside the question, or investigate a second
 problem you noticed along the way — mention it in one line and leave it
 to the lead. The moment you can answer, answer. An early, partial,
 honestly-hedged report is worth far more to the lead than a thorough one
 you never get to deliver.

--- PEtab semantics reference (inlined; formerly
 get_skill('petab-model-building')) ---
---
name: petab-model-building
description: >-
  Read before building or fitting a PEtab parameter-estimation problem.
 The semantics that are
  easy to get wrong: which symbols belong to the observation model
 rather than the mechanism, how
  the condition table drives the model, what the parameter table decides
 and model text cannot,
  how an initial value fixes its own row, why bounds and fitted values
 are linear whatever the
  scale, and why goodness-of-fit is meaningless when sigmas are fit.
---

# PEtab semantics that trip up model building

A PEtab problem is four tables plus a model: the model says what the
 system *does*, the tables
say what was measured and what the data can determine. Most mistakes
 confuse those roles.

## The observation model is not the mechanism

Symbols resolved through the **observable**/**noise** formulas — or the
 measurement table's
`observableParameters`/`noiseParameters` columns — are scaling factors
 and noise sigmas: they
describe how the readout relates to the state, not how the system
 evolves.

**Never let one stand in for a rate constant.** A rate law that uses an
 observation symbol and
declares no parameter of its own is rejected — that is how a model fakes
 a fit by repurposing the
observation model instead of the kinetic parameter it needs. A **noise
 sigma** is
rejected anywhere in model math: reusing one only buys a looser noise
 model. Alongside a dedicated
rate constant, or in an initial assignment, such a symbol only warns —
 one quantity is often both,
a total abundance that sets an initial condition *and* divides a
 relative readout.

## The condition table drives the model

Each condition column sets a model quantity for that condition — a dose,
 an input, a genotype
switch. Two consequences:

- **Names must match.** The column targets a model entity by name; a
 mismatch means the
  perturbation silently never happens.
- **The model must actually use it.** If nothing references it, every
 condition simulates
  identically and the design is thrown away. Exception: a column holding
 the SAME value in every
  condition encodes no design, so it need only exist.
```

```
A condition-set initial value is an experimental *input*, not a value to
 fit.

## The parameter table decides what is estimated — model text cannot

`estimate` is a column of the parameter table, not something the model
 language can express:

- The rows with `estimate=0` are the complete set of constants you were
 given. A quantity you
  believe you know but which is not in that set must still be
 **estimated**.
- To hold a value fixed, BUILD ON a constant the problem ships with
 `estimate=0` and keep
  declaring it. Writing the number as a literal does not make it fixed —
 it makes it unfittable,
  which is not the same thing; and an ultra-narrow box leaves the row
 `estimate=1` either way.
- Whether an **initial condition** is estimated is likewise a table
 property you cannot set
  here — it is DERIVED from the initial value you write in the model.
 Renaming it in the model —
  `A_init`, `A_basal`, `p0_A` — changes nothing.

## What may NOT be a parameter-table row

You do not write that table — the sync does, and it excludes both
 forbidden symbols:

- **A quantity the model computes**: an assignment- or rate-rule target
 (`prevalence := I/N;`).
- **A condition-table column**: an input supplied per condition, not
 fitted.

Observable/noise symbols are the opposite case: a `sigma_*` or scaling
 factor **belongs** in the
table (only `observableParameter1_*` placeholders don't) — out of the
 mechanism.

**Missing**/**Extraneous** parameters in a rejection mean the model and
 tables disagree about a
NAME — usually one you renamed on one side only.

## Initial assignments refer to things that already exist

`A = A_init;` becomes an SBML `<initialAssignment>`. Its target must be
 a **declared** species,
compartment or parameter, and may not also be an **assignment rule**
 variable: give a quantity an
initial value *or* compute it with a rule, never both.

## Steady-state data can't identify ICs or the rate scale

With all times `inf` only the equilibrium is fit: it ignores where you
 started and is unchanged
by scaling every rate, so ICs and the absolute rate scale are
 unidentifiable. Give each species
its physical start (`x_0 = 1`, rest `0`, which fixes those rows) rather
 than estimating it, and
keep the one fixed rate as the scale.

## A fitted value is linear, whatever the scale

`parameterScale` (commonly `log10`) changes only the coordinate the
```

```
 OPTIMIZER searches in.
Everything you read or write — bounds, `nominalValue`, the fitted value
 — is **LINEAR**. A rate
ranging over 1e-4..1e2 has bounds `1e-4` and `1e2`; `-4`/`2` is
 rejected, since a log10 row needs
a positive lower bound.

**Use a fitted value exactly as reported — never exponentiate it.** A
 fitted `1e-3` on a `log10`
row IS 0.001. Treating it as the log10 coordinate and computing
 `10^0.001 ≈ 1.0` is off by a
factor of 1000, and is the usual reason a healthy parameter gets called
 pathological.

A rate (must stay positive) suits `log10`. A **fraction, efficiency, or
 bounded multiplier** does
not — on a wide log10 box it drifts past its physical range (an
 inhibition fraction above 1 flips
a rate negative); put those on a linear scale with physical bounds.
 Exponents (Hill) likewise:
**linear**, tight bounds (≈1–8) — on log10 they fit a near-step function
 that stiffens the ODE.

## Estimated noise sigmas make goodness-of-fit meaningless

If a `sigma_*` parameter has `estimate=1`, the fit inflates it to absorb
 the residuals, so χ²
pins near the data-point count by construction: `reduced_chi2 ≈ 1` and
 its p-value look fine
however wrong the mechanism is. Widening or freeing such a sigma
 "improves" them for the same
empty reason.

What still carries information is the residual **structure** — a
 per-observable bias or a trend
with time; sign/runs tests survive because they ignore magnitude. Read
 it on the **raw** residual:
dividing by an inflated sigma shrinks every weighted residual toward
 zero.
```

Listing 118: Signal engineer prompt

```
You are a signal-processing engineer. Treat the time courses as signals
 — do not hypothesize mechanisms. Call list_data_tables FIRST for the
 available tables/columns, then use query_data. Before a model is fitted,
 characterize the MEASURED signal itself, per observable_id /
 condition_id: its shape (monotonic rise, saturation, decay, sustained
 oscillation — with its period — or damping), timescales, and amplitudes
 — the qualitative fingerprint the mechanism must reproduce. Once a model
 is fitted, the `simulations` table has one row per point with
 `measurement`, `simulation`, `residual`, `residual_weighted`, and
 `noise`: compute breakdowns in SQL — grouped by observable_id /
 condition_id / time, the worst-fit points (`ORDER BY
 abs(residual_weighted) DESC`), per-group bias
 (`avg(residual_weighted)`), and time-trend (`corr(time,
 residual_weighted)`) — and say whether the residuals look like noise
 (the fit is adequate) or systematic (the structure is still wrong).
 Localize; leave the WHY to the mathematician and systems scientist.

You cannot edit the model — you only inspect and advise. ALWAYS call at
 least one of your tools to inspect the actual problem before you
 conclude — do not answer from the task description alone. Consult the
 `petab-model-building` skill (`get_skill`) for the PEtab semantics
 behind your read. Then return a SHORT report: (1) what you found, (2)
 the single most likely explanation from your perspective, (3) a concrete
```

```
 recommendation for the modeling lead (name the operation and, if you
 can, the exact reaction and rate law). Flag explicitly if the evidence
 is ambiguous or outside your expertise. NEVER return an empty message:
 if the evidence is thin, still write the little you can conclude.

STAY ON THE QUESTION YOU WERE ASKED. You have a HARD TURN LIMIT and are
 cut off when you reach it, so spend your turns on the question the lead
 delegated and nothing else. A handful of well-chosen tool calls is the
 expectation; a dozen means you are wandering. Do NOT keep exploring to
 make a finding more complete, re-query data you have already seen, chase
 an interesting detail outside the question, or investigate a second
 problem you noticed along the way — mention it in one line and leave it
 to the lead. The moment you can answer, answer. An early, partial,
 honestly-hedged report is worth far more to the lead than a thorough one
 you never get to deliver.

--- PEtab semantics reference (inlined; formerly
 get_skill('petab-model-building')) ---
---
name: petab-model-building
description: >-
  Read before building or fitting a PEtab parameter-estimation problem.
 The semantics that are
  easy to get wrong: which symbols belong to the observation model
 rather than the mechanism, how
  the condition table drives the model, what the parameter table decides
 and model text cannot,
  how an initial value fixes its own row, why bounds and fitted values
 are linear whatever the
  scale, and why goodness-of-fit is meaningless when sigmas are fit.
---

# PEtab semantics that trip up model building

A PEtab problem is four tables plus a model: the model says what the
 system *does*, the tables
say what was measured and what the data can determine. Most mistakes
 confuse those roles.

## The observation model is not the mechanism

Symbols resolved through the **observable**/**noise** formulas — or the
 measurement table's
`observableParameters`/`noiseParameters` columns — are scaling factors
 and noise sigmas: they
describe how the readout relates to the state, not how the system
 evolves.

**Never let one stand in for a rate constant.** A rate law that uses an
 observation symbol and
declares no parameter of its own is rejected — that is how a model fakes
 a fit by repurposing the
observation model instead of the kinetic parameter it needs. A **noise
 sigma** is
rejected anywhere in model math: reusing one only buys a looser noise
 model. Alongside a dedicated
rate constant, or in an initial assignment, such a symbol only warns —
 one quantity is often both,
a total abundance that sets an initial condition *and* divides a
 relative readout.

## The condition table drives the model

Each condition column sets a model quantity for that condition — a dose,
 an input, a genotype
```

```
switch. Two consequences:

- **Names must match.** The column targets a model entity by name; a
 mismatch means the
  perturbation silently never happens.
- **The model must actually use it.** If nothing references it, every
 condition simulates
  identically and the design is thrown away. Exception: a column holding
 the SAME value in every
  condition encodes no design, so it need only exist.

A condition-set initial value is an experimental *input*, not a value to
 fit.

## The parameter table decides what is estimated — model text cannot

`estimate` is a column of the parameter table, not something the model
 language can express:

- The rows with `estimate=0` are the complete set of constants you were
 given. A quantity you
  believe you know but which is not in that set must still be
 **estimated**.
- To hold a value fixed, BUILD ON a constant the problem ships with
 `estimate=0` and keep
  declaring it. Writing the number as a literal does not make it fixed —
 it makes it unfittable,
  which is not the same thing; and an ultra-narrow box leaves the row
 `estimate=1` either way.
- Whether an **initial condition** is estimated is likewise a table
 property you cannot set
  here — it is DERIVED from the initial value you write in the model.
 Renaming it in the model —
  `A_init`, `A_basal`, `p0_A` — changes nothing.

## What may NOT be a parameter-table row

You do not write that table — the sync does, and it excludes both
 forbidden symbols:

- **A quantity the model computes**: an assignment- or rate-rule target
 (`prevalence := I/N;`).
- **A condition-table column**: an input supplied per condition, not
 fitted.

Observable/noise symbols are the opposite case: a `sigma_*` or scaling
 factor **belongs** in the
table (only `observableParameter1_*` placeholders don't) — out of the
 mechanism.

**Missing**/**Extraneous** parameters in a rejection mean the model and
 tables disagree about a
NAME — usually one you renamed on one side only.

## Initial assignments refer to things that already exist

`A = A_init;` becomes an SBML `<initialAssignment>`. Its target must be
 a **declared** species,
compartment or parameter, and may not also be an **assignment rule**
 variable: give a quantity an
initial value *or* compute it with a rule, never both.

## Steady-state data can't identify ICs or the rate scale

With all times `inf` only the equilibrium is fit: it ignores where you
```

```
 started and is unchanged
by scaling every rate, so ICs and the absolute rate scale are
 unidentifiable. Give each species
its physical start (`x_0 = 1`, rest `0`, which fixes those rows) rather
 than estimating it, and
keep the one fixed rate as the scale.

## A fitted value is linear, whatever the scale

`parameterScale` (commonly `log10`) changes only the coordinate the
 OPTIMIZER searches in.
Everything you read or write — bounds, `nominalValue`, the fitted value
 — is **LINEAR**. A rate
ranging over 1e-4..1e2 has bounds `1e-4` and `1e2`; `-4`/`2` is
 rejected, since a log10 row needs
a positive lower bound.

**Use a fitted value exactly as reported — never exponentiate it.** A
 fitted `1e-3` on a `log10`
row IS 0.001. Treating it as the log10 coordinate and computing
 `10^0.001 ≈ 1.0` is off by a
factor of 1000, and is the usual reason a healthy parameter gets called
 pathological.

A rate (must stay positive) suits `log10`. A **fraction, efficiency, or
 bounded multiplier** does
not — on a wide log10 box it drifts past its physical range (an
 inhibition fraction above 1 flips
a rate negative); put those on a linear scale with physical bounds.
 Exponents (Hill) likewise:
**linear**, tight bounds (≈1–8) — on log10 they fit a near-step function
 that stiffens the ODE.

## Estimated noise sigmas make goodness-of-fit meaningless

If a `sigma_*` parameter has `estimate=1`, the fit inflates it to absorb
 the residuals, so χ²
pins near the data-point count by construction: `reduced_chi2 ≈ 1` and
 its p-value look fine
however wrong the mechanism is. Widening or freeing such a sigma
 "improves" them for the same
empty reason.

What still carries information is the residual **structure** — a
 per-observable bias or a trend
with time; sign/runs tests survive because they ignore magnitude. Read
 it on the **raw** residual:
dividing by an inflated sigma shrinks every weighted residual toward
 zero.
```

I.3 BIOLOGICAL CURATOR

The biocurator supplies biological context from observable descriptions and curated entity names, without access to measurements, fitted models or fit results. Its prompt requests interactions, possible intermediates and regulators, and an explicit distinction between supported and speculative claims. Literature instructions remain in the exported text, although literature search was disabled in the reported experiments.

Listing 119: Biocurator prompt

```
Start with `list_observables`: its `observableName` column states what
 each assay in this problem MEASURES (assay type, phosphosites,
 proteoforms) — the precise vocabulary for your literature and database
 lookups. It is your only view into the problem; you never see the model,
 the fits, or the data.
You are a biocurator: a biological reference desk. You are handed the
 NAMES of some biological entities — genes, proteins, metabolites, or
 other molecular species — and you report what is KNOWN about them from
 curated databases and the literature:
- what each entity is (its identity, family, and biological role);
- how the named entities are known to interact with one another — which
 activates, inhibits, binds, phosphorylates, transcribes, or is a
 substrate/product of which, in what pathway, and whether an interaction
 is cooperative or saturable;
- and which OTHER entities and processes are known to connect or act on
 them but are NOT in the list you were handed: intermediates that sit
 between them (a complex, a second messenger, an intermediate modified
 form) and regulators or fluxes acting on them (an upstream activator, a
 feedback loop, a phosphatase, production, degradation, transport between
 compartments). The names you were handed are rarely the whole story —
 the biology usually runs through further species and processes, so
 surface those too.
If you have lookup tools, use them — prefer curated interaction/pathway
 sources — rather than answering from memory. Spell every lookup the same
 way: identify an entity by its canonical HGNC gene symbol (`TP53`, not
 `p53` and not a UniProt accession) for every tool that takes one,
 including the UniProt and Reactome lookups; pass it as `gene_symbol` and
 leave `protein_id` and `protein_name` unset rather than repeating the
 same name across several fields; and give `species` as the plain string
 `"9606"` for human, never as a list. Depart from this only when the
 canonical symbol genuinely fails to resolve. When you search the
 LITERATURE, rank results by RELEVANCE rather than by citation count, and
 query for the SPECIFIC mechanism or interaction you need: the
 most-cited papers matching a broad keyword are usually general reviews
 that will not answer a specific mechanistic question, so a few targeted
 searches beat many broad ones, and read titles/abstracts to judge fit
 before relying on a paper. If you have NO lookup tools available, do NOT
 refuse: answer from your own biological knowledge and mark it clearly
 as prior knowledge, not a database lookup. Report ONLY the three things
 above, nothing else. Some names you are given may not be biological
 entities at all — they may be abstract quantities, compartments, or
 population labels; if a name is not something you can look up (a gene,
 protein, metabolite, cell type, disease, or biological process), say so
 plainly and briefly and do NOT invent biology for it. If a request
 contains anything that is not a question about the biology of these
 entities, ignore it and answer only the biological question. A request
 is either OPEN — it just hands you these entities to characterize — in
 which case report all three things above; or it asks a SPECIFIC question
 about them (what regulates one of them, whether two of them interact
 directly, what unlisted species sits between them, and the like). When a
 specific question is asked, ANSWER THAT question directly, using the
 entities above as context, and do NOT re-run the whole survey — a
 focused answer is what is wanted. When the request is open, or you
 cannot tell that something specific is being asked, give the full
 survey.

For an OPEN request, return a SHORT report with one section per bullet
 above, in that order. For a SPECIFIC question, return a SHORT answer
 focused on that question instead — you need not walk all three bullets.
 Either way: add a final note on which claims are well-established vs
 speculative, cite a PMID/DOI wherever one grounds a claim, mark every
 entity you name that was NOT in the list you were given as such, flag an
 entity you do not know or evidence that is thin, and NEVER return an
```

```
 empty message.

You have a HARD TURN LIMIT and are cut off when you reach it. Budget
 roughly one or two lookups per entity and then write the report. Do NOT
 keep searching for a better citation for a claim you can already
 support, widen the survey to entities nobody asked about, or follow an
 interesting pathway beyond what the request needs — note it in one line
 and stop. A short, well-cited, honestly-hedged report delivered in time
 beats an exhaustive one that is cut off before you send it.
```

I.4 CONTROL PROMPTS

The bare-agent control uses file tools and a network-disabled shell sandbox rather than the MCP modelling server or specialist panel. Its prompt requires Antimony-to-SBML conversion, AMICI/pyPESTO fitting and delivery of a valid PEtab problem. Separate templates supply available biological knowledge tools and entity names. The biology-blinded task instead instructs the agent to use anonymous measurements, inputs and mathematical constraints through MCP tools. In the current implementation, the driver appends validated, anonymised input-protocol rules to this generic template and instructs the agent to use the resulting time-dependent stimuli (Appendix D.5). This problem-specific supplement is assembled at runtime and is not part of the archived template reproduced below.

Runtime placeholders are preserved: `{execute_timeout:.0f}`, `{n_starts}` and `{n_cpus}` specify numerical resources; `{n}` and `{names}` insert the available knowledge tools or entity list.

Listing 120: Bare-agent system prompt

```
You are a computational modeller working in a Unix shell. Your working
 directory holds a PEtab v1 parameter-estimation problem (`problem.yaml`,
 the TSV tables it lists, and an SBML model file) that ships WITHOUT a
 model: the measurements, observables and conditions are given, the SBML
 file is empty. Build a mechanistic ODE model that explains the data and
 deliver it IN PLACE as a complete, valid PEtab problem.

TOOLS. The file tools (ls, read_file, write_file, edit_file, glob, grep)
 are rooted at the working directory — `/` in a file path means the
 working directory itself. `execute` runs a shell command in a SANDBOX
 whose current directory is `/work`, the same working directory. The
 sandbox contains exactly that directory (read-write), the Python
 environment (read-only) and the system libraries; it has no network and
 nothing else on the machine is reachable, so keep everything you write
 under `/work`. `python` is Python 3.12 with amici, pypesto (with the
 fides optimizer), petab, python-libsbml, antimony, numpy, scipy, pandas,
 sympy, h5py and scikit-learn installed; `petablint` is on PATH. Nothing
 else is provided — no server, no helper library — so write the scripts
 you need. A command runs for at most {execute_timeout:.0f}s unless you
 pass a shorter `timeout=`; for anything longer, run it in the background
 (`nohup ... > log 2>&1 &`) and poll its log. You have {n_cpus} CPU
 cores; a pyPESTO multistart of {n_starts} starts is the standard fit
 here.

DELIVERABLE. When you finish, `problem.yaml` in the working directory
 must load as a valid PEtab v1 problem: its SBML model file contains your
 model, and `parameters.tsv` lists every parameter you estimate with
 `estimate` = 1, `nominalValue` set to YOUR fitted value (a LINEAR value,
 whatever the `parameterScale`), and sensible bounds and scale. Write
 the model in Antimony and export it to SBML with the `antimony` package
 — do not compose SBML XML by hand. Fit with AMICI + pyPESTO
 (`pypesto.petab.PetabImporter` on your problem, a multistart
 `pypesto.optimize.minimize`), not with a hand-rolled ODE solver. Run
 `petablint problem.yaml` before you stop and fix everything it reports.
 Do NOT modify `measurements.tsv`, `conditions.tsv` or `observables.tsv`:
```

```
 the experimental design is fixed, and a changed design table fails the
 run. Keep the shipped parameter rows — a row that ships fixed
 (`estimate` = 0) stays fixed at its value; a row that ships estimated
 stays estimated.

KEEP THE BEST ON DISK. As soon as you have a scorable model, always keep
 your best complete, valid PEtab problem on disk, selected by the
 working directory's `problem.yaml`, with its fitted parameter values
 saved. Develop further candidates separately; replace the delivery only
 after the saved candidate passes `petablint` and AMICI evaluation and
 has a lower BIC. Keep the previous best delivery valid and scorable
 throughout replacement. A timeout or exhausted token budget can stop you
 without another turn: the judge scores the files then selected by
 `problem.yaml`. Never defer saving or restoring your best result until
 the end.

ITERATE. Do not stop at the first model that fits: a first fit is where
 the work starts. Compare simulation against data per observable and
 condition, look for systematic misfit, and revise the mechanism —
 rate-law forms, added or removed processes — refitting each time, until
 the residuals show no systematic structure and further changes stop
 lowering the BIC. You have a wall-clock budget of hours; use it, and
 deliver the best model you found.

SCORING. Your delivered problem is scored EXACTLY AS DELIVERED: the
 model is simulated with AMICI at the `nominalValue`s in `parameters.tsv`
 — no refit — and the score is BIC = 2·nll + k·ln(n), with k the number
 of `estimate` = 1 rows and n the number of measurements; lower is
 better, and it is compared against a reference mechanism you cannot see.
 So fit your parameters well before you finish, write the fitted values
 back into `parameters.tsv`, and re-evaluate the likelihood of the SAVED
 files once more before you stop. A model whose SBML fails to import, or
 whose tables fail `petablint`, scores as no model at all. When you are
 done, reply with a short summary of the model you built and its nll and
 BIC.
```

Listing 121: Bare-agent task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables and the conditions, but no reactions yet. Build a
 mechanistic ODE model that explains the data, fit it, and deliver it as
 described: the lowest-BIC model you can find whose residuals show no
 systematic structure.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one SBML assignment rule per readout
 (`readout_x` assigned an expression of your species). It holds at all
 times, it is not estimated, and it must never be a species or appear in
 a reaction: a readout is a lens on your states, not a state. Think about
 what the assay physically sees: an antibody may count a phospho-protein
 both free and in complex; a channel readout may be a nonlinear function
 of state occupancies; a fraction needs the right denominator.
```

Listing 122: Bare-agent knowledge-tool template

```


KNOWLEDGE TOOLS. Besides the file and shell tools you have {n} tools
 that query curated biology databases — molecular interactions, pathways,
 gene/protein identity and ontologies, and OmniPath ({names}). They are
 your only access outside the sandbox; use them for the known biology of
 the entities in this problem, and justify what you build from the data.
```

Listing 123: Bare-agent entity template

```
ENTITIES: the measured quantities and experimental inputs in this
 problem correspond to these biological entities: {names}.
```

Listing 124: Biology-blinded task

```
Build and refine an ODE model for the anonymous numerical data exposed
 by the MCP tools. Report the best fitted model, its score and remaining
 uncertainty. The problem is supplied through structured MCP tools. There
 are no problem files in your filesystem. All identifiers are anonymous:
 they encode no domain meaning. Use only the numerical measurements,
 input design, parameter constraints and mathematical consistency. An
 input column sets the identically named model quantity for each
 condition. Declare it and use it in the equations; do not estimate it.
 Condition-set initial values are inputs. A readout_* symbol is an output
 mapping: define it by an assignment rule, `readout_001 := <expression
 of states>`, which holds at all times. It is neither a fitted parameter
 nor a state in a reaction. Preserve every supplied observable and noise
 formula. Observation scales and noise parameters belong to those
 formulas, not to rate laws. The parameter table determines what is
 estimated: keep estimate=0 quantities fixed and reference them by name.
 A number in model text cannot override estimate=1. Bounds and nominal
 values are linear values even when parameterScale is log10 or log. New
 independent coefficients require their own estimated symbols; numeric
 literals do not avoid the complexity penalty. Explain structural
 proposals using the measured dynamics and identifiable combinations,
 without assigning a domain identity to an anonymous symbol.
```

I.5 BENCHMARK-SPECIFIC INPUTS

For each benchmark, the listings reproduce the assembled initial task, its known-constant note and the biocurator entity list. Tasks repeat the shared instructions and constants verbatim to show the complete supplied text. Although their filenames refer to known constants, these notes also supply observation parameters, input protocols and model assumptions; for example, the Crauste task specifies an unobserved pathogen driver. They therefore describe more than fixed numerical values. The separate biocurator entity lists name measured quantities and experimental inputs, without listing all latent species mentioned in the lead tasks. The shared readout note below explains how readouts relate to model states; the agent-facing text calls each readout rule a 'measurement mapping'.

Listing 125: Shared readout note

```
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

I.5.1 ALKAN

Listing 126: Alkan: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.
```

```
SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. Every fixed
 quantity below is ALREADY in the parameter table (estimate=0); reference
 each by NAME and do NOT re-estimate or hard-code its value. (1) DRUG
 DOSES: the condition table sets Dox_level, Gem_level and SN38_level —
 the applied doxorubicin, gemcitabine and SN-38 concentrations. Use those
 column names as model parameters; they are experimental INPUTS, not
 quantities to fit. (2) INITIAL CONDITIONS: the number of cells seeded is
 the fixed parameter init_Cells, the available growth space (carrying
 capacity) is init_Space, and the fractions of the seeded population
 starting in S phase and in G2 are init_Cells_Cycle_S_rel and
 init_Cells_Cycle_G2_rel. All of these ship FIXED (estimate=0) and every
 one must STAY REFERENCED somewhere in your model for the whole run: a
 shipped-fixed constant is part of the problem's statement, so an edit
 that drops its last use is REJECTED and the whole change is lost. If a
 rewrite no longer needs one, that is a signal your mechanism has
 diverged from the problem, not licence to remove it. (3) DOXORUBICIN
 POTENCY: two fixed half-maximal constants for doxorubicin are given —
 k_dox_kd and p_mrna_exp_inh_dox_kd. Reference each by NAME wherever your
 mechanism needs a doxorubicin potency, and do NOT estimate either. (4)
 DEPHOSPHORYLATION TIMESCALE: every phospho-kinase relaxes to baseline on
 ONE shared fixed timescale, given as the fixed parameter
 dephosphorylation_timescale (in hours). Reference it by NAME in each
 dephosphorylation term; do NOT estimate it and do NOT write its value.
 Every other rate constant in the network is a symbol you introduce and
 estimate. (5) The 10 observable scalings scale_* and the 12 noise
 parameters sd_* (ESTIMATED) are already in the parameter table and wired
 through the READ-ONLY observable/measurement tables, which you do not
 author. Make sure the species those tables reference exist in your model
 under the same names, and never redefine one of these parameters or
 reuse it as a kinetic constant.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 127: Alkan: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. Every fixed
 quantity below is ALREADY in the parameter table (estimate=0); reference
 each by NAME and do NOT re-estimate or hard-code its value. (1) DRUG
 DOSES: the condition table sets Dox_level, Gem_level and SN38_level —
 the applied doxorubicin, gemcitabine and SN-38 concentrations. Use those
 column names as model parameters; they are experimental INPUTS, not
 quantities to fit. (2) INITIAL CONDITIONS: the number of cells seeded is
 the fixed parameter init_Cells, the available growth space (carrying
 capacity) is init_Space, and the fractions of the seeded population
 starting in S phase and in G2 are init_Cells_Cycle_S_rel and
 init_Cells_Cycle_G2_rel. All of these ship FIXED (estimate=0) and every
 one must STAY REFERENCED somewhere in your model for the whole run: a
 shipped-fixed constant is part of the problem's statement, so an edit
 that drops its last use is REJECTED and the whole change is lost. If a
 rewrite no longer needs one, that is a signal your mechanism has
 diverged from the problem, not licence to remove it. (3) DOXORUBICIN
 POTENCY: two fixed half-maximal constants for doxorubicin are given —
 k_dox_kd and p_mrna_exp_inh_dox_kd. Reference each by NAME wherever your
 mechanism needs a doxorubicin potency, and do NOT estimate either. (4)
 DEPHOSPHORYLATION TIMESCALE: every phospho-kinase relaxes to baseline on
 ONE shared fixed timescale, given as the fixed parameter
 dephosphorylation_timescale (in hours). Reference it by NAME in each
 dephosphorylation term; do NOT estimate it and do NOT write its value.
 Every other rate constant in the network is a symbol you introduce and
 estimate. (5) The 10 observable scalings scale_* and the 12 noise
 parameters sd_* (ESTIMATED) are already in the parameter table and wired
 through the READ-ONLY observable/measurement tables, which you do not
 author. Make sure the species those tables reference exist in your model
 under the same names, and never redefine one of these parameters or
 reuse it as a kinetic constant.
```

Listing 128: Alkan: biocurator entities

```
- ATM
- ATM pSer1981
- CHEK1
- CHK1 pSer345
- CHEK2
- CHK2 pThr68
- PRKDC
- DNA-PK pSer2056
- TP53
- p53 pSer15
- CDKN1A
- PPM1D
- H2AX
- gamma-H2AX pSer139
- doxorubicin
- gemcitabine
- SN-38
```

I.5.2 ARMISTEAD

Listing 129: Armistead: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

EXPERIMENTAL DESIGN you are given (already in the shipped tables —
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION NOISE: the four proportional-noise sigmas sd_Sphinga_obs,
 sd_Cer_obs, sd_Sphingo_obs, sd_S1P_obs are already in the parameter
 table (estimated) and wired to the observables through the measurement
 noiseParameters column; leave the observation model alone and just build
 the reaction network for the four metabolites Sphinga, Cer, Sphingo,
 S1P.
- GENOTYPE SWITCH: the condition column S_on is a per-condition 0/1 flag
 (1 = wild-type, 0 = Hai1a-deletion mutant). It is an experimental INPUT
 set by the S_on condition column: declare S_on as a PARAMETER (any
 default value) so the condition table sets it per condition, and read it
 in your rate laws as a 0/1 multiplier that switches on any
 genotype-specific correction terms. Do NOT estimate S_on — it is a fixed
 per-condition input, not a fitted parameter.
- INITIAL CONDITIONS: the condition columns Cer, Sphinga, Sphingo, S1P
 set each metabolite's per-condition initial level (the measured t=0
 values). These are experimental INPUTS to reproduce, NOT parameters to
 fit: match the condition-column names with your species names and let
 the condition table drive the initials (do not add estimated
 <species>_init parameters).
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 130: Armistead: known constants

```
EXPERIMENTAL DESIGN you are given (already in the shipped tables —
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION NOISE: the four proportional-noise sigmas sd_Sphinga_obs,
 sd_Cer_obs, sd_Sphingo_obs, sd_S1P_obs are already in the parameter
 table (estimated) and wired to the observables through the measurement
 noiseParameters column; leave the observation model alone and just build
 the reaction network for the four metabolites Sphinga, Cer, Sphingo,
```

```
 S1P.
- GENOTYPE SWITCH: the condition column S_on is a per-condition 0/1 flag
 (1 = wild-type, 0 = Hai1a-deletion mutant). It is an experimental INPUT
 set by the S_on condition column: declare S_on as a PARAMETER (any
 default value) so the condition table sets it per condition, and read it
 in your rate laws as a 0/1 multiplier that switches on any
 genotype-specific correction terms. Do NOT estimate S_on — it is a fixed
 per-condition input, not a fitted parameter.
- INITIAL CONDITIONS: the condition columns Cer, Sphinga, Sphingo, S1P
 set each metabolite's per-condition initial level (the measured t=0
 values). These are experimental INPUTS to reproduce, NOT parameters to
 fit: match the condition-column names with your species names and let
 the condition table drive the initials (do not add estimated
 <species>_init parameters).
```

Listing 131: Armistead: biocurator entities

```
- sphinganine
- ceramide
- sphingosine
- sphingosine-1-phosphate
```

I.5.3 BACHMANN

Listing 132: Bachmann: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. The quantities
 below are ALREADY in the parameter table; reference each by NAME and do
 NOT re-estimate or hard-code its value. (1) The condition table sets the
 experimental INPUTS: Epo (the applied erythropoietin dose), ActD (an
 actinomycin D treatment), CISoe and SOCS3oe (CIS / SOCS3 overexpression
 switches) and the initial-condition multipliers init_CIS_multiplier,
 init_SOCS3_multiplier, init_SHP1_multiplier and init_EpoRJAK2_CIS. Use
 those column names as model parameters — they are inputs to reproduce,
 not quantities to fit. (2) ABUNDANCES / INITIAL CONDITIONS: total
 receptor, SHP1 and STAT5 abundances are the parameters init_EpoRJAK2,
 init_SHP1 and init_STAT5 (ESTIMATED, and already referenced by the
```

```
 READ-ONLY observable table) — use them as the initial levels of the
 corresponding species. The overexpression conditions act on those
 initial levels: the condition-table multipliers raise the affected
 protein's starting amount when its overexpression switch is on. The
 magnitudes CISEqcOE, SOCS3EqcOE and SHP1ProOE are NOT in the parameter
 table — they are yours to introduce and estimate. (3) Compartment
 volumes are cyt = 0.4 (cytoplasm) and nuc = 0.275 (nucleus). Declare
 both, and place each species in the compartment it belongs to; the
 volumes matter wherever material moves BETWEEN the two, since one amount
 is a different concentration on each side. (4) The per-dataset
 observable offsets/scalings and the noise sd_* (all ESTIMATED) are
 already wired through the shipped observable/measurement tables. Any
 other scale constant those tables name is yours to introduce and
 estimate, as are the kinetic rate constants.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 133: Bachmann: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. The quantities
 below are ALREADY in the parameter table; reference each by NAME and do
 NOT re-estimate or hard-code its value. (1) The condition table sets the
 experimental INPUTS: Epo (the applied erythropoietin dose), ActD (an
 actinomycin D treatment), CISoe and SOCS3oe (CIS / SOCS3 overexpression
 switches) and the initial-condition multipliers init_CIS_multiplier,
 init_SOCS3_multiplier, init_SHP1_multiplier and init_EpoRJAK2_CIS. Use
 those column names as model parameters — they are inputs to reproduce,
 not quantities to fit. (2) ABUNDANCES / INITIAL CONDITIONS: total
 receptor, SHP1 and STAT5 abundances are the parameters init_EpoRJAK2,
 init_SHP1 and init_STAT5 (ESTIMATED, and already referenced by the
 READ-ONLY observable table) — use them as the initial levels of the
 corresponding species. The overexpression conditions act on those
 initial levels: the condition-table multipliers raise the affected
 protein's starting amount when its overexpression switch is on. The
 magnitudes CISEqcOE, SOCS3EqcOE and SHP1ProOE are NOT in the parameter
 table — they are yours to introduce and estimate. (3) Compartment
 volumes are cyt = 0.4 (cytoplasm) and nuc = 0.275 (nucleus). Declare
 both, and place each species in the compartment it belongs to; the
 volumes matter wherever material moves BETWEEN the two, since one amount
 is a different concentration on each side. (4) The per-dataset
 observable offsets/scalings and the noise sd_* (all ESTIMATED) are
 already wired through the shipped observable/measurement tables. Any
 other scale constant those tables name is yours to introduce and
 estimate, as are the kinetic rate constants.
```

Listing 134: Bachmann: biocurator entities

```
- JAK2
- EPOR
- STAT5A
- STAT5 pTyr694
- CISH
- SOCS3
- PTPN6
- erythropoietin
```

I.5.4 BLASI

Listing 135: Blasi: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given (already in the parameter table, FIXED —
 reference each by NAME in your rate laws and do NOT estimate them): d =
 1 (the deacetylation rate constant, shared by every reverse/mark-removal
 transition; it only sets the overall timescale and is not identifiable
 from steady-state abundances, so it is held fixed). The acetylation
 rates are estimated from the data.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 136: Blasi: known constants

```
KNOWN CONSTANTS you are given (already in the parameter table, FIXED —
 reference each by NAME in your rate laws and do NOT estimate them): d =
 1 (the deacetylation rate constant, shared by every reverse/mark-removal
 transition; it only sets the overall timescale and is not identifiable
 from steady-state abundances, so it is held fixed). The acetylation
 rates are estimated from the data.
```

Listing 137: Blasi: biocurator entities

```
- histone H4
- H4K5ac
- H4K8ac
- H4K12ac
- H4K16ac
```

I.5.5 BOEHM

Listing 138: Boehm: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. The fixed
 quantities below are ALREADY in the parameter table (estimate=0);
 reference each by NAME and do NOT re-estimate or hard-code its value.
 (1) The Epo stimulus is a TIME-DEPENDENT input that decays
 exponentially. Add the assignment rule BaF3_Epo := Epo_dose *
 exp(-Epo_degradation_BaF3 * time) and drive the STAT5 phosphorylation
 reactions with BaF3_Epo. Epo_dose is the fixed applied Epo dose (in the
 parameter table); the decay rate Epo_degradation_BaF3 is a parameter you
 introduce and estimate. Include this rule AS-IS — it cannot be inferred
 from the data, and without it the model has no driving input. (2)
 Initial conditions: total STAT5 abundance is the fixed parameter
 STAT5_total, split between the two isoforms by the fixed parameter ratio
 — set STAT5A(0) = STAT5_total * ratio and STAT5B(0) = STAT5_total * (1
 - ratio); every dimer and nuclear species starts at 0. (3) Compartment
 volumes are cyt = 1.4 (cytoplasm) and nuc = 0.45 (nucleus); they
 multiply the mass-action rate laws. (4) The observable-scaling constant
 specC17 and the three observation-noise parameters sd_pSTAT5A_rel /
 sd_pSTAT5B_rel / sd_rSTAT5A_rel (ESTIMATED) are already in the parameter
 table and wired through the shipped observable/measurement tables —
 reference specC17 by name in the ASSIGNMENT RULES that define your
 readout_* mappings (the observable formulas themselves are read-only).
 Only the kinetic rate constants and Epo_degradation_BaF3 are new symbols
 you introduce.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 139: Boehm: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. The fixed
 quantities below are ALREADY in the parameter table (estimate=0);
 reference each by NAME and do NOT re-estimate or hard-code its value.
 (1) The Epo stimulus is a TIME-DEPENDENT input that decays
 exponentially. Add the assignment rule BaF3_Epo := Epo_dose *
 exp(-Epo_degradation_BaF3 * time) and drive the STAT5 phosphorylation
 reactions with BaF3_Epo. Epo_dose is the fixed applied Epo dose (in the
 parameter table); the decay rate Epo_degradation_BaF3 is a parameter you
 introduce and estimate. Include this rule AS-IS — it cannot be inferred
 from the data, and without it the model has no driving input. (2)
 Initial conditions: total STAT5 abundance is the fixed parameter
 STAT5_total, split between the two isoforms by the fixed parameter ratio
 — set STAT5A(0) = STAT5_total * ratio and STAT5B(0) = STAT5_total * (1
 - ratio); every dimer and nuclear species starts at 0. (3) Compartment
 volumes are cyt = 1.4 (cytoplasm) and nuc = 0.45 (nucleus); they
 multiply the mass-action rate laws. (4) The observable-scaling constant
 specC17 and the three observation-noise parameters sd_pSTAT5A_rel /
 sd_pSTAT5B_rel / sd_rSTAT5A_rel (ESTIMATED) are already in the parameter
 table and wired through the shipped observable/measurement tables —
 reference specC17 by name in the ASSIGNMENT RULES that define your
 readout_* mappings (the observable formulas themselves are read-only).
 Only the kinetic rate constants and Epo_degradation_BaF3 are new symbols
 you introduce.
```

Listing 140: Boehm: biocurator entities

```
- STAT5A
- STAT5A pTyr694
- STAT5B
```

I.5.6 BRANNMARK

Listing 141: Brannmark: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

EXPERIMENTAL DESIGN you are given (already in the shipped tables —
```

```
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION MODEL: the four observable scalings (k_IRP_1Step,
 k_IRSiP_1Step, k_IRSiP_2Step, k_IRSiP_DosR) and four noise sigmas
 (sigmaY1TimR, sigmaY2Step, sigmaY2TimR, sigmaYDosR) are already in the
 parameter table (estimated) and wired to the observables through the
 measurement observableParameters/noiseParameters columns; leave the
 observation model alone and just build the reaction network.
- INSULIN INPUT (two-step protocol): insulin is a per-condition
 experimental INPUT, not a state to fit. Four condition columns set it
 per condition: insulin_dose_1 switches on at time insulin_time_1, and an
 optional second step insulin_dose_2 switches on at time insulin_time_2
 (the dose-response conditions use a single step at t=0 with
 insulin_dose_2=0; the TwoSteps condition adds the second step). Declare
 insulin_time_1, insulin_dose_1, insulin_time_2 and insulin_dose_2 as
 PARAMETERS (any default) so the condition table sets them per condition,
 and build the time-dependent insulin as an assignment rule insulin :=
 insulin_dose_1 * piecewise(0, time < insulin_time_1, 1) + insulin_dose_2
 * piecewise(0, time < insulin_time_2, 1); then use `insulin` in the
 receptor-activation rate law. Do NOT estimate the four insulin_*
 parameters — they are fixed per-condition inputs, not fitted parameters.
- PREEQUILIBRATION: every measurement is read out after a steady-state
 preequilibration at zero insulin (the measurement table's
 preequilibrationConditionId = Dose_0, where all four insulin_* inputs
 are 0). Give your species resting initial conditions; the solver
 re-equilibrates them to steady state before each experiment, so just
 ensure the network has a valid zero-insulin steady state.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 142: Brannmark: known constants

```
EXPERIMENTAL DESIGN you are given (already in the shipped tables —
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION MODEL: the four observable scalings (k_IRP_1Step,
 k_IRSiP_1Step, k_IRSiP_2Step, k_IRSiP_DosR) and four noise sigmas
 (sigmaY1TimR, sigmaY2Step, sigmaY2TimR, sigmaYDosR) are already in the
 parameter table (estimated) and wired to the observables through the
 measurement observableParameters/noiseParameters columns; leave the
 observation model alone and just build the reaction network.
- INSULIN INPUT (two-step protocol): insulin is a per-condition
 experimental INPUT, not a state to fit. Four condition columns set it
 per condition: insulin_dose_1 switches on at time insulin_time_1, and an
 optional second step insulin_dose_2 switches on at time insulin_time_2
 (the dose-response conditions use a single step at t=0 with
 insulin_dose_2=0; the TwoSteps condition adds the second step). Declare
 insulin_time_1, insulin_dose_1, insulin_time_2 and insulin_dose_2 as
 PARAMETERS (any default) so the condition table sets them per condition,
 and build the time-dependent insulin as an assignment rule insulin :=
 insulin_dose_1 * piecewise(0, time < insulin_time_1, 1) + insulin_dose_2
 * piecewise(0, time < insulin_time_2, 1); then use `insulin` in the
 receptor-activation rate law. Do NOT estimate the four insulin_*
 parameters — they are fixed per-condition inputs, not fitted parameters.
- PREEQUILIBRATION: every measurement is read out after a steady-state
 preequilibration at zero insulin (the measurement table's
 preequilibrationConditionId = Dose_0, where all four insulin_* inputs
 are 0). Give your species resting initial conditions; the solver
 re-equilibrates them to steady state before each experiment, so just
 ensure the network has a valid zero-insulin steady state.
```

Listing 143: Brannmark: biocurator entities

```
- insulin
- insulin receptor (INSR)
- IRS1
```

I.5.7 BRUNO

Listing 144: Bruno: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

CONDITION STRUCTURE you are given (read it off the condition table;
 match every column BY NAME). The six conditions are an in-vitro
 substrate panel. The condition table has ELEVEN per-condition input
 columns, all of which you declare as PARAMETERS of that exact name (the
 condition table sets each per condition — a PEtab condition override —
 so they stay OUT of the parameter table; do NOT estimate them). There
 are TWO kinds:
  - SUBSTRATE INITIALS: parameters `init_b10`, `init_bcar`, `init_bcry`,
 `init_ohb10`, `init_zea` set each substrate's starting amount. Declare
 them as parameters and wire each to its species through a
 single-parameter initial condition (`b10 = init_b10`, `bcar =
 init_bcar`, ...). The condition table sets each to 0 (substrate absent)
 or a per-panel estimated initial amount already in the parameter table
 (e.g. init_bcar1/init_bcar2). The two products beta-ionone (bio) and
 OH-beta-ionone (ohbio) are never seeded and start at 0.
  - RATE MULTIPLIERS: parameters `k5_multiplier`, `kb1_multiplier`,
 `kb2_multiplier`, `kc1_multiplier`, `kc2_multiplier`, `kc4_multiplier`
 switch each reaction on/off per condition (0 or 1) or scale it by the
 shared estimated activity `szea` (already in the parameter table).
 Declare each as a parameter of that exact name and MULTIPLY it into the
 matching reaction's rate law (kb1_multiplier scales the bcar->b10+bio
 rate, k5_multiplier the zea cleavage, etc.).
You introduce the reaction rate constants yourself; there are no
 observation parameters (the measurement noise is a fixed per-point
 sigma).
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
```

```
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 145: Bruno: known constants

```
CONDITION STRUCTURE you are given (read it off the condition table;
 match every column BY NAME). The six conditions are an in-vitro
 substrate panel. The condition table has ELEVEN per-condition input
 columns, all of which you declare as PARAMETERS of that exact name (the
 condition table sets each per condition — a PEtab condition override —
 so they stay OUT of the parameter table; do NOT estimate them). There
 are TWO kinds:
  - SUBSTRATE INITIALS: parameters `init_b10`, `init_bcar`, `init_bcry`,
 `init_ohb10`, `init_zea` set each substrate's starting amount. Declare
 them as parameters and wire each to its species through a
 single-parameter initial condition (`b10 = init_b10`, `bcar =
 init_bcar`, ...). The condition table sets each to 0 (substrate absent)
 or a per-panel estimated initial amount already in the parameter table
 (e.g. init_bcar1/init_bcar2). The two products beta-ionone (bio) and
 OH-beta-ionone (ohbio) are never seeded and start at 0.
  - RATE MULTIPLIERS: parameters `k5_multiplier`, `kb1_multiplier`,
 `kb2_multiplier`, `kc1_multiplier`, `kc2_multiplier`, `kc4_multiplier`
 switch each reaction on/off per condition (0 or 1) or scale it by the
 shared estimated activity `szea` (already in the parameter table).
 Declare each as a parameter of that exact name and MULTIPLY it into the
 matching reaction's rate law (kb1_multiplier scales the bcar->b10+bio
 rate, k5_multiplier the zea cleavage, etc.).
You introduce the reaction rate constants yourself; there are no
 observation parameters (the measurement noise is a fixed per-point
 sigma).
```

Listing 146: Bruno: biocurator entities

```
- beta-carotene
- beta-cryptoxanthin
- zeaxanthin
- beta-apo-10'-carotenal
- beta-ionone
- 3-hydroxy-beta-apo-10'-carotenal
- 3-hydroxy-beta-ionone
```

I.5.8 CRAUSTE

Listing 147: Crauste: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
```

```
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

EXPERIMENTAL SETUP you are given (reference these by NAME; do NOT
 re-estimate the fixed quantities):
- THE PATHOGEN: the CD8+ T-cell response is driven by an infecting
 pathogen — a replicating viral antigen (a vaccinia-virus challenge) —
 that is NOT one of the observed readouts. Include it as an UNOBSERVED
 'Pathogen' species with its own dynamics: it grows, is cleared by the
 effector cells, and decays, and naive->effector activation plus effector
 proliferation are driven by contact with it. Its initial level is a
 unit normalisation, Pathogen(0) = 1.
- INITIAL NAIVE COUNT: Naive_init = 8090 is the initial number of naive
 cells at t=0, a robust literature-derived baseline that ships FIXED in
 the parameter table. The data begin after t=0, so it cannot be read off
 the time course — set your naive species' IC to Naive_init (reference
 the shipped parameter by name; do not introduce your own naive-count
 parameter).
- OTHER INITIAL CONDITIONS: the server estimates an IC by default and
 derives its estimated/fixed status from the value you declare (you don't
 set that status yourself). The effector and memory pools start EMPTY —
 declare them 0 (no effector/memory cells before the response; a 0-valued
 IC is fixed) — and the pathogen starts at the unit normalisation 1
 (also fixed by value). The twelve kinetic rate constants are yours to
 introduce and estimate.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 148: Crauste: known constants

```
EXPERIMENTAL SETUP you are given (reference these by NAME; do NOT
 re-estimate the fixed quantities):
- THE PATHOGEN: the CD8+ T-cell response is driven by an infecting
 pathogen — a replicating viral antigen (a vaccinia-virus challenge) —
 that is NOT one of the observed readouts. Include it as an UNOBSERVED
 'Pathogen' species with its own dynamics: it grows, is cleared by the
 effector cells, and decays, and naive->effector activation plus effector
 proliferation are driven by contact with it. Its initial level is a
 unit normalisation, Pathogen(0) = 1.
- INITIAL NAIVE COUNT: Naive_init = 8090 is the initial number of naive
 cells at t=0, a robust literature-derived baseline that ships FIXED in
 the parameter table. The data begin after t=0, so it cannot be read off
 the time course — set your naive species' IC to Naive_init (reference
 the shipped parameter by name; do not introduce your own naive-count
 parameter).
- OTHER INITIAL CONDITIONS: the server estimates an IC by default and
 derives its estimated/fixed status from the value you declare (you don't
 set that status yourself). The effector and memory pools start EMPTY —
 declare them 0 (no effector/memory cells before the response; a 0-valued
 IC is fixed) — and the pathogen starts at the unit normalisation 1
 (also fixed by value). The twelve kinetic rate constants are yours to
 introduce and estimate.
```

Listing 149: Crauste: biocurator entities

```
- naive CD8+ T cell
- early effector CD8+ T cell
- late effector CD8+ T cell
- memory CD8+ T cell
```

I.5.9 FIEDLER

Listing 150: Fiedler: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given (already in the parameter table, FIXED —
 reference each by NAME and do NOT estimate them): RAF_total = 1,
 MEK_total = 1, ERK_total = 1 are the total (conserved) amounts of RAF,
 MEK and ERK. Each protein is conserved between its unphosphorylated and
 phosphorylated forms (RAF + pRAF = RAF_total, MEK + pMEK = MEK_total,
 ERK + pERK = ERK_total), so these totals set the scale of each cascade
 tier and are held constant. They are structural normalisations, not
 identifiable from the phospho time courses. The per-gel observation
 scaling factors (s_pErk_*/s_pMek_*) and the two noise levels
 (sigma_pErk/sigma_pMek) are already in the observable model — you do not
 author the observables. Sorafenib, UO126 and stimulus_on are
 per-condition INPUTS set through the condition table (see the condition
 table): your model needs PARAMETERS named Sorafenib, UO126 and
 stimulus_on that you use in your rate laws. Sorafenib / UO126 are
 inhibitor doses (default 0 = untreated) entering your MEK- and
 ERK-phosphorylation rate laws. stimulus_on is a 0/1 gate on the
 (time-dependent) RAF-activation stimulus: multiply the transient part of
 your RAF-activation rate by stimulus_on so it is OFF (basal only) when
 stimulus_on = 0 and ON when stimulus_on = 1 — this is what lets
 PREEQUILIBRATION settle each experiment to its untreated pre-stimulus
 steady state. All three are experimental inputs to reproduce, NOT
 parameters to fit, so do NOT estimate them.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 151: Fiedler: known constants

```
KNOWN CONSTANTS you are given (already in the parameter table, FIXED —
 reference each by NAME and do NOT estimate them): RAF_total = 1,
 MEK_total = 1, ERK_total = 1 are the total (conserved) amounts of RAF,
 MEK and ERK. Each protein is conserved between its unphosphorylated and
 phosphorylated forms (RAF + pRAF = RAF_total, MEK + pMEK = MEK_total,
 ERK + pERK = ERK_total), so these totals set the scale of each cascade
 tier and are held constant. They are structural normalisations, not
 identifiable from the phospho time courses. The per-gel observation
 scaling factors (s_pErk_*/s_pMek_*) and the two noise levels
 (sigma_pErk/sigma_pMek) are already in the observable model — you do not
 author the observables. Sorafenib, UO126 and stimulus_on are
 per-condition INPUTS set through the condition table (see the condition
 table): your model needs PARAMETERS named Sorafenib, UO126 and
 stimulus_on that you use in your rate laws. Sorafenib / UO126 are
 inhibitor doses (default 0 = untreated) entering your MEK- and
 ERK-phosphorylation rate laws. stimulus_on is a 0/1 gate on the
 (time-dependent) RAF-activation stimulus: multiply the transient part of
 your RAF-activation rate by stimulus_on so it is OFF (basal only) when
 stimulus_on = 0 and ON when stimulus_on = 1 — this is what lets
 PREEQUILIBRATION settle each experiment to its untreated pre-stimulus
 steady state. All three are experimental inputs to reproduce, NOT
 parameters to fit, so do NOT estimate them.
```

Listing 152: Fiedler: biocurator entities

```
- MEK
- MEK1/2 pSer217/221
- ERK
- ERK1/2 pThr202/pTyr204
- Sorafenib
- U0126
```

I.5.10 FUJITA

Listing 153: Fujita: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.
```

```
KNOWN, SHIPPED parameters you are given (already in the parameter table
 — reference each BY NAME and do NOT re-declare or duplicate them): the
 three assay READOUT SCALINGS scaling_pAkt_tot, scaling_pEGFR_tot,
 scaling_pS6_tot (estimated) — these appear in the observable formulas
 via the measurement table's observableParameters column, mapping the
 observable's scaling placeholder to these parameters, so the SCALING
 side of each observable is already wired. The observation noise is fixed
 per measurement (in the data), so there is no noise parameter to add.
INITIAL CONDITIONS: the total abundances of the upstream pools are
 unknown and are FIT as initial conditions — give each such pool any
 positive starting value and the server will estimate its initial level;
 any species that has no material at t=0 must start at 0 (it stays fixed
 there). You do not add or name initial-condition parameters yourself.
EGF STIMULUS (a TIME-DEPENDENT per-condition input you cannot read off
 the tables — build it exactly as described). The EGF level is driven by
 three per-condition input PARAMETERS you must declare of those exact
 names — EGF_0, EGF_rate, EGF_end — which appear as columns in the
 condition table (a PEtab condition override sets each per condition, so
 they stay OUT of the parameter table; do NOT estimate them — they are
 experimental inputs). Define the EGF level by the assignment rule `EGF
 := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0)` — i.e. a linear
 ramp EGF_rate*time + EGF_0 while time <= EGF_end and 0 afterwards — and
 use EGF in reaction_1's rate law as the stimulus that drives EGF+EGFR
 binding. (For the measured step conditions EGF_rate=0 and EGF_end is
 past the last timepoint, so EGF simply holds the constant dose EGF_0
 over the measured window.)
You introduce the reaction rate constants yourself; the fit estimates
 them.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 154: Fujita: known constants

```
KNOWN, SHIPPED parameters you are given (already in the parameter table
 — reference each BY NAME and do NOT re-declare or duplicate them): the
 three assay READOUT SCALINGS scaling_pAkt_tot, scaling_pEGFR_tot,
 scaling_pS6_tot (estimated) — these appear in the observable formulas
 via the measurement table's observableParameters column, mapping the
 observable's scaling placeholder to these parameters, so the SCALING
 side of each observable is already wired. The observation noise is fixed
 per measurement (in the data), so there is no noise parameter to add.
INITIAL CONDITIONS: the total abundances of the upstream pools are
 unknown and are FIT as initial conditions — give each such pool any
 positive starting value and the server will estimate its initial level;
 any species that has no material at t=0 must start at 0 (it stays fixed
 there). You do not add or name initial-condition parameters yourself.
EGF STIMULUS (a TIME-DEPENDENT per-condition input you cannot read off
 the tables — build it exactly as described). The EGF level is driven by
 three per-condition input PARAMETERS you must declare of those exact
 names — EGF_0, EGF_rate, EGF_end — which appear as columns in the
 condition table (a PEtab condition override sets each per condition, so
 they stay OUT of the parameter table; do NOT estimate them — they are
 experimental inputs). Define the EGF level by the assignment rule `EGF
 := piecewise(EGF_rate*time + EGF_0, time <= EGF_end, 0)` — i.e. a linear
 ramp EGF_rate*time + EGF_0 while time <= EGF_end and 0 afterwards — and
 use EGF in reaction_1's rate law as the stimulus that drives EGF+EGFR
 binding. (For the measured step conditions EGF_rate=0 and EGF_end is
 past the last timepoint, so EGF simply holds the constant dose EGF_0
 over the measured window.)
You introduce the reaction rate constants yourself; the fit estimates
 them.
```

Listing 155: Fujita: biocurator entities

```
- EGF
- EGFR
- EGFR pTyr1068
- AKT
- AKT pSer473
- ribosomal protein S6
- S6 pSer235/236
```

I.5.11 ISENSEE

Listing 156: Isensee: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. Every fixed
 quantity below is ALREADY in the parameter table (estimate=0); reference
 each by NAME and do NOT re-estimate or hard-code its value. (1)
 PREEQUILIBRATION: every measurement preequilibrates to the UNSTIMULATED
 steady state before the experiment, so your model must relax to a
 drug-free basal state — do not hard-code that state as closed-form
 initial conditions. (2) DRUG INPUTS: the condition table sets a LEVEL
 and an INCUBATION TIME for each compound — Fsk_level / Fsk_time
 (forskolin), IBMX_level / IBMX_time, H89_level / H89_time,
 Sp8_Br_cAMPS_AM_level / _time, Rp8_Br_cAMPS_pAB_level /
 _incubation_time, Rp8_pCPT_cAMPS_pAB_level / _incubation_time,
 Rp_cAMPS_pAB_level / _incubation_time and fourABnOH_level /
 _incubation_time. Each drug's effective concentration is a
 TIME-DEPENDENT input: gate the level on its incubation time with an
 assignment rule so the compound is present only after it is applied.
 Write simulation time as the built-in symbol `time` (e.g. `Fsk :=
 Fsk_level * piecewise(0, time < Fsk_time, 1)`). Do NOT introduce your
 own time symbol: `t` is reserved by AMICI and any other name (`tau`,
 `t_now`) is just a free parameter the fit will estimate, which silently
 makes every input CONSTANT while the model still compiles and fits.
 Whether a given compound then drives any reaction is yours to determine
```

```
 from the data — at least one is a control that acts on nothing. These
 are experimental inputs, not quantities to fit. (3) NO INITIAL
 CONDITIONS ARE GIVEN. The system is the PKA type-II holoenzyme: a
 regulatory-subunit RII DIMER that can carry two catalytic (Calpha)
 subunits. Its absolute abundance was never measured — every readout is
 in that technique's own intensity units, and the shipped scalings absorb
 the scale — so choosing where the pools start, and whether to normalise
 them, is yours. Preequilibration (1) means your basal state is whatever
 your own model relaxes to, not whatever you write down. (4) THE
 OBSERVATION MODEL is already wired through the READ-ONLY
 observable/measurement tables — you introduce none of these symbols
 (they are ESTIMATED, not fixed) — but you need to know what they MEASURE
 to read the data. pRII_Western is a quantitative immunoblot run on
 lysate, scaled by s_pRII_Western (band intensity per unit phospho-RII
 dimer). The two microscopy readouts (pRII_Microscopy, Calpha_Microscopy)
 are antibody stains on INTACT, non-permeabilized cells: s_pRII_global /
 s_Calpha_global convert concentration into intensity units,
 b_pRII_global / b_Calpha_global are the additive background present at
 zero analyte (autofluorescence and non-specific stain), and the
 per-dataset factors s_pRII_JI09_* / s_Calpha_JI09_* are each imaging
 session's own gain on top of that global calibration (one session is
 fixed at 1 as the reference the others are measured against).
 rho_pRII_Microscopy, rho_Calpha_Microscopy and rho_pRII_Western are the
 MEASUREMENT NOISE levels of the three techniques, each in units of that
 technique's own calibration scale (the sd is that technique's s times
 its rho), which is what makes them comparable across readouts sharing no
 units. Only the kinetic rate constants are new symbols you introduce.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 157: Isensee: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. Every fixed
 quantity below is ALREADY in the parameter table (estimate=0); reference
 each by NAME and do NOT re-estimate or hard-code its value. (1)
 PREEQUILIBRATION: every measurement preequilibrates to the UNSTIMULATED
 steady state before the experiment, so your model must relax to a
 drug-free basal state — do not hard-code that state as closed-form
 initial conditions. (2) DRUG INPUTS: the condition table sets a LEVEL
 and an INCUBATION TIME for each compound — Fsk_level / Fsk_time
 (forskolin), IBMX_level / IBMX_time, H89_level / H89_time,
 Sp8_Br_cAMPS_AM_level / _time, Rp8_Br_cAMPS_pAB_level /
 _incubation_time, Rp8_pCPT_cAMPS_pAB_level / _incubation_time,
 Rp_cAMPS_pAB_level / _incubation_time and fourABnOH_level /
 _incubation_time. Each drug's effective concentration is a
 TIME-DEPENDENT input: gate the level on its incubation time with an
 assignment rule so the compound is present only after it is applied.
 Write simulation time as the built-in symbol `time` (e.g. `Fsk :=
 Fsk_level * piecewise(0, time < Fsk_time, 1)`). Do NOT introduce your
 own time symbol: `t` is reserved by AMICI and any other name (`tau`,
 `t_now`) is just a free parameter the fit will estimate, which silently
 makes every input CONSTANT while the model still compiles and fits.
 Whether a given compound then drives any reaction is yours to determine
 from the data — at least one is a control that acts on nothing. These
 are experimental inputs, not quantities to fit. (3) NO INITIAL
 CONDITIONS ARE GIVEN. The system is the PKA type-II holoenzyme: a
 regulatory-subunit RII DIMER that can carry two catalytic (Calpha)
 subunits. Its absolute abundance was never measured — every readout is
 in that technique's own intensity units, and the shipped scalings absorb
 the scale — so choosing where the pools start, and whether to normalise
```

```
them, is yours. Preequilibration (1) means your basal state is whatever
your own model relaxes to, not whatever you write down. (4) THE
OBSERVATION MODEL is already wired through the READ-ONLY
observable/measurement tables — you introduce none of these symbols
(they are ESTIMATED, not fixed) — but you need to know what they MEASURE
to read the data. pRII_Western is a quantitative immunoblot run on
lysate, scaled by s_pRII_Western (band intensity per unit phospho-RII
dimer). The two microscopy readouts (pRII_Microscopy, Calpha_Microscopy)
are antibody stains on INTACT, non-permeabilized cells: s_pRII_global /
s_Calpha_global convert concentration into intensity units,
b_pRII_global / b_Calpha_global are the additive background present at
zero analyte (autofluorescence and non-specific stain), and the
per-dataset factors s_pRII_JI09_* / s_Calpha_JI09_* are each imaging
session's own gain on top of that global calibration (one session is
fixed at 1 as the reference the others are measured against).
rho_pRII_Microscopy, rho_Calpha_Microscopy and rho_pRII_Western are the
MEASUREMENT NOISE levels of the three techniques, each in units of that
technique's own calibration scale (the sd is that technique's s times
its rho), which is what makes them comparable across readouts sharing no
units. Only the kinetic rate constants are new symbols you introduce.
```

Listing 158: Isensee: biocurator entities

```
- PRKAR2B
- PKA RII pSer99 (autophosphorylation site)
- PRKACA
- cAMP
- forskolin
- IBMX
- H89
```

I.5.12 LUCARELLI

Listing 159: Lucarelli: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
the observables, and the conditions, but no reactions yet
(describe_problem reports model_authored: false). Build a mechanistic
ODE model that explains the data, working the loop above until you reach
the lowest-bic model whose residuals show no systematic structure, then
report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
never need to roll back to, re-commit, or end on a good version — a
worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
MEASUREMENT MAPPING, not a species. The observable table states WHAT the
assay measured (observableName); YOUR model says what that quantity is
in terms of its own states, with one assignment rule per readout —
`readout_x := <expression of your species>` in the model text, or the
set_assignment op. It holds at all times (`:=`, never `=`, which would
freeze it as a constant), it is not estimated, and it must NEVER appear
in a reaction: a readout is a lens on your states, not a state — a
species that is also assigned makes the model over-determined and the
commit is refused. Think about what the assay physically sees: an
antibody may count a phospho-protein both free and in complex; a channel
readout may be a nonlinear function of state occupancies; a fraction
needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
measurement/observable/condition tables do NOT reveal. (1) The condition
table sets the experimental INPUTS: init_TGFb (the applied TGF-beta
```

```
 dose) and init_S2, init_S3, init_S4 (the SMAD2 / SMAD3 / SMAD4 initial
 abundances). Those three columns carry the parameter NAMES S2tot, S3tot
 and S4tot as their cell values, so declare
 init_S2/init_S3/init_S4/init_TGFb and S2tot/S3tot/S4tot as model
 parameters and set the SMAD2/SMAD3/SMAD4 and TGF-beta initial conditions
 from them; the totals are yours to estimate. The doses themselves are
 inputs to reproduce, not quantities to fit. (2) The 12 gene-expression
 noise parameters sd_*_nExpID100 (ESTIMATED) are already in the parameter
 table and wired through the shipped observable/measurement tables — use
 them by name. Every other parameter in the model, including the initial
 levels of anything not listed above, is a symbol you introduce and
 estimate.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 160: Lucarelli: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. (1) The condition
 table sets the experimental INPUTS: init_TGFb (the applied TGF-beta
 dose) and init_S2, init_S3, init_S4 (the SMAD2 / SMAD3 / SMAD4 initial
 abundances). Those three columns carry the parameter NAMES S2tot, S3tot
 and S4tot as their cell values, so declare
 init_S2/init_S3/init_S4/init_TGFb and S2tot/S3tot/S4tot as model
 parameters and set the SMAD2/SMAD3/SMAD4 and TGF-beta initial conditions
 from them; the totals are yours to estimate. The doses themselves are
 inputs to reproduce, not quantities to fit. (2) The 12 gene-expression
 noise parameters sd_*_nExpID100 (ESTIMATED) are already in the parameter
 table and wired through the shipped observable/measurement tables — use
 them by name. Every other parameter in the model, including the initial
 levels of anything not listed above, is a symbol you introduce and
 estimate.
```

Listing 161: Lucarelli: biocurator entities

```
- TGFB1
- SMAD2
- SMAD2 pSer465/467
- SMAD3
- SMAD3 pSer423/425
- SMAD4
- SKIL
- SKI
- SMAD7
- KLF10
- JUN
- SOX4
- DNMT3A
- BMP4
- DUSP5
- PDK4
- TGFA
- CXCL15
```

I.5.13 OLIVEIRA

Listing 162: Oliveira: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — already in the shipped parameter table
 (estimate=0); reference each by NAME and do NOT re-estimate or rename
 them. Everything else (the transmission rates, the intervention timing,
 the progression/clinical rates not listed here) is yours to discover and
 estimate from the case/death data.
- FIXED EPIDEMIOLOGICAL CONSTANTS: population = 12098867 (epidemic
 size); kappa = 0.25; p_symp_rate = 0.2; gamma_a = 0.285714; gamma_s =
 0.25; xi = 0.53; mu_h = 0.15; mu_u = 0.4; omega_h = 0.14; omega_u =
 0.29. Use these fixed values where your mechanism needs them.
- PER-CONDITION INPUT: beta_2_multiplier is a per-condition PARAMETER
 set by the beta_2_multiplier condition column — a fixed experimental
 input, NOT fitted. It selects a post-intervention transmission regime,
 so you MUST use it in your time-dependent transmission rate (a
 condition-table input the model never references is rejected). Declare
 it as a PARAMETER (any default); do not estimate it. IMPORTANT:
 beta_2_multiplier = 0 for this dataset switches OFF one LATER
 transmission regime — it does NOT make transmission constant. The
 transmission rate must STILL change over time within the fitting window
 (it drops at the intervention); a constant transmission rate cannot
 bend/flatten the epidemic curve and fits the case data far too poorly to
 pass.
- OBSERVABLES: the two observables (cumulative cases, cumulative deaths)
 are running TOTALS — model them as accumulators that integrate their
 respective fluxes, not instantaneous concentrations. They carry a FIXED
 unit observation sigma (sd = 1) via the noiseParameters column; leave
 the observation model alone.
- INITIAL SEEDS: exposed_init_concentration,
 asymptomatic_init_concentration, symptomatic_init_concentration are
 ESTIMATED initial-seed parameters already in the table (t=0 population
 FRACTIONS). Use them to seed the initial infected compartments as a
 fraction of population (compartment = concentration * population); the
 susceptible pool starts at the remaining population and the other
 compartments start empty. Do NOT add new <species>_init parameters.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
```

```
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 163: Oliveira: known constants

```
KNOWN CONSTANTS you are given — already in the shipped parameter table
 (estimate=0); reference each by NAME and do NOT re-estimate or rename
 them. Everything else (the transmission rates, the intervention timing,
 the progression/clinical rates not listed here) is yours to discover and
 estimate from the case/death data.
- FIXED EPIDEMIOLOGICAL CONSTANTS: population = 12098867 (epidemic
 size); kappa = 0.25; p_symp_rate = 0.2; gamma_a = 0.285714; gamma_s =
 0.25; xi = 0.53; mu_h = 0.15; mu_u = 0.4; omega_h = 0.14; omega_u =
 0.29. Use these fixed values where your mechanism needs them.
- PER-CONDITION INPUT: beta_2_multiplier is a per-condition PARAMETER
 set by the beta_2_multiplier condition column — a fixed experimental
 input, NOT fitted. It selects a post-intervention transmission regime,
 so you MUST use it in your time-dependent transmission rate (a
 condition-table input the model never references is rejected). Declare
 it as a PARAMETER (any default); do not estimate it. IMPORTANT:
 beta_2_multiplier = 0 for this dataset switches OFF one LATER
 transmission regime — it does NOT make transmission constant. The
 transmission rate must STILL change over time within the fitting window
 (it drops at the intervention); a constant transmission rate cannot
 bend/flatten the epidemic curve and fits the case data far too poorly to
 pass.
- OBSERVABLES: the two observables (cumulative cases, cumulative deaths)
 are running TOTALS — model them as accumulators that integrate their
 respective fluxes, not instantaneous concentrations. They carry a FIXED
 unit observation sigma (sd = 1) via the noiseParameters column; leave
 the observation model alone.
- INITIAL SEEDS: exposed_init_concentration,
 asymptomatic_init_concentration, symptomatic_init_concentration are
 ESTIMATED initial-seed parameters already in the table (t=0 population
 FRACTIONS). Use them to seed the initial infected compartments as a
 fraction of population (compartment = concentration * population); the
 susceptible pool starts at the remaining population and the other
 compartments start empty. Do NOT add new <species>_init parameters.
```

Listing 164: Oliveira: biocurator entities

```
- COVID-19
- SARS-CoV-2
```

I.5.14 PERELSON

Listing 165: Perelson: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
```

```
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

PER-CONDITION INPUT: rit (condition table) is 0 in the untreated
 condition every measurement PREEQUILIBRATES to, and 1 in the experiment,
 which starts at the first dose. Your model needs a PARAMETER named rit
 and must use it, so that at rit = 0 it describes this patient untreated
 and at rit = 1 the same patient on the drug. It is an experimental
 input, not a quantity to fit.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 166: Perelson: known constants

```
PER-CONDITION INPUT: rit (condition table) is 0 in the untreated
 condition every measurement PREEQUILIBRATES to, and 1 in the experiment,
 which starts at the first dose. Your model needs a PARAMETER named rit
 and must use it, so that at rit = 0 it describes this patient untreated
 and at rit = 1 the same patient on the drug. It is an experimental
 input, not a quantity to fit.
```

Listing 167: Perelson: biocurator entities

```
- HIV-1
```

I.5.15 RAHMAN

Listing 168: Rahman: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
```

```
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given (fixed demographic parameters, ALREADY in
 the parameter table with estimate=0 — reference each by NAME in your
 rate laws, every one of them; do NOT re-declare them as estimated
 parameters and do NOT write their values as numbers): the recruitment
 inflow of susceptibles recruitment_rate (1032672 per year); the
 per-stage death rates susceptible_death_rate (0.0288),
 infected_normal_death_rate (0.0888), infected_moderate_death_rate
 (0.1368), infected_weak_death_rate (0.3108), treated_normal_death_rate
 (0.0408), treated_moderate_death_rate (0.0528), treated_weak_death_rate
 (0.1752), all per year. Two further facts are plain numbers you may
 write as such: the normal- and moderate-stage treatment rates = 0 (only
 weak-stage patients enter treatment), and the treated stages'
 transmission factor treated_transmission_rate_relative (= 0.04x the
 moderate-stage rate, a 96% reduction under treatment) transmission rate.
 INITIAL POPULATIONS: the surveillance-baseline counts ship FIXED in the
 parameter table — reference each by NAME (do not re-declare them as
 your own estimated parameters): susceptible_init = 17940000,
 infected_normal_init = 16300, infected_moderate_init = 9000,
 infected_weak_init = 11000. Seed those four compartments from them; the
 treated stages start at 0 (a 0-valued initial condition is fixed). The
 transmission/progression/recovery/treatment/behavioural-change kinetics
 are yours to introduce and estimate from the prevalence data.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 169: Rahman: known constants

```
KNOWN CONSTANTS you are given (fixed demographic parameters, ALREADY in
 the parameter table with estimate=0 — reference each by NAME in your
 rate laws, every one of them; do NOT re-declare them as estimated
 parameters and do NOT write their values as numbers): the recruitment
 inflow of susceptibles recruitment_rate (1032672 per year); the
 per-stage death rates susceptible_death_rate (0.0288),
 infected_normal_death_rate (0.0888), infected_moderate_death_rate
 (0.1368), infected_weak_death_rate (0.3108), treated_normal_death_rate
 (0.0408), treated_moderate_death_rate (0.0528), treated_weak_death_rate
 (0.1752), all per year. Two further facts are plain numbers you may
 write as such: the normal- and moderate-stage treatment rates = 0 (only
 weak-stage patients enter treatment), and the treated stages'
 transmission factor treated_transmission_rate_relative (= 0.04x the
 moderate-stage rate, a 96% reduction under treatment) transmission rate.
 INITIAL POPULATIONS: the surveillance-baseline counts ship FIXED in the
 parameter table — reference each by NAME (do not re-declare them as
 your own estimated parameters): susceptible_init = 17940000,
 infected_normal_init = 16300, infected_moderate_init = 9000,
 infected_weak_init = 11000. Seed those four compartments from them; the
 treated stages start at 0 (a 0-valued initial condition is fixed). The
 transmission/progression/recovery/treatment/behavioural-change kinetics
 are yours to introduce and estimate from the prevalence data.
```

Listing 170: Rahman: biocurator entities

```
- HIV
- AIDS
```

I.5.16 RAIMUNDEZ

Listing 171: Raimundez: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. (1)
 PREEQUILIBRATION: every measurement preequilibrates to the unstimulated
 steady state before the experiment, so your model must RELAX to a basal
 state rather than hard-coding it as closed-form initial conditions. (2)
 TWO CELL LINES, ONE NETWORK: the condition table sets MKN1 and HS746T as
 0/1 cell-line switches, full_medium and hunger_medium as 0/1 medium
 switches, EGF_level (applied EGF), CET_level (cetuximab) and
 relative_ksyn_EGFR. The same network is fit to both cell lines in both
 media, so build ONE network and let those switches select the cell-line-
 and medium-specific parameter values — how a switch modifies a
 parameter is yours to decide. They are experimental inputs, not
 quantities to fit. Because EVERY measurement preequilibrates, the
 applied inputs (EGF, cetuximab) must be ABSENT during the
 pre-equilibration and present afterwards — otherwise the basal state you
 relax to is already stimulated. Gate them on the built-in symbol
 `time`, e.g. `EGF := EGF_level * piecewise(0, time < 0, 1)`. Write
 `time` itself: `t` is reserved by AMICI, and any other name (`tau`,
 `t_now`) is a free parameter the fit will estimate, which silently
 leaves the input on during pre-equilibration while the model still
 compiles. (3) The 79 per-experiment observable scalings s_* (ESTIMATED)
 are already in the parameter table and wired through the shipped
 observable/measurement tables — use them by name. Every other parameter
 is a symbol you introduce and estimate.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 172: Raimundez: known constants

```
KNOWN CONSTANTS you are given — experimental setup that the shipped
 measurement/observable/condition tables do NOT reveal. (1)
 PREEQUILIBRATION: every measurement preequilibrates to the unstimulated
 steady state before the experiment, so your model must RELAX to a basal
 state rather than hard-coding it as closed-form initial conditions. (2)
 TWO CELL LINES, ONE NETWORK: the condition table sets MKN1 and HS746T as
 0/1 cell-line switches, full_medium and hunger_medium as 0/1 medium
 switches, EGF_level (applied EGF), CET_level (cetuximab) and
 relative_ksyn_EGFR. The same network is fit to both cell lines in both
 media, so build ONE network and let those switches select the cell-line-
 and medium-specific parameter values — how a switch modifies a
 parameter is yours to decide. They are experimental inputs, not
 quantities to fit. Because EVERY measurement preequilibrates, the
 applied inputs (EGF, cetuximab) must be ABSENT during the
 pre-equilibration and present afterwards — otherwise the basal state you
 relax to is already stimulated. Gate them on the built-in symbol
 `time`, e.g. `EGF := EGF_level * piecewise(0, time < 0, 1)`. Write
 `time` itself: `t` is reserved by AMICI, and any other name (`tau`,
 `t_now`) is a free parameter the fit will estimate, which silently
 leaves the input on during pre-equilibration while the model still
 compiles. (3) The 79 per-experiment observable scalings s_* (ESTIMATED)
 are already in the parameter table and wired through the shipped
 observable/measurement tables — use them by name. Every other parameter
 is a symbol you introduce and estimate.
```

Listing 173: Raimundez: biocurator entities

```
- EGFR
- EGFR pTyr1068
- MET
- AKT1
- AKT pSer473
- MAPK1
- ERK1/2 pThr202/pTyr204
- PIK3CA
- KRAS
- EGF
- cetuximab
```

I.5.17 SCHWEN

Listing 174: Schwen: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
```

```
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

EXPERIMENTAL DESIGN you are given (already in the shipped tables —
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION MODEL: the per-experiment insulin-ELISA / IR-readout
 parameters (IR_obs_std, std, offset, scale, offset_nExpID1-4,
 scaleElisa_nExpID1-4, km_nExpID1-4, fragments) are already in the
 parameter table (estimated) and wired to the observables through the
 measurement observableParameters/noiseParameters columns; leave the
 observation model alone and just build the insulin-receptor reaction
 network.
- RECEPTOR INITIAL LEVELS: ini_R1 and ini_R2fold are estimated
 initial-condition parameters already in the parameter table. The
 low-affinity receptor starts at Rec1 = ini_R1; the high-affinity pool is
 a fold-multiple of it, Rec2 = ini_R1 * ini_R2fold (a compound initial
 condition — write it exactly as that product). Do NOT add new
 <species>_init parameters for the receptors.
- INSULIN DOSE: init_Ins is the per-condition insulin dose, an
 experimental INPUT set by the init_Ins condition column. Declare
 init_Ins as a PARAMETER (any default value) and use it as the initial
 amount of your free-insulin species (Ins = init_Ins), so the condition
 table sets the dose per condition. Do NOT estimate init_Ins — it is a
 fixed per-condition input, not a fitted parameter.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 175: Schwen: known constants

```
EXPERIMENTAL DESIGN you are given (already in the shipped tables —
 reference these by NAME and do NOT re-estimate or rename them):
- OBSERVATION MODEL: the per-experiment insulin-ELISA / IR-readout
 parameters (IR_obs_std, std, offset, scale, offset_nExpID1-4,
 scaleElisa_nExpID1-4, km_nExpID1-4, fragments) are already in the
 parameter table (estimated) and wired to the observables through the
 measurement observableParameters/noiseParameters columns; leave the
 observation model alone and just build the insulin-receptor reaction
 network.
- RECEPTOR INITIAL LEVELS: ini_R1 and ini_R2fold are estimated
 initial-condition parameters already in the parameter table. The
 low-affinity receptor starts at Rec1 = ini_R1; the high-affinity pool is
 a fold-multiple of it, Rec2 = ini_R1 * ini_R2fold (a compound initial
 condition — write it exactly as that product). Do NOT add new
 <species>_init parameters for the receptors.
- INSULIN DOSE: init_Ins is the per-condition insulin dose, an
 experimental INPUT set by the init_Ins condition column. Declare
 init_Ins as a PARAMETER (any default value) and use it as the initial
 amount of your free-insulin species (Ins = init_Ins), so the condition
 table sets the dose per condition. Do NOT estimate init_Ins — it is a
 fixed per-condition input, not a fitted parameter.
```

Listing 176: Schwen: biocurator entities

```
- insulin
- insulin receptor (INSR)
```

I.5.18 SNEYD

Listing 177: Sneyd: initial task

```
This PEtab problem ships WITHOUT a model — you have the measured data,
 the observables, and the conditions, but no reactions yet
 (describe_problem reports model_authored: false). Build a mechanistic
 ODE model that explains the data, working the loop above until you reach
 the lowest-bic model whose residuals show no systematic structure, then
 report the model you built and its fit.

SCORING: the run is scored on your BEST-bic fitted version
 AUTOMATICALLY, wherever HEAD ends up. Every fitted version is kept; you
 never need to roll back to, re-commit, or end on a good version — a
 worse attempt after it costs nothing.

READOUTS: a `readout_<name>` symbol in an observable formula is a
 MEASUREMENT MAPPING, not a species. The observable table states WHAT the
 assay measured (observableName); YOUR model says what that quantity is
 in terms of its own states, with one assignment rule per readout —
 `readout_x := <expression of your species>` in the model text, or the
 set_assignment op. It holds at all times (`:=`, never `=`, which would
 freeze it as a constant), it is not estimated, and it must NEVER appear
 in a reaction: a readout is a lens on your states, not a state — a
 species that is also assigned makes the model over-determined and the
 commit is refused. Think about what the assay physically sees: an
 antibody may count a phospho-protein both free and in complex; a channel
 readout may be a nonlinear function of state occupancies; a fraction
 needs the right denominator.

KNOWN CONSTANTS you are given. (1) The observation noise sigma is
 already in the parameter table (ESTIMATED) — reference it by NAME
 through the observable's noiseFormula
 (noiseParameter1_open_probability); it is fit from the data along with
 the gating constants. (2) IP3 and Ca are per-condition INPUTS set by the
 `IP3` and `Ca` condition columns (a fixed IP3 level and a fixed
 cytosolic Ca2+ level per experiment): name two constant-input SPECIES
 `IP3` and `Ca` in your model — species that no reaction produces or
 consumes, so each holds its per-condition value throughout — and use
 them in your rate laws. They are experimental inputs to reproduce, not
 values to fit.
MEASUREMENT MAPPINGS: each observable formula references a readout_*
 symbol your model must define with an assignment rule (`readout_x :=
 <expression of your species>`, or the set_assignment edit op).
 list_observables says what each readout measures biologically
 (observableName); deciding what that quantity IS in terms of your
 model's states is part of the modelling.
```

Listing 178: Sneyd: known constants

```
KNOWN CONSTANTS you are given. (1) The observation noise sigma is
 already in the parameter table (ESTIMATED) — reference it by NAME
 through the observable's noiseFormula
 (noiseParameter1_open_probability); it is fit from the data along with
 the gating constants. (2) IP3 and Ca are per-condition INPUTS set by the
 `IP3` and `Ca` condition columns (a fixed IP3 level and a fixed
 cytosolic Ca2+ level per experiment): name two constant-input SPECIES
 `IP3` and `Ca` in your model — species that no reaction produces or
 consumes, so each holds its per-condition value throughout — and use
```

```
them in your rate laws. They are experimental inputs to reproduce, not
values to fit.
```

Listing 179: Sneyd: biocurator entities

```
- IP3 receptor (ITPR)
- inositol 1,4,5-trisphosphate (IP3)
- calcium ion
```