AGENTIC SIMULATOR CONSTRUCTION WITH FALSIFICATION-GUIDED STRUCTURAL DIAGNOSIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models can turn natural-language briefs into runnable simulation code, yet automated simulator refinement remains vulnerable to unreliable structural diagnosis: the same macro-level mismatch may admit multiple plausible explanations, while an LLM-generated diagnosis can be acted upon before it is tested. We present `FASim`, an agentic framework for automated simulator construction with *falsification-guided structural diagnosis*. `FASim` constructs and calibrates an executable simulator, routes directly verifiable implementation issues to repair, and treats structurally ambiguous failures as competing falsifiable hypotheses. A Hypothesis Agent proposes alternative explanations, while a separate Falsification Agent derives their observable counterfactual signatures, freezes the test plan before execution, and challenges them through controlled simulator probes. Directly observable signatures are evaluated from execution traces, while stochastic or borderline responses use repeated matched-seed executions. Only sufficiently discriminated diagnoses are allowed to guide structural repair; falsified or inconclusive explanations are blocked from triggering persistent edits. Across user modeling, intervention-driven mask adoption, and mobility under ID and OOD shifts, `FASim` achieves strong multi-metric simulator fidelity. Ablation and process analyses further show that pre-repair falsification improves refinement quality, produces actionable structural diagnoses, and prevents unsupported explanations from being converted into edits. These results support *falsify before repair* as a practical principle for more reliable and auditable agentic simulator construction. Code and data: `https://anonymous.4open.science/status/FASim-8D87`.

1 INTRODUCTION

Simulations are essential for stress-testing decisions, comparing policy alternatives, and anticipating system-level outcomes in complex social settings (Bonabeau, 2002; Epstein, 2008; Farmer & Foley, 2009)—ranging from user behavior to public-health adoption and mobility under disruption (Lazer et al., 2009; Barbosa et al., 2018; Kucharski et al., 2020). Recent LLM-based and agentic systems increasingly translate natural-language briefs into runnable simulation code (Wang et al., 2023; Chen et al., 2024; Islam et al., 2025; Wang et al., 2025), but executability alone is insufficient: useful simulators must encode explicit micro-level states, interactions, and update rules whose executions reproduce observed macro-level behavior and remain meaningful under interventions and distribution shifts (Grimm et al., 2005; Sargent, 2010; Epstein, 2012; Mooney et al., 2025; Li et al., 2025; Huang et al., 2026).

The central challenge is unreliable structural diagnosis. Macro-level residuals are underdetermined: the same mismatch may be explained by a disconnected intervention pathway, an ineffective downstream dependency, or an incorrect state-transition rule. Parameter calibration cannot resolve this ambiguity because flexible parameters may partially compensate for misspecified structure while preserving competitive aggregate fit (Bergstra & Bengio, 2012; Gutmann & Corander, 2016). Conversely, directly acting on an LLM-generated "most plausible" explanation risks converting linguistic plausibility into an unsupported structural edit. We therefore ask a more specific question: *before an ambiguous structural diagnosis modifies the simulator, can its observable implications be tested on the current executable program?*

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.

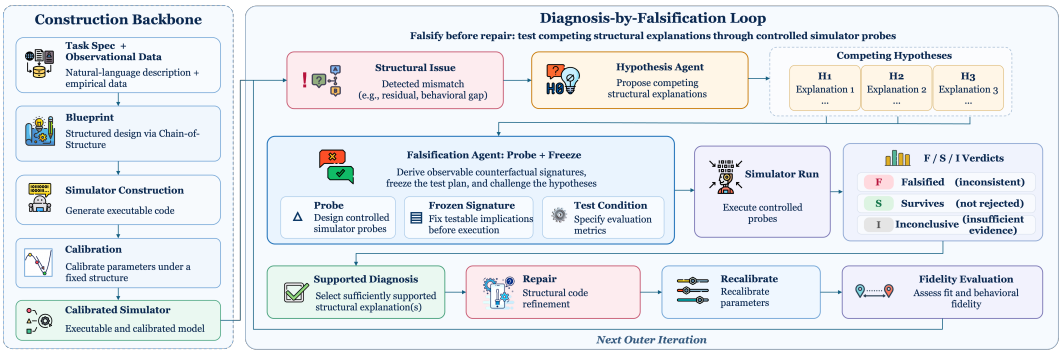


Figure 1: **FASim framework.** FASim separates directly verifiable issues from structurally ambiguous diagnoses, and follows a *falsify-before-repair* principle for the latter before structural refinement and recalibration.

Our answer is *falsify before repair*. Rather than committing to one explanation, FASim treats competing structural diagnoses as hypotheses with observable consequences under controlled execution. This formulation draws on automated falsification, where free-form hypotheses are translated into measurable implications (Huang et al., 2025), and on metamorphic testing, which evaluates expected relations across related executions when no exact oracle is available (Cañizares et al., 2026). Executable simulators make this particularly natural: a structural hypothesis predicts how internal states or outputs should respond to a controlled perturbation, allowing unsupported explanations to be rejected before repair.

Existing approaches address complementary parts of simulator construction but do not directly resolve this pre-repair diagnostic decision. Generate–run–revise systems (Wang et al., 2025) typically allow execution feedback or reflection to drive code revision, while G-SIM (Holt et al., 2025) searches over executable structures with gradient-free calibration. Diagnosis-driven simulator-code optimization localizes residuals into mechanism-level edits (Hua et al., 2025), and long-horizon evolutionary refinement separates parameter calibration from structural optimization while stabilizing repair through strategy memory and subsequent execution feedback (Hua et al., 2026). These methods improve how structural changes are proposed, localized, or retained, but do not require an ambiguous diagnosis to survive executable counterevidence before it is committed.

We propose FASim (**F**alsification-guided **A**gentic **Sim**ulator construction), an agentic framework for automated simulator construction with *falsification-guided structural diagnosis*. FASim constructs and calibrates an executable simulator, routes directly verifiable implementation issues to localized repair, and sends structurally ambiguous failures to a proposer–challenger diagnosis loop. A Hypothesis Agent generates competing explanations; a separate Falsification Agent derives and freezes their counterfactual signatures before controlled probes are executed. Hypotheses are labeled FALSIFIED, SURVIVES, or INCONCLUSIVE, and only sufficiently discriminated diagnoses can guide persistent structural refinement. This separates two questions that iterative refinement often conflates: whether a diagnosis is sufficiently supported to justify an edit, and whether the resulting simulator achieves better fidelity after repair.

Our main contributions are: (1) we formulate unreliable structural diagnosis as a central problem in automated simulator refinement and recast ambiguous residual-to-repair reasoning as falsifiable hypothesis selection; (2) we introduce a proposer–challenger framework that freezes testable implications before execution and gates structural repair using executable counterfactual evidence; and (3) across user modeling, intervention-driven mask adoption, and mobility tasks, we demonstrate strong simulator fidelity, while ablation and process analyses show that falsification both produces actionable diagnoses and blocks unsupported or inconclusive explanations from triggering edits.

2 RELATED WORK

Parameter calibration. Black-box optimization and simulation-based inference tune numerical parameters under fixed structure, but cannot resolve structural misspecification; flexible parameters may instead compensate for incorrect simulator logic (Dyer et al., 2024; Papamakarios & Murray,

2016; McCulloch et al., 2022). Surrogates can improve efficiency (Reiker et al., 2021; Lueckmann et al., 2019), but calibration remains complementary to structural diagnosis. **LLM-driven simulator construction.** Most LLM-based systems follow a *generate–run–revise* pattern (Piao et al., 2025; Yang et al., 2024; Wang et al., 2025; Yamada et al., 2025), where execution feedback or reflection directly guides code revision. This makes simulator construction increasingly automated, but plausible structural diagnoses are typically acted upon without being explicitly tested against executable counterevidence. **Structure search and diagnosis-driven refinement.** G-SIM (Holt et al., 2025) combines LLM-generated structures with gradient-free calibration. Diagnosis-driven code refinement (Hua et al., 2025) localizes residuals into mechanism-level edits, while long-horizon evolutionary optimization (Hua et al., 2026) stabilizes iterative repair through structural–parameter separation and strategy memory. In contrast, `FASim` focuses on the pre-repair diagnostic decision: ambiguous structural explanations must be tested before they can trigger persistent edits. **Falsification and metamorphic testing.** POPPER (Huang et al., 2025) translates free-form hypotheses into measurable implications and falsification experiments. Metamorphic testing checks expected relations across controlled executions, with recent work using LLMs to infer such relations and generate follow-up tests (Cañizares et al., 2026). `FASim` adapts these ideas to simulator refinement by challenging competing structural diagnoses through executable counterfactual probes before repair.

3 FASIM: FALSIFICATION-GUIDED SIMULATOR CONSTRUCTION

Problem formulation. We study automated construction of executable simulators whose micro-level rules reproduce observed macro-level behavior while remaining amenable to intervention and structural refinement. We represent a simulator as $C(\lambda, \omega, \phi)$, where λ denotes its structural program, ω its tunable numerical parameters, and ϕ its calibration schema. Let $\mathcal{A}(\cdot)$ aggregate micro-level executions into macro-level predictions, $\hat{y}_t(\lambda, \omega) = \mathcal{A}(\hat{x}_t(\lambda, \omega))$. Given observations $D = \{(y_t, u_t)\}_{t=1}^{T_{\text{steps}}}$, simulator fidelity is measured by

$$J(\lambda, \omega; D) = \sum_{(y_t, u_t) \in D} \mathcal{L}(y_t, \hat{y}_t(\lambda, \omega)) + \Psi(\lambda, \omega), \tag{1}$$

where $\mathcal{L}$ is the task-specific discrepancy and Ψ encodes task constraints.

`FASim` separates fixed-structure parameter calibration from structural diagnosis and refinement. For a fixed structure λ_t, a calibration backend estimates ω_t^* and produces execution evidence. However, calibration residuals do not uniquely identify their structural cause: the same macro-level mismatch may be compatible with multiple faults in the executable simulator. The central problem is therefore not merely how to optimize a simulator, but how to decide which structural explanation is sufficiently supported to justify an edit. Rather than allowing a plausible LLM-generated diagnosis to directly trigger repair, `FASim` treats competing structural explanations as falsifiable hypotheses and tests their observable implications before modifying the simulator.

3.1 CONSTRUCTION BACKBONE

Given observations D, task description T, and auxiliary data Q, the Data Analysis Agent constructs a unified schema $D_s = \text{BUILDDATASCHEMA}(D, T, Q)$, from which Chain-of-Structure reasoning produces a mechanistic blueprint $B = \mathcal{A}_{\text{DAA}}(D_s, T)$. The blueprint provides a structured scaffold for simulator construction, including the modeled entities, interaction and intervention pathways, evaluation interfaces, and calibratable quantities, but is not treated as an oracle for deciding whether a later structural diagnosis is correct.

The Code Generation Agent instantiates the initial simulator, $(\lambda_0, \phi_0, C_0) = \mathcal{A}_{\text{CGA}}(B, D_s)$. The initial calibration schema ϕ_0 specifies the exposed parameters, their ranges or priors, the task-specific calibration objective and budget, and a parameter-calibration backend from a predefined library such as Random Search, Bayesian Optimization (BO), or Simulation-Based Inference (SBI). These backends correspond to generic derivative-free search, sample-efficient black-box optimization, and likelihood-free parameter inference, respectively. The backend is fixed for each task, whereas the exposed parameterization in ϕ_t may change when structural refinement introduces, removes, or redefines calibratable quantities. The choice of calibration backend is orthogonal to our falsification-guided structural diagnosis.

For the current simulator C_t, the Simulation Execution Agent (SEA) calibrates the exposed numerical parameters according to ϕ_t:

$$(\omega_t^*, J_t, \delta_t) = \mathcal{A}_{\mathrm{SEA}}(C_t, D, \phi_t), \tag{2}$$

where J_t is the calibrated fidelity objective and δ_t denotes structured execution evidence used to expose simulator issues. Subsequent simulator structures are produced only through the diagnosis-gated refinement described below. Implementation details of blueprinting, code generation, calibration backends, and simulator execution are provided in Appendix A.

3.2 Falsification-Guided Structural Diagnosis

A residual does not identify its own cause. The same macro-level mismatch may arise from, for example, a disconnected intervention pathway, an ineffective downstream dependency, or an overly persistent state-transition rule. Conventional LLM-based refinement can turn one such plausible explanation directly into a structural edit, even when alternative explanations remain equally compatible with the observed residual. We instead formulate structurally ambiguous diagnosis as a falsifiable hypothesis-selection problem: competing explanations must make distinguishable predictions under controlled execution before any of them can guide repair. This follows the principle of *falsify before repair* and is inspired by automated falsification of free-form hypotheses (Huang et al., 2025), adapted here to executable simulator diagnosis.

Issue routing. Execution evidence may expose both directly verifiable implementation issues and structurally ambiguous failures. We partition the detected issues into $\mathcal{I}_t = \mathcal{I}_t^{\mathrm{dir}} \cup \mathcal{I}_t^{\mathrm{str}}$, where $\mathcal{I}_t^{\mathrm{dir}}$ contains issues whose cause can be established directly from executable evidence, such as a disconnected code path, invalid data flow, or violated execution/interface condition. Such issues are routed directly to localized repair. By contrast, $\mathcal{I}_t^{\mathrm{str}}$ contains failures that admit multiple plausible structural explanations and therefore require falsification-guided diagnosis. The remainder of this subsection focuses on these ambiguous structural issues.

3.2.1 Competing Structural Hypotheses

For each structurally ambiguous issue $I_{t,j} \in \mathcal{I}_t^{\mathrm{str}}$, the Hypothesis Agent proposes a set of competing explanations,

$$\mathcal{H}_{t,j} = \mathcal{A}_{\mathrm{H}}(I_{t,j}, \delta_t, B, C_t) = \left\{ h_{t,j}^{(1)}, \ldots, h_{t,j}^{(K_j)} \right\}. \tag{3}$$

The hypotheses in $\mathcal{H}_{t,j}$ explain the same observed issue but attribute it to different structural causes. Each hypothesis must admit an observable consequence under controlled execution; explanations that cannot be rendered falsifiable cannot justify structural repair. The Hypothesis Agent therefore acts only as a proposer and does not determine which explanation should be acted upon.

3.2.2 Frozen Falsification Plan

A separate *Falsification Agent* acts as a challenger. Given the competing hypotheses and the current executable simulator, it constructs

$$\Pi_{t,j} = \mathcal{A}_{\mathrm{F}}(\mathcal{H}_{t,j}, C_t) = \left(p_{t,j}, \left\{ s_{t,j}^{(k)}, f_{t,j}^{(k)} \right\}_{k=1}^{K_j}, \rho_{t,j} \right). \tag{4}$$

Here, $p_{t,j}$ denotes a counterfactual probe, which may comprise one or more related simulator executions designed jointly to discriminate among the hypotheses in $\mathcal{H}_{t,j}$; $s_{t,j}^{(k)}$ is the observable signature implied by $h_{t,j}^{(k)}$ under that probe; $f_{t,j}^{(k)}$ specifies an observation that would contradict the hypothesis; and $\rho_{t,j}$ specifies the evidence-collection regime. A signature need not predict an exact numeric outcome: it may instead specify a directional, invariant, ordering, propagation, locality, or trace-level consequence.

Freeze-then-evaluate. Crucially, $\Pi_{t,j}$ is fixed *before* observing the probe outcome. The probe, predicted signatures, and falsification conditions cannot be revised after execution. This prevents the criterion for supporting a diagnosis from being adapted post hoc to the observed response.

3.2.3 EXECUTABLE FALSIFICATION

The frozen probe is executed on the current calibrated simulator. Probe execution is deterministic under the frozen plan, with simulator structure and calibrated parameters fixed except for the declared intervention. Let $R(p; C_t, \omega_t^*, \xi)$ denote the observable response under simulator randomness ξ. When p comprises multiple related executions, $R(p; \cdot)$ denotes their joint recorded response.

We use two evidence regimes. For signatures that are directly observable from controlled execution, such as pathway activation, signal propagation, or branch invocation, the response is evaluated from execution traces. For stochastic or borderline signatures, we use repeated matched-seed executions:

$$\mathcal{R}_{t,j} = \begin{cases} R(p_{t,j}; C_t, \omega_t^*, \xi_0), & \rho_{t,j} = \textsc{Structural}, \\ \{(R(p_0; C_t, \omega_t^*, \xi_n), R(p_{t,j}; C_t, \omega_t^*, \xi_n))\}_{n=1}^{N}, & \rho_{t,j} = \textsc{Matched}. \end{cases} \tag{5}$$

For matched evaluation, p_0 denotes the corresponding unperturbed execution of the same simulator. Using the same seeds for baseline and probe conditions reduces confounding from ordinary simulator randomness. Importantly, neither evidence regime infers structural validity from the aggregate fidelity objective J_t. Probe execution reuses `FASim`'s simulator execution and instrumentation infrastructure; implementation details are provided in Appendix A.6.

3.2.4 FALSIFIED, SURVIVING, OR INCONCLUSIVE

Each hypothesis receives a verdict

$$v_{t,j}^{(k)} = \Gamma\left(s_{t,j}^{(k)}, f_{t,j}^{(k)}, \mathcal{R}_{t,j}\right) \in \{\mathsf{F}, \mathsf{S}, \mathsf{I}\}, \tag{6}$$

where F, S, and I denote FALSIFIED, SURVIVES, and INCONCLUSIVE, respectively. Whenever the frozen falsification condition can be evaluated directly, Γ computes the verdict deterministically from the recorded response. LLM reasoning is used only when a frozen signature requires structured interpretation of execution traces; in either case, the signature and falsification condition remain fixed after the outcome is observed.

A hypothesis SURVIVES when the execution evidence is consistent with its pre-specified implication, but survival does not establish that the hypothesis is the true real-world causal mechanism. We form an actionable, provisionally supported structural diagnosis only when one hypothesis uniquely survives and all competing explanations are falsified:

$$d_{t,j} = \begin{cases} h_{t,j}^{(k)}, & v_{t,j}^{(k)} = \mathsf{S} \ \wedge \ v_{t,j}^{(\ell)} = \mathsf{F}, \quad \forall \ell \neq k, \\ \varnothing, & \text{otherwise.} \end{cases} \tag{7}$$

Thus, a falsified hypothesis is rejected, while an inconclusive hypothesis is withheld from repair because the available executable evidence is insufficient. A structural issue therefore triggers repair only when the falsification procedure sufficiently discriminates among its competing explanations.

3.3 DIAGNOSIS-GATED STRUCTURAL REFINEMENT

The refinement stage combines two sources of actionable evidence. Let $\mathcal{D}_t^{\mathrm{dir}}$ denote directly verified implementation issues routed from $\mathcal{I}_t^{\mathrm{dir}}$, and let $\mathcal{D}_t^{\mathrm{fal}} = \{d_{t,j} \mid d_{t,j} \neq \varnothing\}$ denote the provisionally supported structural diagnoses produced by falsification. The complete repair set is $\mathcal{D}_t = \mathcal{D}_t^{\mathrm{dir}} \cup \mathcal{D}_t^{\mathrm{fal}}$. Only $\mathcal{D}_t^{\mathrm{fal}}$ requires the falsification gate above; direct issues have already been established through executable evidence.

Using the current simulator, the actionable repair set, and the associated execution evidence, the Code Generation Agent produces a minimally modified candidate simulator,

$$(\tilde{\lambda}_{t+1}, \tilde{\phi}_{t+1}, \tilde{C}_{t+1}) = \mathcal{A}_{\mathrm{CGA}}(B, C_t, \mathcal{D}_t, \delta_t). \tag{8}$$

The candidate is then recalibrated using the same task-level calibration backend,

$$(\tilde{\omega}_{t+1}^*, \tilde{J}_{t+1}, \tilde{\delta}_{t+1}) = \mathcal{A}_{\mathrm{SEA}}\left(\tilde{C}_{t+1}, D, \tilde{\phi}_{t+1}\right). \tag{9}$$

Falsification and fidelity serve different roles. Falsification determines whether an ambiguous structural explanation is sufficiently supported to justify repair, whereas J evaluates the predictive

quality of the simulator produced after repair and recalibration. A decrease in J is therefore evidence of improved simulator fidelity, but is not itself treated as evidence that the preceding structural hypothesis was correct.

The candidate is retained only if its recalibrated fidelity improves upon the current simulator:

$$(C_{t+1}, J_{t+1}) = \begin{cases} (\tilde{C}_{t+1}, \tilde{J}_{t+1}), & \tilde{J}_{t+1} < J_t - \epsilon_{\text{imp}}, \\ (C_t, J_t), & \text{otherwise.} \end{cases} \tag{10}$$

The corresponding structure, calibration state, and execution evidence are carried forward with the retained simulator.

The retained C_{t+1} becomes the starting point of the next outer iteration. Issue-, hypothesis-, probe-, and verdict-level states are discarded and reconstructed from the new execution evidence; optimization state therefore persists while diagnostic state is reset. The loop terminates after K consecutive non-improving iterations or when $T_{\max}$ is reached. The final retained simulator $C(\lambda^\star, \omega^\star, \phi^\star)$ is returned under the fixed evaluation protocol.

Overall, FASim follows a routed refinement process. After calibration and execution, directly verifiable implementation issues are sent to localized repair, whereas structurally ambiguous issues enter the hypothesis–falsification pipeline and can guide repair only when the falsification gate yields a supported diagnosis. The resulting candidate simulator is then recalibrated and evaluated before the next outer iteration. Thus, FASim preserves an efficient direct path for executable implementation failures while preventing plausible but ambiguous structural explanations from triggering persistent edits without counterfactual support.

4 FASIM: EMPIRICAL STUDY

4.1 EXPERIMENT SETTINGS

Benchmarks. We evaluate FASim on three real-world–inspired simulation tasks following a common *micro-to-macro* evaluation paradigm: the simulator models micro-entities through explicit states, interactions, and decision rules, while its executions are evaluated against observed macro-level behavior. We consider: (1) *User Modeling* (AgentSociety Challenge (Yan et al., 2025)), where the simulator instantiates users and items with user-level preference states and interaction rules, and predicts *1,200* held-out user–item star ratings from historical purchase logs; (2) *Mask Adoption Behavior Simulation*, which models the time-evolving mask-wearing decisions of *1,000* individuals embedded in a multi-relational social graph under heterogeneous ties, risk perception, and an intervention campaign beginning on Day 10. Following data paradigms in BESSIE (Mortveit et al., 2022) and prior mask-decision simulators (Mitsopoulos et al., 2023), we construct manually perturbed, LLM-assisted synthetic trajectories for controlled evaluation; and (3) *Personal Mobility Generation* (LLMob (Wang et al., 2024a)), tracking *69* modeled residents over *124.10* active days on average and evaluating next-day trajectory generation under normal→normal (ID), abnormal→abnormal (pandemic), and normal→abnormal (OOD) settings.

Evaluation metrics. We report the mean and 95% confidence intervals over five random seeds (Colas et al., 2018). All metrics are computed from task-specific observable evidence generated by the executable simulators. For *User Modeling*, we report mean absolute error (*MAE*) on held-out star ratings (Wang et al., 2024b). For *Mask Adoption*, we report adoption-rate RMSE and *Transition Fit Error* (TFE). RMSE measures the discrepancy between simulated and observed daily aggregate mask-adoption rates, whereas TFE measures how well the simulator reproduces individual-level mask-state transition dynamics. Together, they evaluate both aggregate adoption fidelity and behavioral switching/persistence patterns. Lower values are better for both metrics. For *Personal Mobility*, we report distributional fidelity using Jensen–Shannon divergence (*JSD*) for arrival-time distributions and Wasserstein distance (*WD*) for trip-length distributions. Lower values indicate better performance for all reported metrics.

Baseline methods. We compare FASim with both general-purpose agentic systems and simulation-specific construction methods under matched task evidence and evaluation protocols. *AI Scientist-v2* (Yamada et al., 2025) provides an automated research-code-generation and execution workflow, while *YuLan-OneSim* (Wang et al., 2025) uses ODD-style scenario specifications (Grimm et al., 2010)

Table 1: Evaluation results (±denotes 95% CIs) on three simulation tasks, and lower values indicate better performance. The best and second-best results are highlighted in **bold** and underlined. **Traj. N→N**, **Traj. A→A**, and **Traj. N→A** denote **Personal Mobility** (training on normal period data and predicting normal period trajectory), abnormal-to-abnormal (pandemic) prediction, and abnormal prediction using only normal-period history, respectively.

Methods ↓	User.	Mask.		Traj. N→N (ID)		Traj. A→A (ID)		Traj. N→A (OOD)	
	MAE ↓	RMSE ↓	TFE ↓	JSD ↓	WD ↓	JSD ↓	WD ↓	JSD ↓	WD ↓
SRR	0.70±0.010	0.26±0.009	0.21±0.007	0.53±0.005	0.71±0.006	0.28±0.006	0.72±0.006	0.13±0.007	1.03±0.006
AI Scientist-v2	0.63±0.010	**0.13±0.007**	**0.08±0.003**	0.09±0.001	3.12±0.006	0.04±0.001	4.18±0.006	0.07±0.007	3.98±0.006
Yulan-OneSim	0.83±0.009	0.23±0.006	0.17±0.004	0.19±0.004	**0.15±0.002**	0.16±0.005	**0.23±0.005**	0.18±0.006	**0.29±0.006**
G-SIM-ES	0.71±0.007	0.43±0.008	0.41±0.005	0.55±0.002	0.23±0.003	0.27±0.004	0.52±0.004	0.53±0.005	0.73±0.005
G-SIM-SBI	0.70±0.008	0.20±0.004	0.14±0.005	0.63±0.003	1.64±0.003	**0.02±0.003**	2.15±0.003	**0.04±0.005**	1.32±0.005
FASim	**0.10±0.004**	0.17±0.007	0.14±0.006	**0.05±0.003**	0.50±0.003	0.04±0.003	0.71±0.003	**0.04±0.005**	0.86±0.005

with iterative verify–repair. *G-SIM* (Holt et al., 2025) combines LLM-generated simulator structures with gradient-free calibration and iterative program refinement. We additionally introduce *Structured Reflective Repair (SRR)* as the closest diagnosis-focused baseline for isolating executable falsification. SRR receives the same task description, persisted blueprint, simulator code, and execution evidence as FASim, but resolves competing structural explanations through structured LLM reflection rather than frozen counterfactual predictions and executable probes, and directly applies the selected diagnosis to repair. It therefore controls for blueprint access, code context, execution evidence, and iterative LLM reasoning while removing the central *falsify-before-repair* mechanism. Baseline-specific reproduction, adaptation boundaries, and matched-budget details are reported in Appendix C.

Implementation. Parameter calibration is treated as fixed-structure infrastructure rather than as a competing methodological component. Each task uses a fixed backend from the predefined calibration library, and comparable methods share the same task-level calibration budget whenever applicable; structural refinement may update the exposed parameterization but does not switch the backend across iterations. Backend implementations and configurations are described in Appendix A.4. Unless otherwise stated, FASim and reproduced LLM-based baselines use GPT-5.2 (OpenAI, 2025) as the common backbone. Compute accounting, calibration and execution budgets, LLM/simulator-call limits, wall-clock/token usage, and final evaluation details are reported in Appendix C.

4.2 Main Findings

Strong end-to-end fidelity across heterogeneous tasks. Table 1 compares the final simulators produced by FASim and the baselines across user modeling, intervention-driven mask adoption, and personal mobility under both ID and OOD regimes. FASim achieves the best *User Modeling* MAE (0.10 ± 0.004) and remains strong on both *Mask Adoption* metrics, ranking second in RMSE (0.17 ± 0.007) and tying for second-best TFE (0.14 ± 0.006). The latter is important because RMSE measures aggregate adoption fidelity whereas TFE evaluates individual-level switching and persistence dynamics. Mobility results further reveal a multi-metric trade-off rather than a single-metric winner: FASim achieves the best *Traj. N→N* JSD and ties the best *Traj. N→A* JSD, while avoiding the very large WD exhibited by several low-JSD baselines. Conversely, YuLan-OneSim obtains the lowest WD across the mobility regimes but with noticeably weaker JSD. Overall, FASim provides a strong joint balance across temporal and spatial distributional fidelity rather than optimizing one statistic at the expense of another.

Executable falsification improves over reflection and generic structure search. SRR provides the closest diagnosis-focused comparison: it receives the same task description, blueprint, simulator code, and execution evidence as FASim, but resolves competing explanations through structured LLM reflection rather than executable falsification before repair. FASim consistently outperforms SRR across all reported tasks, including substantially lower User MAE and Mask RMSE/TFE, as well as stronger mobility fidelity. This comparison supports the value of challenging plausible structural explanations with executable counterfactual evidence rather than selecting a diagnosis through reflection alone. The G-SIM variants show a complementary limitation: they can achieve strong individual mobility statistics, but their paired JSD–WD performance is often highly uneven. For example, G-SIM-SBI obtains the best *Traj. A→A* JSD (0.02 ± 0.003) but a much larger WD (2.15 ± 0.003), and under *Traj. N→A* it ties FASim on JSD while yielding a higher WD (1.32 ± 0.005 versus 0.86 ± 0.005). These results suggest that the advantage of FASim is not merely iterative structure modification, but diagnosis-guided refinement in which ambiguous explanations must be tested before they can trigger persistent edits.

Baselines expose complementary strengths but less consistent cross-metric behavior. AI Scientist-v2 performs particularly well on Mask Adoption and several mobility JSD metrics, but its mobility WD is substantially larger, while YuLan-OneSim exhibits the opposite pattern with strong WD but weaker JSD and poorer User/Mask performance. Thus, the baselines capture different components of simulator construction—general-purpose experimentation, scenario structuring, or iterative structure search—whereas `FASim` maintains strong cross-task and multi-metric fidelity under a unified falsification-guided refinement process. Table 1 evaluates the quality of the final simulators and should not by itself be interpreted as evidence that a particular structural diagnosis is correct; the role of executable falsification is examined separately through the process analysis and ablations.

4.3 Falsification Process Analysis

Table 2 summarizes the behavior of the falsification stage over six refinement iterations. *Gate* counts diagnoses that pass the strict falsification gate; *Blocked* denotes adjudicated hypotheses that are either Falsified or Inconclusive and therefore cannot justify structural repair; *Action.* is the fraction of falsification-routed structural issue-instances that produce a gate-passed diagnosis; and *Yield* reports supported diagnoses with a comparable subsequent iteration whose next candidate is retained as a new best simulator. Across the five tasks, 71 hypotheses receive final verdicts: 45.1% are Falsified, 39.4% Survive, and 15.5% remain Inconclusive. Consequently, 60.6% are blocked from serving as structural-repair evidence. At the issue level, 26 of 93 falsification-routed structural issue-instances produce a gate-passed diagnosis, and four of five tasks yield at least one such diagnosis. This selectivity is intentional: ambiguous residuals are not forced into repair when executable evidence is insufficient, while directly verifiable issues can still proceed through the separate direct-repair route described in § 3.2.

Table 2: Falsification behavior over refinement iterations.

Task	Gate	Blocked	Action.	Yield
User.	0	100.0%	0.0%	–
Mask.	5	66.7%	35.7%	2/2
Traj. N→N	6	50.0%	24.0%	3/5
Traj. A→A	9	59.1%	42.9%	2/6
Traj. N→A	6	55.6%	27.3%	3/4
Overall	**26**	**60.6%**	**28.0%**	**10/17**

Falsification can turn ambiguous residuals into actionable diagnoses. The clearest example occurs in *Traj. N→A*, where a large trip-distance residual gives rise to competing explanations of the distance mechanism. A matched-seed counterfactual probe separates these alternatives: one hypothesis Survives, while its competitor is Falsified. The supported diagnosis then guides a targeted refinement of the implicated mechanism. In the following comparable refinement trajectory, trip-distance WD decreases by 71.6% and the overall objective improves by 15.4%. This provides a complete *residual* → *competing hypotheses* → *probe* → *diagnosis* → *repair* chain in which executable falsification constrains the structural edit before it is committed.

Gate-passed diagnoses are often followed by useful repairs, while inconclusive evidence is withheld. Among supported diagnoses that occur early enough to have a comparable subsequent refinement iteration, 10 of 17 (58.8%) are associated with a candidate retained as a new best simulator. We treat this as an *associated repair yield*, rather than a strict causal success rate, because a refinement step may jointly consume gate-passed structural diagnoses and directly verified issues. The complementary behavior appears in *User.*: all hypotheses reaching final adjudication are Inconclusive, so no falsification-supported structural repair is produced. This illustrates the intended conservative role of the gate—uncertainty is withheld from persistent structural modification rather than converted into an edit.

Overall, the process analysis shows that executable falsification serves two complementary roles: it produces actionable diagnoses when competing explanations can be discriminated, and withholds repair when they cannot. Together with the associated repair yield, these results support falsification as both a *diagnosis producer* and a *repair gate*, rather than a post-hoc justification mechanism. We provide three complementary analyses in the appendix. § D.1 traces an end-to-end falsification-supported repair within a formal refinement trajectory. § D.2 evaluates structural recovery under deliberately injected defects for which the corrupted mechanism and reference structure are known, showing that `FASim` can recover and stably retain the intended structure rather than merely optimize aggregate fit. § D.3 further evaluates diagnosis localization, mechanism linkage, and post-edit effectiveness relative to structured reflection under method-blinded assessment.

4.4 Ablation Study

We ablate the principal design choices underlying FASim's falsification-guided structural diagnosis while keeping the construction backbone, task-specific calibration backend, LLM backbone, evaluation protocol, and outer-loop budget fixed unless explicitly modified. We consider *fix-λ*, which disables structural refinement and performs only parameter calibration; *EVO-style*, which lets a diagnosis trigger repair directly and validates the resulting edit only afterwards through downstream execution and fidelity improvement, following the post-hoc repair-validation paradigm of prior evolutionary simulator optimization (Hua et al., 2026); *Single-H*, which tests only one proposed structural explanation rather than competing hypotheses; *No-Freeze*, which removes pre-commitment to predicted signatures and falsification conditions before probe execution; and *Single-Run*, which replaces repeated matched-seed evidence with a single execution. Table 3 reports performance degradation relative to the full FASim, with larger positive Δ indicating a larger loss.

Table 3: We conducted an ablation study using FASim as the baseline while larger positive Δ values indicate greater performance degradation.

Models $\downarrow$	**User.**	**Mask.**	**Mob. N$\rightarrow$A**
	ΔMAE $\downarrow$	ΔRMSE $\downarrow$	ΔWD $\downarrow$
FASim	-	-	-
fix-λ	+0.33$\pm$0.003	+0.43$\pm$0.005	+0.36$\pm$0.003
EVO-style	+0.23$\pm$0.002	+0.30$\pm$0.004	+0.26$\pm$0.006
Single-H	+0.18$\pm$0.004	+0.33$\pm$0.005	+0.22$\pm$0.007
No-Freeze	+0.15$\pm$0.010	+0.29$\pm$0.004	+0.25$\pm$0.007
Single-Run	+0.09$\pm$0.006	+0.14$\pm$0.004	+0.12$\pm$0.002

Structural refinement alone is insufficient. The largest degradation comes from *fix-λ*, confirming that parameter calibration cannot reliably compensate for structural misspecification. However, *EVO-style* also degrades substantially across all three tasks, showing that simply allowing structural edits is not enough. In this variant, a plausible diagnosis can modify the simulator before being tested, and its validity is judged only indirectly through the fidelity of the repaired program. The gap to full FASim therefore supports the distinction between *repair-then-validate* and *falsify-before-repair*: testing the diagnosis itself before committing the edit reduces unsupported structural changes.

The internal design of falsification also matters. *Single-H* consistently degrades performance because a probe can only check the initially preferred explanation rather than discriminate among plausible alternatives, increasing the risk of premature commitment. *No-Freeze* performs similarly worse because allowing testable implications to be interpreted together with the observed outcome weakens the evidential constraint and permits post-hoc reinterpretation. *Single-Run* causes the smallest but still consistent degradation, indicating that individual probes can be informative, but repeated matched-seed comparisons improve reliability when responses are stochastic or borderline by separating probe-induced effects from ordinary simulator randomness.

Overall, the ablations show that structural refinement is necessary, but reliable refinement also depends on competing hypotheses, frozen testable implications, and noise-controlled executable evidence. These results support the central design of FASim: ambiguous structural diagnoses should be tested before they are allowed to modify the simulator. § D.4 further shows that the benefits of falsification-guided refinement are not specific to GPT-5.2 and can transfer to other capable LLM backbones, while remaining sensitive to backbone reasoning quality.

5 Conclusion

We present FASim, an agentic framework for automated simulator construction with *falsification-guided structural diagnosis*. FASim addresses a key weakness of LLM-driven refinement: macro-level residuals may admit multiple plausible structural explanations, yet these explanations are often acted upon before they are tested. To avoid unsupported edits, FASim routes directly verifiable implementation issues to repair while treating ambiguous structural failures as competing falsifiable hypotheses that must survive executable counterfactual probes before modification. Across three benchmark families, including intervention and OOD settings, FASim achieves strong simulator fidelity. Ablation and process analyses further show that pre-repair falsification improves refinement quality while also blocking unsupported or inconclusive diagnoses from triggering edits. These results support *falsify before repair* as a practical principle for more reliable and auditable agentic simulator construction, without assuming recovery of a unique real-world causal mechanism.

AI USE STATEMENT

In this work, we used generative AI tools to assist with proofreading and correcting grammatical errors in the manuscript, as well as organizing and formatting tabular data. All numerical values and reported results were manually checked by the authors against the original experimental outputs to ensure accuracy and to prevent hallucinated or fabricated values. We did not use generative AI tools to synthesize experimental data. We also used generative AI tools to assist with generating portions of the engineering code used in the project. All AI-generated code was manually reviewed by the authors and tested before being incorporated into the experimental pipeline. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, code, tables, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The experiments reported in this paper do not involve human-in-the-loop intervention, crowdsourcing, or human-subject data collection as part of the optimization, diagnosis, repair, or evaluation pipeline. The benchmark datasets and released artifacts used in the reported experiments are synthetic, publicly available, or derived from the task settings described in the paper, and the released materials do not contain personally identifiable information.

Because `FASim` is designed for automated social-simulator construction, its outputs could be misused if treated as ground-truth predictions or deployed without appropriate validation, particularly in domains such as public health, mobility planning, or behavior modeling. Potential risks include overconfident policy guidance, surveillance or behavioral manipulation, and amplification of biases present in the underlying data or task specification. We therefore emphasize that the generated simulators should be interpreted as model-based hypotheses rather than authoritative representations of real-world causal processes, and we recommend transparent reporting of data provenance, uncertainty, model limitations, and external domain review before any high-stakes deployment.

The broader project includes optional human-feedback and expert-inspection procedures for system-development and responsible-use purposes; these were not used to alter the experimental blueprints, structural diagnoses, simulator code, calibration decisions, or benchmark results reported in this paper. Any such auxiliary human-facing activities are conducted under appropriate institutional ethics oversight and privacy protections. The authors take responsibility for the ethical conduct, documentation, and reporting of the work.

REPRODUCIBILITY STATEMENT

We provide detailed implementation and reproduction information in the appendix. Appendix A describes the construction, calibration, execution, and falsification infrastructure; Appendix B specifies the complete refinement workflow; and Appendix C reports baseline-specific reproduction protocols, calibration and execution budgets, random seeds, resource accounting, and final evaluation procedures. Prompt templates and agent interfaces are also included in the supplementary material. The anonymous code repository provided in the abstract section contains the implementation, configurations, and evaluation scripts used for the reported experiments.

REFERENCES

Hugo Barbosa, Marc Barthelemy, Gourab Ghoshal, Charlotte R James, Maxime Lenormand, Thomas Louail, Ronaldo Menezes, José J Ramasco, Filippo Simini, and Marcello Tomasini. Human mobility: Models and applications. *Physics Reports*, 734:1–74, 2018.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The journal of machine learning research*, 13(1):281–305, 2012.

Jan Boelts, Philipp Harth, Richard Gao, Daniel Udvary, Felipe Yanez, Daniel Baum, Hans-Christian Hege, Marcel Oberlaender, and Jakob H Macke. Simulation-based inference for efficient identification of generative models in computational connectomics. *PLOS Computational Biology*, 19(9): e1011406, 2023.

Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, 99(suppl_3):7280–7287, 2002.

Pablo C Cañizares, Pablo Gómez-Abajo, Esther Guerra, and Juan de Lara. Towards metamorphic testing with llm-based workflows: Metamorphic relation inference and follow-up test case generation. *Information and Software Technology*, pp. 108150, 2026.

John Chen, Xi Lu, Yuzhou Du, Michael Rejtig, Ruth Bagley, Mike Horn, and Uri Wilensky. Learning agent-based modeling with llm companions: Experiences of novices and experts using chatgpt & netlogo chat. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–18, 2024.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.

Michael Deistler, Jan Boelts, Peter Steinbach, Guy Moss, Thomas Moreau, Manuel Gloeckler, Pedro LC Rodrigues, Julia Linhart, Janne K Lappalainen, Benjamin Kurt Miller, et al. Simulation-based inference: A practical guide. *arXiv preprint arXiv:2508.12939*, 2025.

Joel Dyer, Patrick Cannon, J Doyne Farmer, and Sebastian M Schmon. Black-box bayesian inference for agent-based models. *Journal of Economic Dynamics and Control*, 161:104827, 2024.

Joshua M Epstein. Why model? *Journal of artificial societies and social simulation*, 11(4):12, 2008.

Joshua M Epstein. Generative social science: Studies in agent-based computational modeling. In *Generative Social Science*. Princeton University Press, 2012.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *Advances in neural information processing systems*, 32, 2019.

J Doyne Farmer and Duncan Foley. The economy needs agent-based modelling. *Nature*, 460(7256): 685–686, 2009.

Volker Grimm, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M Mooij, Steven F Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald L DeAngelis. Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *science*, 310(5750):987–991, 2005.

Volker Grimm, Uta Berger, Donald L DeAngelis, J Gary Polhill, Jarl Giske, and Steven F Railsback. The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768, 2010.

Volker Grimm, Steven F Railsback, Christian E Vincenot, Uta Berger, Cara Gallagher, Donald L DeAngelis, Bruce Edmonds, Jiaqi Ge, Jarl Giske, Juergen Groeneveld, et al. The odd protocol for describing agent-based and other simulation models: A second update to improve clarity, replication, and structural realism. *Journal of Artificial Societies and Social Simulation*, 23(2), 2020.

Michael U Gutmann and Jukka Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016.

Samuel Holt, Max Ruiz Luyten, Antonin Berthon, and Mihaela van der Schaar. G-sim: Generative simulations with large language models and gradient-free calibration. *CoRR*, abs/2506.09272, 2025. doi: 10.48550/ARXIV.2506.09272. URL https://doi.org/10.48550/arXiv.2506.09272.

Yuncheng Hua, Sion Weatherhead, Mehdi Jafari, Hao Xue, and Flora D Salim. SOCIA-∇: Textual gradient meets multi-agent orchestration for automated spatially grounded simulator generation. *arXiv preprint arXiv:2505.12006*, 2025.

Yuncheng Hua, Sion Weatherhead, Mehdi Jafari, Hao Xue, and Flora D Salim. Socia-evo: Automated simulator construction via dual-anchored bi-level optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27596–27634, 2026.

Kaibo Huang, Jin Tan, Yukun Wei, Wanling Li, Zipei Zhang, Hui Tian, Zhongliang Yang, and Linna Zhou. Agentmark: Utility-preserving behavioral watermarking for agents. *arXiv preprint arXiv:2601.03294*, 2026.

Kexin Huang, Ying Jin, Ryan Li, Michael Y. Li, Emmanuel J. Candès, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/huang25n.html.

Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Codesim: Multi-agent code generation and problem solving through simulation-driven planning and debugging. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5113–5139, 2025.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Adam J Kucharski, Petra Klepac, Andrew JK Conlan, Stephen M Kissler, Maria L Tang, Hannah Fry, Julia R Gog, W John Edmunds, Jon C Emery, Graham Medley, et al. Effectiveness of isolation, testing, contact tracing, and physical distancing on reducing transmission of sars-cov-2 in different settings: a mathematical modelling study. *The Lancet infectious diseases*, 20(10):1151–1160, 2020.

Ahmed Laatabi, Nicolas Marilleau, Tri Nguyen-Huu, Hassan Hbid, and Mohamed Ait Babram. Odd+ 2d: An odd based protocol for mapping data to empirical abms. *Journal of Artificial Societies and Social Simulation*, 21(2), 2018.

David Lazer, Alex Pentland, Lada Adamic, Sinan Aral, Albert-László Barabási, Devon Brewer, Nicholas Christakis, Noshir Contractor, James Fowler, Myron Gutmann, et al. Computational social science. *Science*, 323(5915):721–723, 2009.

Chance Jiajie Li, Jiayi Wu, Zhenze Mo, Ao Qu, Yuhan Tang, Kaiya Ivy Zhao, Yulu Gan, Jie Fan, Jiangbo Yu, Jinhua Zhao, et al. Simulating society requires simulating thought. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems Position Paper Track*, 2025.

Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pp. 32–53. PMLR, 2019.

Josie McCulloch, Jiaqi Ge, Jonathan A Ward, Alison Heppenstall, J Gareth Polhill, and Nick Malleson. Calibrating agent-based models using uncertainty quantification methods. *Journal of Artificial Societies and Social Simulation*, 25(2), 2022.

Konstantinos Mitsopoulos, Lawrence Baker, Christian Lebiere, Peter Pirolli, Mark Orr, and Raffaele Vardavas. Masking behaviors in epidemiological networks with cognitively-plausible reinforcement learning. *arXiv preprint arXiv:2312.03301*, 2023.

James Mooney, Josef Woldense, Zheng Robert Jia, Shirley Anugrah Hayati, My Ha Nguyen, Vipul Raheja, and Dongyeop Kang. Are llm agents behaviorally coherent? latent profiles for social simulation. *arXiv preprint arXiv:2509.03736*, 2025.

Henning S. Mortveit, Stephen C. Adams, Faraz Dadgostari, Samarth Swarup, and Peter A. Beling. BESSIE: A behavior and epidemic simulator for use with synthetic populations. *CoRR*, abs/2203.11414, 2022. doi: 10.48550/ARXIV.2203.11414. URL https://doi.org/10.48550/arXiv.2203.11414.

Birgit Müller, Friedrich Bohn, Gunnar Dreßler, Jürgen Groeneveld, Christian Klassert, Romina Martin, Maja Schlüter, Jule Schulze, Hanna Weise, and Nina Schwarz. Describing human decisions in agent-based models–odd+ d, an extension of the odd protocol. *Environmental Modelling & Software*, 48:37–48, 2013.

OpenAI. Introducing gpt-5. 2025.

George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

Jinghua Piao, Yuwei Yan, Jun Zhang, Nian Li, Junbo Yan, Xiaochong Lan, Zhihong Lu, Zhiheng Zheng, Jing Yi Wang, Di Zhou, et al. Agentsociety: Large-scale simulation of llm-driven generative agents advances understanding of human behaviors and society. *arXiv preprint arXiv:2502.08691*, 2025.

Simon P Preston, Richard D Wilkinson, Richard H Clayton, Mike J Chappell, and Gary R Mirams. Think before you fit: parameter identifiability, sensitivity and uncertaintyin systems biology models. *Current Opinion in Systems Biology*, pp. 100563, 2025.

Theresa Reiker, Monica Golumbeanu, Andrew Shattock, Lydia Burgert, Thomas A Smith, Sarah Filippi, Ewan Cameron, and Melissa A Penny. Emulator-based bayesian optimization for efficient multi-objective calibration of an individual-based model of malaria. *Nature communications*, 12 (1):7212, 2021.

Robert G Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 winter simulation conference*, pp. 166–183. IEEE, 2010.

Jiawei Wang, Renhe Jiang, Chuang Yang, Zengqing Wu, Makoto Onizuka, Ryosuke Shibasaki, Noboru Koshizuka, and Chuan Xiao. Large language models as urban residents: An llm agent framework for personal mobility generation. *Advances in Neural Information Processing Systems*, 37:124547–124574, 2024a.

Lei Wang, Heyang Gao, Xiaohe Bo, Xu Chen, and Ji-Rong Wen. Yulan-onesim: Towards the next generation of social simulator with large language models. *arXiv preprint arXiv:2505.07581*, 2025.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. Gensim: Generating robotic simulation tasks via large language models. *arXiv preprint arXiv:2310.01361*, 2023.

Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingzhen Yang. Recmind: Large language model powered agent for recommendation. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pp. 4351–4364. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.FINDINGS-NAACL.271. URL https://doi.org/10.18653/v1/2024.findings-naacl.271.

Yi Wang, Qidong Zhao, Dongkuan Xu, and Xu Liu. Purpose enhanced reasoning through iterative prompting: Uncover latent robustness of chatgpt on code comprehension. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pp. 6513–6521. ijcai.org, 2024c. URL https://www.ijcai.org/proceedings/2024/720.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

Yuwei Yan, Yu Shang, Qingbin Zeng, Yu Li, Keyu Zhao, Zhiheng Zheng, Xuefei Ning, Tianji Wu, Shengen Yan, Yu Wang, Fengli Xu, and Yong Li. Agentsociety challenge: Designing LLM agents for user modeling and recommendation on web platforms. *CoRR*, abs/2502.18754, 2025. doi: 10.48550/ARXIV.2502.18754. URL https://doi.org/10.48550/arXiv.2502.18754.

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, et al. Oasis: Open agent social interaction simulations with one million agents. *arXiv preprint arXiv:2411.11581*, 2024.

Appendix: Table of Contents

A `FASim` Construction and Execution Infrastructure **16**
- A.1 Data Schema Construction . . . 16
- A.2 Chain-of-Structure and Blueprint Generation . . . 16
- A.3 Simulator Code Generation . . . 17
- A.4 Parameter-Calibration Backends . . . 17
- A.5 Diagnosis-to-Code Refinement . . . 18
- A.6 Falsification Execution Infrastructure . . . 19
- A.7 Agent Prompts and Output Schemas . . . 23

B Detailed Workflow of the `FASim` System **25**

C Experimental Setup and Resource Analysis **29**

D Supplementary Experiments **31**
- D.1 End-to-End Falsification Case Study: Trip-Distance Structural Repair . . . 31
- D.2 Controlled Recovery of Known Structural Defects . . . 32
- D.3 Blinded Evaluation of Diagnosis and Repair Quality . . . 35
- D.4 Backbone Sensitivity . . . 36

E An Example of Blueprint **36**

F Prompts in `FASim` **49**
- F.1 Chain-of-Structure Prompt . . . 49
- F.2 Data Analysis Prompt . . . 52
- F.3 Code Generation Prompt . . . 58
- F.4 Code Patch Prompt . . . 60
- F.5 Simulation Execution Prompt . . . 62
- F.6 Hypothesis Agent Prompt . . . 63
- F.7 Falsification Agent Probe Planning Prompt . . . 67
- F.8 Falsification Agent Evidence Evaluation Prompt . . . 73

G Limitations and Future Work **75**

H Societal Impact **76**

I Ethics Statement **76**

A FASIM CONSTRUCTION AND EXECUTION INFRASTRUCTURE

This appendix describes the construction, calibration, execution, code-refinement, and falsification-execution infrastructure underlying the methodology in the main paper. These components provide the executable substrate on which structural hypotheses can be generated, probed, falsified, and subsequently repaired. They are not individually the primary methodological contribution of this work; rather, together they instantiate the simulator-construction and executable-testing backbone summarized in Section 3.1.

A.1 DATA SCHEMA CONSTRUCTION

FASim receives three main sources of task information: the observational dataset D, a natural-language task description T, and optional auxiliary task data Q. Because these sources may differ substantially across simulation tasks, FASim first converts them into a unified data schema

$$D_s = \textsc{BuildDataSchema}(D, T, Q).$$

The schema explicitly records the information required to construct and evaluate an executable simulator, including:

- the set and population of micro-entities;
- observable and latent state variables, together with their types, ranges, units, and temporal resolution;
- interaction structures such as graphs, layers, neighborhoods, or other relational structures;
- exogenous drivers u_t, interventions, and their temporal scope;
- historical or contextual variables available to micro-entities;
- macro-level observations and aggregation operators $\mathcal{A}(\cdot)$;
- training, calibration, validation, and OOD evaluation windows;
- task-specific fidelity metrics and constraints.

The purpose of D_s is to provide a task-independent interface between heterogeneous raw inputs and subsequent simulator construction. In particular, it makes explicit which quantities are observed, which are simulated, which can be intervened upon, and which outputs are available for evaluation and falsification.

A.2 CHAIN-OF-STRUCTURE AND BLUEPRINT GENERATION

Given the unified schema D_s and task description T, the *Data Analysis Agent (DAA)* constructs a mechanistic blueprint

$$B = \mathcal{A}_{\mathrm{DAA}}(D_s, T)$$

using Chain-of-Structure (CoS) prompting. CoS decomposes simulator specification into a sequence of focused structural decisions rather than asking an LLM to produce a monolithic design in one step.

The generated blueprint specifies:

1. the overall simulation objective and modeling stance;
2. temporal and spatial granularity;
3. micro-entity archetypes, attributes, and dynamic states;
4. interaction topology and influence pathways;
5. information or influence propagation mechanisms where applicable;
6. exogenous signals, interventions, and their access paths;
7. decision and state-transition rules;
8. validation and OOD evaluation strategy;
9. macro-level aggregation operators and fidelity metrics;

10. calibratable parameters, including their bounds, priors, and coupling constraints.

The blueprint therefore provides a structured construction scaffold for the initial simulator. In particular, explicit representation of exogenous inputs, intervention hooks, interaction paths, and state-transition rules makes the resulting program amenable to later controlled probing.

Importantly, in the main falsification procedure, the blueprint is used as construction context rather than as a ground-truth oracle for validating structural hypotheses. A hypothesis is evaluated through executable evidence, not merely through consistency with the LLM-generated blueprint.

CoS is inspired by structured chain-of-thought-style decomposition Wei et al. (2022); Wang et al. (2024c). The full prompts used to construct D_s and B are provided in § F.1.

A.3 Simulator Code Generation

The *Code Generation Agent (CGA)* converts the mechanistic blueprint and unified data schema into the initial executable simulator:

$$(\lambda_0, \phi_0, C_0) = \mathcal{A}_{\mathrm{CGA}}(B, D_s),$$

where λ_0 denotes the initial simulator structure, ϕ_0 denotes the associated calibration schema, and C_0 denotes the executable program. Later structural updates use the same CGA through a refinement interface that additionally receives the current simulator, actionable repair set, and execution evidence; this diagnosis-gated update is described in Appendix A.5.

The generated structure λ implements:

- micro-entity state representations;
- interaction and communication topology;
- exogenous and intervention access paths;
- decision and transition functions;
- micro-to-macro aggregation procedures;
- tunable parameter interfaces;
- execution and evaluation hooks required by the calibration schema.

The CGA also exposes the numerical parameter vector ω and specifies the calibration schema ϕ, including data splits, objective computation, parameter bounds or priors, execution mode, and the selected calibration algorithm. Structural refinement may subsequently modify the exposed parameterization when an accepted code edit introduces, removes, or redefines calibratable quantities, but the task-level calibration backend remains fixed as described in Appendix A.4.

The blueprint representation follows the broader principle of explicitly describing agent-based simulation structure before execution Grimm et al. (2010); Müller et al. (2013); Laatabi et al. (2018); Grimm et al. (2020); Wang et al. (2025).

A.4 Parameter-Calibration Backends

Parameter calibration is treated as a modular fixed-structure component of `FASim` rather than as part of the falsification mechanism. For a simulator with fixed structure λ, the calibration schema ϕ specifies the exposed numerical parameters, their ranges or priors, the task-specific calibration objective, the evaluation budget, and a parameter-calibration backend. We write the generic calibration problem as

$$\omega^* = \mathcal{C}_\phi(C(\lambda, \cdot), D)\,,$$

where $\mathcal{C}_\phi$ denotes the backend instantiated by ϕ. The current implementation provides three predefined backends: Random Search, Bayesian Optimization (BO), and Simulation-Based Inference (SBI). They cover complementary calibration paradigms—generic derivative-free search, sample-efficient black-box optimization, and likelihood-free parameter inference, respectively. The choice of backend is orthogonal to the falsification-guided structural diagnosis in the main method.

Random Search. Random Search samples parameter configurations independently from the ranges or priors specified in ϕ,

$$\omega^{(n)} \sim q_\phi(\omega), \qquad n = 1, \ldots, N_{\text{cal}},$$

executes the simulator for each configuration, and retains the setting with the lowest calibration objective,

$$\omega^* = \arg\min_{\omega^{(n)}} J(\lambda, \omega^{(n)}; D).$$

Random Search provides a simple derivative-free calibration backend without surrogate modeling or learned posterior inference.

Bayesian Optimization. BO treats the task-specific fidelity objective $J(\lambda, \omega; D)$ as a black-box function over the exposed parameter space. A Gaussian-process surrogate is fitted to previously evaluated parameter configurations, and new candidates are selected through an acquisition function such as Expected Improvement or Upper Confidence Bound (Jones et al., 1998). For higher-dimensional or expensive simulator objectives, our implementation uses Trust-Region Bayesian Optimization (TuRBO) (Eriksson et al., 2019), which performs local BO within adaptive trust regions. For stochastic simulators, each objective evaluation may aggregate multiple rollouts or use fixed simulation seeds. BO returns a calibrated point estimate ω^*.

Simulation-Based Inference. SBI treats calibration as likelihood-free parameter inference. Given observed task summaries y_{obs}, the goal is to approximate

$$p(\omega \mid y_{\text{obs}})$$

without requiring an analytic likelihood. We instantiate SBI with Neural Posterior Estimation (NPE) (Papamakarios & Murray, 2016) and Masked Autoregressive Flows (MAF) (Papamakarios et al., 2017). Simulation pairs

$$\left\{(\omega^{(n)}, y^{(n)})\right\}_{n=1}^{N_{\text{sim}}}$$

are generated by sampling parameters from the schema-defined prior and executing the fixed simulator. A conditional density estimator is trained on these pairs to approximate the posterior over ω. Posterior-predictive evaluation can then propagate parameter uncertainty through simulator executions (Cranmer et al., 2020; Boelts et al., 2023; Preston et al., 2025; Deistler et al., 2025).

Backend configuration across refinement. The calibration backend is fixed for a task. Structural refinement may nevertheless modify the calibration schema ϕ_t when a persistent code edit introduces, removes, or redefines calibratable quantities. Accordingly, parameter identities, ranges, priors, or coupling constraints may change across outer iterations, whereas the calibration paradigm itself is not switched as part of structural diagnosis. Each candidate simulator is recalibrated using the same task-level backend before its fidelity is evaluated.

Task-specific parameter ranges, priors, calibration budgets, stochastic rollout counts, TuRBO settings, and SBI architecture and posterior-sampling details are reported with the corresponding benchmark configuration.

A.5 Diagnosis-to-Code Refinement

Once issue routing and falsification produce the actionable repair set

$$\mathcal{D}_t = \mathcal{D}_t^{\text{dir}} \cup \mathcal{D}_t^{\text{fal}},$$

the Code Generation Agent converts these supported repair directives into a persistent candidate simulator update. Here, $\mathcal{D}_t^{\text{dir}}$ contains directly verified implementation issues whose causes are established from executable evidence, whereas $\mathcal{D}_t^{\text{fal}}$ contains structurally ambiguous diagnoses that have passed the falsification gate. Only the latter require competing hypotheses and executable falsification before they can authorize repair.

This stage is deliberately separated from diagnosis. Issue routing and falsification determine *which failures are sufficiently supported to justify modification*, whereas code refinement determines *how those supported changes should be implemented*. In particular, the CGA does not receive all proposed

structural hypotheses: falsified and inconclusive explanations are excluded from the actionable repair set.

For each actionable repair item, the refinement procedure receives the corresponding issue or supported diagnosis, the residual and execution evidence associated with that issue, relevant execution traces or code locations, the current simulator implementation C_t, and the blueprint B as construction context. The CGA then produces a localized repair proposal that identifies the implicated mechanism or update rule, the implementation region realizing that mechanism, the minimal persistent modification required to address the supported issue, and any corresponding changes to the exposed parameterization or calibration schema.

When multiple actionable items are present in the same outer iteration, the CGA considers them jointly and produces a coherent candidate program rather than independently committing potentially conflicting patches. Formally,

$$(\tilde{\lambda}_{t+1}, \tilde{\phi}_{t+1}, \tilde{C}_{t+1}) = \mathcal{A}_{\mathrm{CGA}}\left(B, C_t, \mathcal{D}_t, \delta_t\right).$$

The resulting candidate is re-validated for executability and recalibrated before its fidelity is compared with the currently retained simulator. This stage therefore implements the transition

$$\text{actionable evidence} \rightarrow \text{localized persistent edit} \rightarrow \text{recalibrated candidate simulator},$$

while preserving the distinction between transient falsification probes and persistent structural repair. Directly verified implementation issues reach this stage through the direct route, whereas structurally ambiguous issues reach it only after executable falsification yields a supported diagnosis.

A.6 Falsification Execution Infrastructure

The execution infrastructure in `FASim` separates ordinary simulator execution from the more tightly controlled execution required for falsification. The *Simulation Execution Agent (SEA)* remains responsible for ordinary fixed-structure calibration and simulator evaluation. For a simulator $C(\lambda, \omega, \phi)$ and execution configuration η, SEA returns

$$\mathcal{E}(C, \eta) = (\hat{x}, \hat{y}, J, \delta, \tau),$$

where $\hat{x}$ denotes micro-level execution states, $\hat{y}$ denotes aggregated outputs, J is the fidelity objective, δ contains structured residual evidence, and τ denotes optional internal execution traces.

Falsification probes impose stricter experimental requirements than ordinary simulator execution. Probe runs must preserve the calibrated parameter state, match random seeds between baseline and intervention conditions, alter only the intervention declared in the frozen plan, collect the requested observables, verify that the intervention was actually exercised, and reject evidence when simulator identity, parameter identity, or execution conditions drift. For this reason, executable falsification is not delegated to an LLM agent or to the ordinary SEA interface. Instead, `FASim` uses a dedicated deterministic probe-execution layer.

The resulting execution chain is

$$\begin{aligned}&\text{Hypothesis Agent} \rightarrow \text{Falsification Agent.plan()}\\ &\rightarrow \text{capability validation and probe materialization} \rightarrow \text{freeze}\\ &\rightarrow \textsc{ProbeExecutor} \rightarrow \text{evidence bundle}\\ &\rightarrow \text{Falsification Agent.evaluate()} \rightarrow \text{deterministic diagnosis gate}.\end{aligned}$$

This separation establishes a strict responsibility boundary. The Falsification Agent determines *what executable evidence would discriminate among the competing hypotheses*; deterministic infrastructure determines whether that experiment is legally realizable on the current simulator, binds it to an executable contract, runs it, and validates the resulting evidence. The Code Generation Agent does not participate in probe execution and receives only repair directives admitted by the downstream diagnosis gate.

Probe capability preflight. Before a falsification plan is generated, `FASim` dynamically inspects the current executable simulator and constructs a probe-capability manifest. The manifest records the

interventions and observables already exposed by the simulator, internal assignment points that can support temporary read-only instrumentation or scalar overrides, whether candidate access points are exercised in actual execution, their runtime value types, and their compatibility with the supported evidence collectors.

The Falsification Agent therefore does not design an experiment against a purely textual or hypothetical description of the simulator. Its available actions are restricted to runtime-validated capabilities of the current executable program. The capability manifest may expose, for example, existing runtime hooks, observation-only access points, or existing scalar states that can be temporarily overridden without introducing a new mechanism. If no available access path can support a discriminating experiment for the active hypothesis set, the planning stage returns an unresolved result instead of inventing a new structural pathway.

Semantic probe planning and deterministic contract validation. Given the active hypothesis set and the capability manifest, the Falsification Agent produces a semantic probe plan containing the intervention, access mode, requested observables, evidence regime, matched seeds when required, stability rule, predicted signature for each hypothesis, falsification condition, and corresponding evidence requirements. At this stage the plan remains an LLM-generated proposal and is not yet executable.

A deterministic contract layer validates the proposed plan before any simulator process is launched. Among other checks, it verifies that the requested access mode is supported, each intervention target exists, requested observables are available, instrumentation points originate from the capability manifest, evidence collectors are type-compatible, quantitative signatures declare an effect direction, matched seeds are valid and unique, calibrated parameters are held fixed, and every frozen predicate is backed by the required executable evidence.

A plan that violates this contract is rejected rather than interpreted leniently by an LLM. Thus, natural-language experimental reasoning is translated into an executable experiment only after satisfying a machine-checkable interface.

Probe materialization. Valid semantic plans are materialized through one of three bounded access modes.

Existing hook. If the simulator already exposes both the required intervention and the corresponding observables, the probe is mapped directly onto those runtime interfaces without modifying simulator code.

Observation instrumentation. If the required intervention already exists but the relevant internal state is not externally observable, the materializer may construct an isolated instrumented runner that records the existing state at a validated access point. Such instrumentation is read-only and does not alter the simulator's state-transition semantics.

Temporary intervention adapter. If a relevant scalar state already exists in the current simulator but lacks an intervention interface, a temporary adapter may override that existing state at a validated assignment point. The original right-hand side is still executed once, preserving its computation and random-number consumption, and the resulting state is overridden only in the probe run. The baseline execution does not apply this override.

These temporary mechanisms may expose or manipulate an existing pathway for experimental purposes, but they may not add an absent mechanism, alter the persistent simulator structure, modify the calibrated parameter vector, or serve as a structural repair. The materialized runner is isolated from the persistent program and is discarded after evidence collection.

Supported bounded evidence. Temporary instrumentation is restricted to bounded evidence collectors rather than unrestricted runtime dumps. Depending on the validated access point, the current implementation supports scalar summaries, histograms, quantiles, categorical counts, grouped histograms, paired records, and bounded event traces. Structured collectors operate on the scalar state exposed at the validated access point together with explicitly declared read-only context.

Numeric histogram- or quantile-based evidence is permitted only for runtime points verified as numerically compatible. Boolean or categorical states may instead use scalar predicates, categorical

counts, paired records, or event traces. This typed evidence contract prevents later adjudication from claiming distributional, paired, or temporal relationships that were not actually collected by the runtime.

Controlled probe execution. A falsification probe changes a controlled execution condition while preserving the current simulator as the object being diagnosed. Depending on the probe, the execution infrastructure may alter an exogenous input, temporarily override an existing internal state, or expose additional execution traces required to observe the hypothesized pathway. Such probe-specific modifications are transient and are not committed as structural repairs.

The execution infrastructure records both baseline and probe conditions, including random seeds, calibrated parameters, initialization state, execution identity, and requested observables. For stochastic signatures, baseline and probe executions use matched seeds so that their response difference is less confounded by simulator randomness.

The intended comparison is therefore

$$R_0(s) = R(p_0; C_t, \omega_t^*, s), \qquad R_p(s) = R(p_{t,j}; C_t, \omega_t^*, s),$$

where s is a frozen random seed, p_0 denotes the corresponding unperturbed execution, and $p_{t,j}$ is the frozen probe. Except for the declared intervention, simulator structure, calibrated parameters, data, initial state, topology, timing, and stochastic seed are held fixed.

Frozen execution contract. After validation and materialization, the executable experiment is frozen before the outcome is observed. The frozen contract records the active hypotheses, intervention, access mode, observables, evidence types, instrumentation points, random seeds, stability rule, predicted signatures, falsification conditions, calibrated-parameter identity, simulator identity, execution command, bound runtime arguments, environment configuration, and evidence schema.

A canonical plan hash is attached to the frozen contract and propagated into all subsequent executions and evidence artifacts. Freezing therefore means more than persisting an LLM-generated prediction: both the executor and the later adjudicator are constrained to the same pre-execution experimental contract, preventing predictions, observables, thresholds, or falsification criteria from being altered after the result is known.

Execution-contract preflight. Before freezing the experiment, the infrastructure also verifies that the current simulator can actually be launched under the probe runtime contract. Required command-line arguments, execution entry points, paths, and environment bindings are resolved in advance. If a mandatory runtime requirement cannot be bound, the probe is marked non-executable and is not launched.

This preflight prevents runtime-interface failures from being confused with scientific evidence. In particular, a probe that cannot be executed under the specified simulator interface does not produce a hypothesis verdict.

Deterministic ProbeExecutor. Actual falsification runs are performed by a dedicated non-LLM component, the PROBEEXECUTOR. The executor launches real simulator processes under the frozen execution contract and never synthesizes evidence through LLM reasoning. For matched-seed evidence, each frozen seed produces a paired baseline/probe execution:

$$\text{baseline}(s) \rightarrow \text{probe}(s) \rightarrow \text{paired evidence}.$$

The baseline receives no probe intervention, whereas the probe receives exactly the runtime intervention encoded in the frozen contract. Both executions use the same calibrated parameters and all other frozen execution conditions. The simulator is instructed to skip recalibration during probe execution, load the supplied fixed parameter state, initialize the specified random sources, collect only the requested evidence, and emit a structured evidence artifact.

For scalar evidence, the executor may additionally compute paired effects

$$\Delta_s = Y_{\text{probe}}(s) - Y_{\text{baseline}}(s),$$

along with bounded summaries such as the mean effect and the fractions of positive, negative, or unchanged matched responses. Typed evidence collectors retain their corresponding bounded histogram, quantile, categorical, grouped, paired-record, or event-trace representation.

Baseline reuse across sequential probes. When sequential probes require an identical baseline observation contract, a previously validated baseline execution may be reused to reduce simulator calls. Such reuse is permitted only when simulator identity, source-simulator identity, calibrated parameters, seed, requested observables, instrumentation configuration, and execution entry point are identical. The cache key is a canonical hash over these fields rather than a loose iteration identifier.

This optimization therefore changes computational cost but not the scientific baseline: any change to the observation or execution contract forces the baseline to be re-executed.

Evidence auditing. A process exit code of zero is not sufficient for accepting a probe run. After each execution, the PROBEEXECUTOR validates the emitted evidence artifact. The audit checks that the evidence is present and well-formed, that the run identifier and frozen plan hash match the requested execution, that the execution role and seed are correct, that all required observables are present, that the calibrated parameters actually used match the frozen state, and that the declared intervention was applied exactly as requested.

Any identity, parameter, intervention, or evidence mismatch is represented as a structured execution failure and cannot be interpreted as scientific counterevidence. This ensures that a simulator cannot be treated as having passed a probe merely because a process completed successfully while silently ignoring the requested intervention.

Conditional observations and unavailable evidence. Read-only observation points may lie on conditional execution paths and may therefore not be exercised for every random seed. A missing conditional observation is recorded as structured unavailable evidence rather than automatically invalidating the entire experiment. The downstream adjudication stage then determines whether enough valid matched evidence remains to satisfy the frozen stability rule.

Intervention targets have stricter semantics. If the intervention target is never exercised, the experimental manipulation itself did not occur and the corresponding probe execution is invalid. This distinction separates *missing observation* from *missing intervention*.

Evidence bundle. After all valid executions have completed, the executor packages the results into a structured evidence bundle. For matched-seed probes, the bundle contains the baseline and probe evidence for each seed, paired differences where defined, aggregate summaries, actual simulator-call accounting, and baseline-reuse metadata. It also carries the frozen plan identity and simulator identity required for later audit.

The evidence bundle is the only post-execution evidence exposed to the adjudication stage; neither the Hypothesis Agent nor the Code Generation Agent can replace it with free-form claims about what the simulator "should have" done.

Threshold compilation. For quantitative signatures, the Falsification Agent freezes the requested observable and qualitative effect direction—for example, INCREASE, DECREASE, or NO-MEANINGFUL-CHANGE—but does not choose a post-hoc numerical cutoff. After successful baseline execution, the workflow controller instantiates the pre-specified threshold rule from baseline variability and attaches the realized meaningful-effect threshold to the evidence bundle.

The division of responsibility is therefore

$$\begin{gathered}\text{PROBEEXECUTOR} \rightarrow \text{paired execution evidence},\\ \text{threshold compiler} \rightarrow \text{pre-specified realized cutoff},\\ \text{Falsification Agent.evaluate()} \rightarrow \text{hypothesis verdicts}.\end{gathered}$$

Neither the executor nor the Falsification Agent may move or relax the threshold after observing the probe result.

Evidence adjudication and failure semantics. Once valid probe evidence and controller-owned thresholds are available, the Falsification Agent is invoked in an evidence-adjudication role. It evaluates each hypothesis only against the predictions and falsification conditions contained in the frozen plan and returns

$$v_{t,j}^{(k)} \in \{\text{FALSIFIED}, \text{SURVIVES}, \text{INCONCLUSIVE}\}.$$

A SURVIVES verdict indicates only that the available execution evidence is consistent with the frozen signature; it does not establish that the hypothesis is a true real-world causal mechanism. Likewise, execution failure is never interpreted as falsification:

$$\text{probe execution failure} \neq \text{hypothesis falsification.}$$

Runtime failure, unavailable required evidence, an unaudited intervention, insufficient valid matched pairs, or an unavailable required controller threshold therefore prevents a scientific FALSIFIED verdict and instead yields a non-evaluable or INCONCLUSIVE outcome.

Only successfully adjudicated hypothesis verdicts are passed to the deterministic diagnosis gate. A structural diagnosis becomes actionable only when exactly one hypothesis survives and every competing explanation is falsified. The adjudicator itself does not select the final diagnosis and does not propose a repair.

Probe isolation. Probe execution is isolated from persistent simulator refinement. After evidence is collected, any temporary instrumentation, materialized runner, or intervention adapter introduced solely for the experiment is discarded. Only the subsequent diagnosis-to-code refinement stage can produce a persistent new simulator structure.

This distinction ensures that

$$\text{probe modification} \neq \text{structural repair.}$$

Taken together, the falsification infrastructure separates LLM-based experimental reasoning from deterministic execution. The Falsification Agent designs discriminating tests and interprets valid evidence; contract validation, materialization, freezing, process execution, matched pairing, evidence auditing, threshold compilation, and budget accounting are enforced by deterministic infrastructure; and the Code Generation Agent receives only diagnoses admitted by the downstream repair gate.

A.7 AGENT PROMPTS AND OUTPUT SCHEMAS

`FASim` uses structured prompts and machine-readable output schemas to constrain the roles of its five LLM-driven agents: the Data Analysis Agent, Code Generation Agent, Simulation Execution Agent, Hypothesis Agent, and Falsification Agent. An outer workflow controller coordinates these components and applies deterministic routing, contract enforcement, adjudication support, gating, and retain/reject logic, but is not itself treated as an additional reasoning agent. The falsification subsystem additionally contains non-LLM infrastructure, including probe-capability discovery, contract validation, probe materialization, the PROBEEXECUTOR, evidence auditing, and threshold compilation. We summarize the principal interfaces below; the complete prompt templates are provided in the supplementary material accompanying the implementation.

Data Analysis Agent. The Data Analysis Agent (DAA) receives (D_s, T) and outputs the mechanistic blueprint

$$B = \mathcal{A}_{\mathrm{DAA}}(D_s, T).$$

Its schema records entities, states, interaction topology, exogenous signals, intervention pathways, transition rules, macro-level aggregations, evaluation settings, and calibratable quantities. The resulting blueprint provides construction context for downstream agents but is not used as a ground-truth oracle for structural diagnosis. Prompt details are provided in § F.2.

Code Generation Agent. The Code Generation Agent (CGA) serves two related roles. During initial construction, it receives (B, D_s) and produces the executable simulator, exposed parameter definitions, and calibration schema. During structural refinement, it receives the current simulator C_t, the actionable repair set $\mathcal{D}_t = \mathcal{D}_t^{\mathrm{dir}} \cup \mathcal{D}_t^{\mathrm{fal}}$, the blueprint, and associated execution evidence. Its refinement output contains a minimally modified executable simulator, an updated calibration schema when necessary, and a concise record of persistent structural modifications. Falsified or inconclusive structural hypotheses are not forwarded as repair directives.

During diagnosis-gated refinement, the CGA may additionally receive an iteration-specific coding patch compiled from the actionable repair set and a reusable implementation Playbook. The Playbook provides implementation guidance only: it does not establish structural validity and cannot authorize an edit that has not entered the actionable repair set. Prompt details are provided in § F.3 and § F.4.

Hypothesis Agent. For each structurally ambiguous issue $I_{t,j} \in \mathcal{I}_t^{\mathrm{str}}$, the Hypothesis Agent returns a set of competing structural explanations,

$$\mathcal{H}_{t,j} = \mathcal{A}_H(I_{t,j}, \delta_t, B, C_t) = \{h_{t,j}^{(1)}, \ldots, h_{t,j}^{(K_j)}\}.$$

Each hypothesis contains a structural explanation of the observed issue, the mechanism or executable pathway implicated by that explanation, and an observable consequence that should arise under an appropriate controlled execution. The agent is instructed to generate competing explanations rather than a single preferred diagnosis or repair. Explanations that cannot be connected to an observable executable consequence cannot justify persistent structural modification. Prompt details are provided in § F.6.

Falsification Agent: probe planning. The Falsification Agent acts as a challenger to the hypotheses proposed above. It receives the competing hypothesis set, current simulator context, and the runtime-validated probe-capability manifest and produces a semantic falsification plan

$$\Pi_{t,j} = \mathcal{A}_F(\mathcal{H}_{t,j}, C_t) = \left(p_{t,j}, \{s_{t,j}^{(k)}, f_{t,j}^{(k)}\}_{k=1}^{K_j}, \rho_{t,j}\right).$$

Its output schema specifies the counterfactual probe, access mode, controlled variables and execution context, requested observables and evidence types, the predicted signature associated with each competing hypothesis, its corresponding falsification condition, and the evidence-collection regime. The plan itself is subsequently validated, materialized, and frozen by deterministic infrastructure before execution. The Falsification Agent does not directly launch simulator processes, modify the persistent simulator, or change calibrated parameters. Prompt details are provided in § F.7.

Falsification Agent: evidence adjudication. After the frozen probe has been executed and a validated evidence bundle has been produced, the Falsification Agent is invoked in a separate adjudication role. It may not change, relax, reinterpret, or add predictions, thresholds, observables, seeds, or falsification conditions after seeing the outcome. Instead, it evaluates each hypothesis strictly against the frozen criteria and the supplied execution evidence.

The resulting hypothesis-level verdict is

$$v_{t,j}^{(k)} = \Gamma\left(s_{t,j}^{(k)}, f_{t,j}^{(k)}, \mathcal{R}_{t,j}\right) \in \{\textsc{Falsified}, \textsc{Survives}, \textsc{Inconclusive}\}.$$

Whenever a frozen predicate is directly computable from the recorded evidence, the workflow controller can apply the corresponding deterministic rule directly. LLM interpretation is used only where the frozen signature requires structured interpretation of bounded execution traces. In either case, the signature, falsification condition, and any controller-owned quantitative threshold remain fixed after the outcome is observed.

The adjudication stage returns exactly one verdict per hypothesis. It does not select a final diagnosis and does not propose a repair; the downstream unique-survivor rule remains controller-enforced. Prompt details are provided in § F.8.

Simulation Execution Agent. The Simulation Execution Agent (SEA) provides the ordinary execution interface for fixed-structure calibration and simulator evaluation. It receives an executable simulator, calibration configuration, and the relevant data split, and returns calibrated parameters where applicable, simulator outputs, fidelity metrics, residual evidence, and requested execution traces.

Falsification probes use the same underlying executable simulator but are not run through an unconstrained SEA decision process. They are executed by the dedicated deterministic ProbeExecutor under the validated and frozen probe contract described in Appendix A.6. Accordingly, SEA supplies the ordinary calibrated simulator state and execution evidence from which issues are detected, whereas ProbeExecutor is responsible for controlled baseline/probe execution during structural falsification. Prompt details are provided in § F.5.

Iteration-level orchestration. The outer controller coordinates the five agents and the deterministic falsification infrastructure but does not itself perform open-ended structural reasoning. At each outer

iteration it: (i) calibrates and executes the current simulator through SEA; (ii) constructs residual and execution evidence; (iii) routes directly verifiable implementation issues to $\mathcal{D}_t^{\mathrm{dir}}$ and structurally ambiguous issues to the Hypothesis Agent; (iv) performs probe-capability preflight and invokes the Falsification Agent to construct a candidate probe plan for each active ambiguous issue; (v) validates and materializes the plan, binds the execution contract, and freezes it before outcome observation; (vi) executes the frozen probe through PROBEEXECUTOR, validates the resulting evidence, and compiles any controller-owned quantitative thresholds; (vii) invokes the Falsification Agent's adjudication interface, or applies directly computable frozen predicates, to obtain hypothesis-level verdicts; (viii) admits a structural diagnosis to $\mathcal{D}_t^{\mathrm{fal}}$ only when exactly one hypothesis survives and all competing explanations are falsified; (ix) combines $\mathcal{D}_t^{\mathrm{dir}}$ and $\mathcal{D}_t^{\mathrm{fal}}$ into the actionable repair set $\mathcal{D}_t$ and forwards it to the CGA; (x) recalibrates the resulting candidate simulator through SEA; and (xi) retains or rejects the candidate according to the task-level fidelity objective.

Issue-, hypothesis-, probe-, and verdict-level diagnostic state is reconstructed from the currently retained simulator in each new outer iteration rather than being preserved as persistent diagnostic memory. Optimization state persists through the retained simulator and its calibration state, while diagnostic state is reset after each retain/reject decision.

B DETAILED WORKFLOW OF THE FASIM SYSTEM

The **FASim pipeline** constructs an executable simulator and iteratively refines its structure while keeping numerical calibration and structural diagnosis conceptually separate. The workflow coordinates three simulator components:

- **Structure (λ):** the executable mechanism specification, including micro-entity states, interaction topology, exogenous/intervention access paths, and decision or transition rules.
- **Parameters (ω):** tunable numerical quantities within the current structure, such as rates, weights, probabilities, and thresholds.
- **Calibration schema (ϕ):** the fixed-structure calibration configuration, including exposed parameters, ranges or priors, objective, budget, data splits, and the task-level calibration backend.

The resulting workflow alternates between parameter calibration under a fixed structure and diagnosis-gated structural refinement, as summarized in Algorithm 1.

STAGE I: MECHANISTIC BLUEPRINTING AND INITIAL CONSTRUCTION

Given dataset D, task description T, and auxiliary task data Q, FASim first constructs a unified data schema

$$D_s \leftarrow \textsc{BuildDataSchema}(D, T, Q).$$

The schema records the micro-entities, observable and latent states, interaction structures, exogenous signals and interventions, evaluation windows, and task-specific macro-level evidence.

The **Data Analysis Agent (DAA)** then applies Chain-of-Structure reasoning to produce a mechanistic blueprint

$$B \leftarrow \mathcal{A}_{\mathrm{DAA}}(D_s, T).$$

The blueprint specifies the initial modeling scaffold, including micro-entity rules, interaction and intervention pathways, macro-level aggregation and evaluation interfaces, and calibratable quantities. Importantly, the blueprint provides construction context rather than a ground-truth oracle for subsequent structural diagnosis: later hypotheses are evaluated through executable evidence.

The **Code Generation Agent (CGA)** converts this scaffold into the initial executable simulator,

$$(\lambda_0, \phi_0, C_0) \leftarrow \mathcal{A}_{\mathrm{CGA}}(B, D_s).$$

The initial schema ϕ_0 selects a task-level calibration backend from the predefined library and exposes the corresponding tunable parameters, ranges or priors, objective, and calibration budget. The backend remains fixed for the task, although later structural edits may change which parameters are exposed by ϕ_t.

Algorithm 1 FASim with Falsification-Guided Structural Diagnosis

Input: Dataset D, task description T, auxiliary data Q, maximum outer iterations $T_{\max}$, non-improvement patience K
Output: Retained simulator $C^\star(\lambda^\star, \omega^\star, \phi^\star)$

Construction backbone
$D_s \leftarrow \textsc{BuildDataSchema}(D, T, Q)$
$B \leftarrow \mathcal{A}_{\mathrm{DAA}}(D_s, T)$
$(\lambda_0, \phi_0, C_0) \leftarrow \mathcal{A}_{\mathrm{CGA}}(B, D_s)$
$(\omega_0^*, J_0, \delta_0) \leftarrow \mathcal{A}_{\mathrm{SEA}}(C_0, D, \phi_0)$
$t \leftarrow 0$
no_improve $\leftarrow 0$
Outer refinement loop
while $t < T_{\max}$ **do**
  Issue detection and routing
  $(\mathcal{I}_t^{\mathrm{dir}}, \mathcal{I}_t^{\mathrm{str}}) \leftarrow \textsc{RouteIssues}(\delta_t, B, C_t)$
  $\mathcal{D}_t^{\mathrm{dir}} \leftarrow \textsc{DirectDiagnose}(\mathcal{I}_t^{\mathrm{dir}}, \delta_t, C_t)$
  $\mathcal{D}_t^{\mathrm{fal}} \leftarrow \emptyset$
  for all $I_{t,j} \in \mathcal{I}_t^{\mathrm{str}}$ **do**
    Competing structural hypotheses
    $\mathcal{H}_{t,j} \leftarrow \mathcal{A}_{\mathrm{H}}(I_{t,j}, \delta_t, B, C_t)$
    Frozen falsification plan
    $\Pi_{t,j} \leftarrow \mathcal{A}_{\mathrm{F}}(\mathcal{H}_{t,j}, C_t)$
    ▷ $\Pi_{t,j} = (p_{t,j}, \{s_{t,j}^{(k)}, f_{t,j}^{(k)}\}_{k=1}^{K_j}, \rho_{t,j})$
    Executable falsification
    $\mathcal{R}_{t,j} \leftarrow \textsc{ExecuteProbe}(\Pi_{t,j}, C_t, \omega_t^*)$
    for all $h_{t,j}^{(k)} \in \mathcal{H}_{t,j}$ **do**
      $v_{t,j}^{(k)} \leftarrow \Gamma(s_{t,j}^{(k)}, f_{t,j}^{(k)}, \mathcal{R}_{t,j})$
    end for
    $d_{t,j} \leftarrow \textsc{UniqueSurvivor}(\mathcal{H}_{t,j}, \{v_{t,j}^{(k)}\}_{k=1}^{K_j})$
    if $d_{t,j} \neq \varnothing$ **then**
      $\mathcal{D}_t^{\mathrm{fal}} \leftarrow \mathcal{D}_t^{\mathrm{fal}} \cup \{d_{t,j}\}$
    end if
  end for
  Diagnosis-gated structural refinement
  $\mathcal{D}_t \leftarrow \mathcal{D}_t^{\mathrm{dir}} \cup \mathcal{D}_t^{\mathrm{fal}}$
  if $\mathcal{D}_t = \emptyset$ **then**
    no_improve $\leftarrow$ no_improve $+ 1$
  else
    $(\tilde{\lambda}_{t+1}, \tilde{\phi}_{t+1}, \tilde{C}_{t+1}) \leftarrow \mathcal{A}_{\mathrm{CGA}}(B, C_t, \mathcal{D}_t, \delta_t)$
    $(\tilde{\omega}_{t+1}^*, \tilde{J}_{t+1}, \tilde{\delta}_{t+1}) \leftarrow \mathcal{A}_{\mathrm{SEA}}(\tilde{C}_{t+1}, D, \tilde{\phi}_{t+1})$
    if $\tilde{J}_{t+1} < J_t - \epsilon_{\mathrm{imp}}$ **then**
      $(\lambda_{t+1}, \phi_{t+1}, C_{t+1}) \leftarrow (\tilde{\lambda}_{t+1}, \tilde{\phi}_{t+1}, \tilde{C}_{t+1})$
      $(\omega_{t+1}^*, J_{t+1}, \delta_{t+1}) \leftarrow (\tilde{\omega}_{t+1}^*, \tilde{J}_{t+1}, \tilde{\delta}_{t+1})$
      no_improve $\leftarrow 0$
    else
      $(\lambda_{t+1}, \phi_{t+1}, C_{t+1}) \leftarrow (\lambda_t, \phi_t, C_t)$
      $(\omega_{t+1}^*, J_{t+1}, \delta_{t+1}) \leftarrow (\omega_t^*, J_t, \delta_t)$
      no_improve $\leftarrow$ no_improve $+ 1$
    end if
  end if
  $t \leftarrow t + 1$
  ▷ Issue-, hypothesis-, probe-, and verdict-level states are reset
  if no_improve $\geq K$ **then**
    break
  end if
end while
return $C^\star(\lambda^\star, \omega^\star, \phi^\star) \leftarrow C_t(\lambda_t, \omega_t^*, \phi_t)$

STAGE II: FIXED-STRUCTURE CALIBRATION AND ISSUE ROUTING

For the current simulator C_t with fixed structure λ_t, the **Simulation Execution Agent (SEA)** calibrates its numerical parameters according to ϕ_t:

$$(\omega_t^*, J_t, \delta_t) \leftarrow \mathcal{A}_{\mathrm{SEA}}(C_t, D, \phi_t),$$

where ω_t^* denotes the calibrated parameter state, J_t is the task-specific fidelity objective, and δ_t contains execution evidence such as residuals, logs, traces, and task-specific diagnostic statistics.

The resulting evidence is then routed according to the nature of the detected issue:

$$\mathcal{I}_t = \mathcal{I}_t^{\mathrm{dir}} \cup \mathcal{I}_t^{\mathrm{str}}.$$

Issues in $\mathcal{I}_t^{\mathrm{dir}}$ have causes that can be established directly from executable evidence, such as invalid data flow, disconnected code paths, interface violations, or runtime/implementation failures. These issues can be converted directly into localized repair directives. In contrast, issues in $\mathcal{I}_t^{\mathrm{str}}$ admit multiple plausible structural explanations and therefore enter the falsification-guided diagnosis procedure.

This routing is important because `FASim` does not require every simulator defect to undergo hypothesis testing. Executable falsification is reserved for structurally ambiguous diagnoses, while directly established implementation failures can be repaired without unnecessary probing.

STAGE III: FALSIFICATION-GUIDED STRUCTURAL DIAGNOSIS

For each ambiguous structural issue $I_{t,j} \in \mathcal{I}_t^{\mathrm{str}}$, the **Hypothesis Agent** generates competing explanations:

$$\mathcal{H}_{t,j} = \mathcal{A}_{\mathrm{H}}(I_{t,j}, \delta_t, B, C_t) = \{h_{t,j}^{(1)}, \ldots, h_{t,j}^{(K_j)}\}.$$

These hypotheses explain the same observed residual through alternative structural causes. Each candidate must imply an observable consequence under controlled simulator execution; otherwise it cannot justify persistent structural repair.

A separate **Falsification Agent** converts the hypothesis set into a frozen test plan,

$$\Pi_{t,j} = \mathcal{A}_{\mathrm{F}}(\mathcal{H}_{t,j}, C_t) = \left(p_{t,j}, \{s_{t,j}^{(k)}, f_{t,j}^{(k)}\}_{k=1}^{K_j}, \rho_{t,j}\right),$$

where $p_{t,j}$ is a counterfactual probe, $s_{t,j}^{(k)}$ is the predicted observable signature for hypothesis $h_{t,j}^{(k)}$, $f_{t,j}^{(k)}$ specifies its falsification condition, and $\rho_{t,j}$ selects the evidence-collection regime. The probe, signatures, and falsification conditions are fixed before the execution outcome is exposed.

The frozen probe is then executed on the current calibrated simulator. Directly observable signatures are evaluated from controlled execution traces, while stochastic or borderline responses use repeated matched-seed executions. Each hypothesis receives one of three verdicts,

$$v_{t,j}^{(k)} \in \{\textsc{Falsified}, \textsc{Survives}, \textsc{Inconclusive}\}.$$

A structural diagnosis is considered actionable only when exactly one hypothesis survives and all competing alternatives are falsified. We denote the resulting provisionally supported diagnosis by $d_{t,j}$; otherwise $d_{t,j} = \varnothing$. Thus, FALSIFIED hypotheses are rejected, while INCONCLUSIVE hypotheses are withheld because the available executable evidence is insufficient to justify repair.

The complete set of falsification-supported diagnoses is

$$\mathcal{D}_t^{\mathrm{fal}} = \{d_{t,j} \mid d_{t,j} \neq \varnothing\}.$$

Together with the directly verified issues,

$$\mathcal{D}_t = \mathcal{D}_t^{\mathrm{dir}} \cup \mathcal{D}_t^{\mathrm{fal}},$$

this forms the actionable repair set for the current iteration.

STAGE IV: DIAGNOSIS-GATED REPAIR AND RECALIBRATION

The **Code Generation Agent** receives the current retained simulator C_t, blueprint B, execution evidence δ_t, and actionable repair set $\mathcal{D}_t$, and produces a minimally modified candidate simulator:

$$(\tilde{\lambda}_{t+1}, \tilde{\phi}_{t+1}, \tilde{C}_{t+1}) \leftarrow \mathcal{A}_{\mathrm{CGA}}(B, C_t, \mathcal{D}_t, \delta_t).$$

Repairs are persistent only at this stage. Temporary probe interventions or instrumentation used during falsification are discarded after evidence collection and are never themselves committed as structural changes.

Because a structural edit may introduce, remove, or redefine calibratable quantities, the candidate simulator is recalibrated using the same task-level calibration backend:

$$(\tilde{\omega}^*_{t+1}, \tilde{J}_{t+1}, \tilde{\delta}_{t+1}) \leftarrow \mathcal{A}_{\mathrm{SEA}}(\tilde{C}_{t+1}, D, \tilde{\phi}_{t+1}).$$

Falsification and fidelity play different roles in this stage. Falsification determines whether an ambiguous structural explanation is sufficiently supported to justify an edit; the fidelity objective J evaluates the simulator obtained after that edit and recalibration. Improved fidelity therefore supports retaining a candidate simulator, but does not retrospectively establish that a structural hypothesis is a unique real-world causal explanation.

The candidate is retained only when it improves the current fidelity objective:

$$(C_{t+1}, J_{t+1}) = \begin{cases} (\tilde{C}_{t+1}, \tilde{J}_{t+1}), & \tilde{J}_{t+1} < J_t - \epsilon_{\mathrm{imp}}, \\ (C_t, J_t), & \text{otherwise.} \end{cases}$$

When the candidate is retained, its structure, calibration schema, calibrated parameters, and execution evidence become the state for the next outer iteration. Otherwise, the previous retained simulator remains unchanged.

DIAGNOSTIC-STATE RESET AND STOPPING

`FASim` does not maintain persistent issue or hypothesis identities across outer iterations. After each retain/reject decision, the next iteration re-executes the retained simulator and reconstructs issues from its current evidence. Consequently, optimization state persists through the retained simulator, whereas issue-, hypothesis-, probe-, and verdict-level diagnostic state is reset.

The outer loop terminates when its stopping criterion is reached. In the reported experiments, we use a maximum of six outer iterations (iterations 0–5); targeted replay experiments reported separately are not counted as additional refinement iterations. The controller can additionally apply a non-improvement patience criterion when early stopping is enabled. Upon termination, `FASim` returns the retained simulator

$$C^\star = C(\lambda^\star, \omega^\star, \phi^\star).$$

WORKFLOW SUMMARY

Algorithm 1 implements the following refinement logic:

1. **Construct:** build the unified schema and mechanistic blueprint, then instantiate the initial executable simulator.
2. **Calibrate and observe:** fit numerical parameters under the fixed current structure and collect execution evidence.
3. **Route issues:** send directly verifiable implementation failures to localized repair and structurally ambiguous failures to falsification-guided diagnosis.
4. **Hypothesize and falsify:** generate competing structural explanations, freeze their predicted implications, execute controlled probes, and admit only uniquely surviving diagnoses.
5. **Repair and recalibrate:** combine direct issues with falsification-supported diagnoses, generate a minimally modified candidate simulator, and recalibrate it under the same task-level backend.

Table 4: Baseline-specific reproduction protocol. All methods use matched task evidence, evaluation conditions, seeds, and resource caps; adaptations are restricted to interface compatibility unless otherwise stated.

Method	Shared inputs and conditions	Native components preserved	Adaptation boundary
SRR	Task description, blueprint, simulator code, execution evidence, splits, seeds, and matched calibration/refinement budgets.	Structured reflection, diagnosis selection, and iterative code repair.	No frozen counterfactual signatures or executable falsification.
G-SIM	Task/data inputs, preprocessing, splits, seeds, metrics, and matched resource caps.	Native structure-generation/refinement and calibration logic.	Common executable interface, logging, and output schema only.
AI Scientist-v2	Task goal, data evidence, splits, seeds, and final metrics.	Native research-code-generation, execution, and search workflow.	Input/output adapters only; no `FASim` diagnosis or repair machinery.
YuLan-OneSim	Task evidence, agent/network information, splits, seeds, and final metrics.	Native workflow/event-agent runtime and control flow.	Profile/runtime adapters only; no `FASim` diagnosis or calibration feedback.

6. **Retain or reject:** keep the candidate only when it improves simulator fidelity; otherwise retain the previous simulator and reconstruct diagnostic state in the next iteration.

Through this routed workflow, `FASim` separates numerical fitting from structural diagnosis and, within structural refinement, distinguishes directly established failures from ambiguous explanations that require executable challenge. The resulting *falsify-before-repair* design prevents linguistic plausibility alone from authorizing persistent simulator edits while preserving an efficient direct path for implementation issues that are already supported by executable evidence.

C Experimental Setup and Resource Analysis

Fairness and reproducibility. As shown in Table 4, to ensure a fair comparison, we standardize the external experimental conditions across `FASim` and all reproduced baselines. Within each benchmark, all methods use the same task-level evidence, data preprocessing, train/validation/test splits, random seeds, final evaluation metrics, and reporting protocol. We additionally enforce matched resource caps on LLM calls, simulator executions, parameter-calibration trials, structural-refinement iterations, and wall-clock/compute budget whenever the corresponding resource is used by the method. If a native baseline cannot consume a particular resource in the same way as `FASim`, the unused budget is not reallocated to additional reflection, tuning, or manual intervention. This prevents performance differences from arising simply from a larger optimization or test-time feedback budget.

Baseline-specific reproduction protocol. We preserve each baseline's intended methodological workflow rather than embedding all methods inside the `FASim` reasoning stack. G-SIM is reproduced through the common executable simulator interface while retaining its own structure-generation/refinement and calibration logic. AI Scientist-v2 and YuLan-OneSim are executed through their native research-code-generation and workflow/runtime abstractions, respectively, with lightweight adapters used only to expose the same task evidence, data splits, inputs, outputs, seeds, and evaluation interface. No `FASim`-specific diagnosis prompts, falsification evidence, repair directives, or calibration feedback are injected into these baselines. SRR is designed as the closest diagnosis-focused comparison: it receives the same task description, persisted blueprint, current simulator code, execution evidence, random seeds, calibration budget, and refinement budget as `FASim`, but resolves competing structural explanations through structured LLM reflection rather than frozen counterfactual predictions and executable falsification.

Micro-to-macro evaluation protocol. Across all tasks, the simulator instantiates **micro-entities** (e.g., *Modeled Residents*) with explicit micro-level states, interactions, and decision rules. Performance is assessed by whether their executions induce the correct **macro-level aggregates** under the task-defined aggregation operators and metrics, consistent with the main-text experimental settings. For final cross-method reporting, all methods are evaluated using the same fixed external evaluation implementation for each benchmark; method-internal diagnostic or monitoring code does not determine the values reported in the main comparison table.

Calibration and simulator-execution budgets. Parameter calibration is treated as fixed-structure infrastructure. For each task, the selected calibration backend is fixed throughout structural refinement, while the exposed parameterization may change when persistent code edits introduce, remove, or redefine calibratable quantities. Comparable methods use the same task-specific calibration-trial cap, stochastic-rollout count, random-seed set, and simulator-execution budget whenever applicable. Backend-specific settings—including parameter ranges or priors, BO/TuRBO configurations, SBI simulation budgets and posterior settings, and stochastic rollout counts—are fixed before the comparative evaluation and are not expanded for methods that fail to consume their full budget. Detailed backend definitions are provided in Appendix A.4.

LLM backbone and inference settings. All LLM-invoking components in `FASim` and the reproduced LLM-based baselines use **GPT-5.2** as the unified backbone model (main text, §4.1). This controls for backbone-capability differences when comparing algorithmic mechanisms. Where a baseline originally used another model, replacing it with GPT-5.2 typically strengthens rather than weakens the baseline, making the comparison conservative with respect to `FASim`. Comparable LLM roles use consistent inference settings, and all methods are subject to the same task-level LLM-call and token-budget caps.

Iteration budgets and stopping criteria. We preserve each baseline's native stopping logic while enforcing the same task-level outer-loop budget. In the reported `FASim` experiments, all formal refinement trajectories contain six outer iterations (iter 0–5); targeted replay cases used only for auxiliary analysis are not counted as additional refinement iterations. Candidate simulators are recalibrated and re-evaluated after structural repair, and no method receives extra iterations when its native workflow terminates early or fails to use the full allocation.

Wall-clock and token accounting. We report computational cost using both wall-clock time and total LLM token throughput. In our setup, generation of the initial executable simulator takes approximately $\sim$**20–25** minutes, while a structural-refinement iteration typically requires approximately **10–15** minutes, depending on simulator execution and calibration cost. Each `FASim` iteration invokes a fixed small set of LLM-based components, resulting in an overall task-level token budget on the order of 10^6 tokens. We log LLM calls, token usage, simulator executions, calibration evaluations, outer iterations, random seeds, and elapsed wall-clock time for each run so that budget matching can be audited directly.

LLM caching for cost control and reproducibility. To reduce redundant LLM calls during iterative refinement and calibration, we implement a two-level caching mechanism that treats each LLM invocation as a deterministic mapping from a fully specified input context to an output. The cache operates at (i) an *instance level* for reuse within a single run and (ii) a *global persistent level* that can be reused across repeated trials and ablations. Each invocation first queries the persistent cache, then the run-local cache; only on a cache miss is the LLM called, after which the result is written back to both levels.

Caching is particularly relevant to *User Modeling*, where the LLM produces discrete ratings or scores that are subsequently reused across parameter-calibration trials. Reusing an identical cached response avoids repeated LLM calls without changing the calibration objective, execution evidence, or evaluation protocol. Cache keys encode all factors that can affect an output, including the input content, conditioning attributes, model identifier, and decoding configuration, making cached results auditable and preventing reuse across semantically different contexts.

The persistent cache is not cleared between calibration/search trials when the fully specified key is unchanged. This reduces redundant token consumption while preserving the experimental protocol.

At the end of each run, the persistent cache and per-run resource logs are serialized for reproducibility and cost accounting.

D SUPPLEMENTARY EXPERIMENTS

D.1 END-TO-END FALSIFICATION CASE STUDY: TRIP-DISTANCE STRUCTURAL REPAIR

To complement the aggregate process statistics in Section 4.3, we examine one complete falsification-guided repair from the formal *Traj. N→A* refinement trajectory. This is an ordinary iter 2→iter 3 transition from the main optimization run rather than a targeted replay. The case is useful because it exposes the full decision chain from an ambiguous macro-level residual to a retained simulator update: issue detection, competing hypotheses, a frozen counterfactual probe, hypothesis-level verdicts, diagnosis gating, structural repair, recalibration, and subsequent fidelity evaluation.

Ambiguous residual rather than an immediate repair target. At iter 2, the simulator exhibits a large trip-distance mismatch: trip-distance Wasserstein distance is 4.35, even though Top-K POI recall is already 1.00. The simulator therefore reaches the relevant set of frequently visited POIs, but does not reproduce their spatial sequencing and consecutive-trip distance profile. This distinction is important because the residual cannot be explained simply by missing the correct destinations. Several structural causes remain plausible, including insufficient distance sensitivity, preference-dominant routing, candidate restrictions, initialization effects, transition errors, resident heterogeneity, and accumulated route-ordering effects. The issue is therefore classified as *structurally ambiguous* and is not allowed to trigger an immediate code edit.

Competing explanations. The Hypothesis Agent narrows this ambiguity to two explanations that make different predictions under a common intervention. Hypothesis H_1 attributes the mismatch to an active but insufficiently sensitive distance-decay pathway: geographic distance affects POI choice, but the existing decay is too weak to distinguish nearby from distant candidates strongly enough. Hypothesis H_2 instead attributes the mismatch to routing: the spatial pathway exists, but is largely suppressed by the preference branch, so changing the distance signal should have little effect on realized trajectories. The key distinction is therefore not whether a spatial term exists in the program, but whether that term materially controls the selected POIs.

Frozen matched-seed falsification. The Falsification Agent constructs a controlled probe that temporarily removes the spatial distance penalty by setting runtime candidate distances to zero. Under H_1, removing an influential distance penalty should make farther POIs easier to select and therefore *increase* mean trip distance. Under H_2, if preference routing already suppresses the spatial pathway, the same intervention should cause *no meaningful change*. The probe plan is frozen before execution, including the observable, predicted directions, falsification conditions, five matched random seeds, and the stability criterion. A verdict is considered stable only when at least four of the five matched-seed pairs receive the same effect classification; the meaningful-effect threshold is computed by the workflow controller from baseline variability rather than selected after observing the probe outcome.

The experiment executes five baseline runs and five matched probe runs. Table 5 reports the resulting mean trip distances. The probe increases mean trip distance for all five seed pairs, with an average paired effect of $+0.34$ km, exceeding the precomputed meaningful-effect threshold of 0.12 km in every pair. Instrumentation additionally verifies that the intended distance signal is removed while the preference–distance mixture, distance-scale parameter, data split, calibrated parameters, and simulator configuration remain unchanged. Thus, the observed response is attributable to the controlled intervention on the spatial pathway rather than to a simultaneous change elsewhere in the simulator.

From verdict to an admissible repair. The stable increase is consistent with H_1 and directly contradicts the frozen no-change prediction of H_2. The resulting verdicts are therefore $H_1 =$ SURVIVES and $H_2 =$ FALSIFIED. Because exactly one hypothesis survives and its competitor is falsified, the strict diagnosis gate is satisfied and only H_1 is forwarded as a falsification-supported diagnosis. This should not be interpreted as proving that H_1 is the unique real-world causal explanation; rather, H_1

Table 5: Matched-seed evidence for the *Traj. N→A* trip-distance falsification case. The probe temporarily removes the runtime distance penalty; positive effects indicate longer simulated trips.

Seed	Baseline	Probe	Paired effect
1103	10.88	11.24	+0.36
2281	10.85	11.22	+0.36
3469	10.86	11.16	+0.30
4657	10.76	11.11	+0.35
5843	10.98	11.28	+0.30
Mean	–	–	**+0.34**

Table 6: Effect of the falsification-supported repair from iter 2 to iter 3.

Metric	Iter 2	Iter 3	Change
Overall objective	0.71	0.60	-15.4%
Trip-distance WD	4.35	1.24	-71.6%
Stop-count mean error	0.63	0.39	-37.6%
Top-K POI recall	1.00	1.00	unchanged

is the only candidate among the tested explanations that remains consistent with the pre-registered executable evidence.

The repair stage then combines this diagnosis with truth-side directional evidence. The simulated mean trip distance is 10.99 km, compared with 6.64 km in the 2021 target data, indicating that the current simulator produces trips that are systematically too long. The resulting structural repair therefore strengthens the existing distance-decay pathway, reducing its base scale from 50 km to 10 km so that nearby POIs receive greater relative weight. Importantly, the falsification result constrains *which mechanism* should be modified, while the truth-side discrepancy determines the direction of that modification.

The same iteration also illustrates that falsification support for a pathway does not authorize arbitrary edits to that pathway. A separate diagnosis involving inter-event gaps established that the timing pathway was behaviorally consequential, but an automatically generated 120-minute cap was rejected because the probe did not identify that threshold and the target data retained a substantial long-gap tail. Thus, `FASim` distinguishes evidence that a mechanism matters from evidence that a particular numerical or structural modification is justified.

Repair outcome. After the supported spatial repair is applied and the candidate simulator is recalibrated, trip-distance Wasserstein distance decreases by 71.6%, while the overall fidelity objective improves by 15.4%. Top-K POI recall remains at 1.00, showing that the spatial improvement does not come from sacrificing destination coverage. Some non-target metrics worsen slightly, so the result is not a claim of uniform improvement across every behavioral dimension; rather, the repair strongly improves the residual it was designed to address and yields a net improvement under the task-level objective. The outer-loop controller consequently retains iter 3 as the new best simulator at that point in the trajectory.

Interpretation. This case illustrates the intended role of falsification in `FASim`. The framework does not simply generate a plausible explanation from a residual and immediately convert it into code. Instead, competing explanations are required to make distinct executable predictions, the test is frozen before observing the outcome, unsupported explanations are eliminated, and only the surviving diagnosis is allowed to constrain repair. The subsequent fidelity improvement then evaluates the *outcome* of that repair rather than retrospectively serving as evidence for the diagnosis itself. In this sense, falsification functions both as a mechanism-selection step and as a repair gate: it narrows the set of admissible structural edits before the simulator is modified.

D.2 Controlled Recovery of Known Structural Defects

The process analysis in Section 4.3 evaluates falsification during ordinary refinement, where the true structural cause of a residual is generally unknown. We therefore complement it with a controlled setting in which structural ground truth is available by construction. Starting from a runnable Mask

Table 7: Controlled structural defects used for ground-truth recovery evaluation. Each run injects exactly one known defect into the Mask Adoption simulator.

ID	Injected defect	Known structural error
D1	Campaign activation disconnected	The intervention channel is disconnected and no longer affects downstream updating.
D2	Information has no behavioral effect	Information exposure is computed, but no longer influences the mask-decision rule.
D3	Work/community influence removed	Work and community social influence are removed, leaving only family influence.
D4	Social-influence layers miswired	Work and community influence channels are exchanged, preserving execution but corrupting layer semantics.
D5	Behavioral persistence removed	Current mask behavior no longer depends on the individual's previous-day state.
D6	Individual persistence collapsed to population level	Individual previous-state dependence is replaced by population-level persistence.
D7	High-risk subgroup response inverted	The risk-dependent response is reversed for high-risk agents while the rest of the simulator remains unchanged.

Adoption simulator, we inject exactly one predefined structural defect per run. Because both the corrupted mechanism and the corresponding reference structure are known, this setting allows us to evaluate whether a method merely improves aggregate fit or actually identifies and restores the intended structural dependency.

Controlled defect construction. We consider seven defects spanning disconnected mechanisms and semantic structural errors, summarized in Table 7. D1 disconnects the campaign pathway, D2 preserves information exposure but removes its behavioral effect, D3 removes work/community social influence, and D5 removes individual behavioral persistence. The remaining defects preserve executable pathways but corrupt their semantics: D4 swaps work and community influence channels, D6 replaces individual previous-state dependence with population-level persistence, and D7 inverts the response of the high-risk subgroup. These defects are deliberately chosen so that some corrupted simulators can still achieve plausible aggregate fit after recalibration even though the underlying structure is known to be wrong.

Falsification protocol. Each D1–D7 condition is processed using the same falsification design as the main method. After the injected defect produces execution residuals, the issue is routed as structurally ambiguous and passed to the Hypothesis Agent, which proposes competing structural explanations. The Falsification Agent then constructs a frozen matched-seed probe with pre-specified observable signatures and falsification conditions. Baseline and probe executions use the same random seeds, and the resulting hypotheses are assigned FALSIFIED, SURVIVES, or INCONCLUSIVE verdicts under the frozen plan. A repair is permitted only when exactly one hypothesis survives and all competing explanations are falsified. Thus, recovery is evaluated under the same *falsify-before-repair* gate used in the main experiments rather than by directly exposing the injected defect label to the repair stage.

Baselines. We compare `FASim` with three baselines under matched task evidence, temporal splits, seeds, rollout counts, calibration budgets, and structural-refinement budgets. *Calibration only* uses the defect-injected simulator and applies the same fixed-structure parameter calibration budget, but performs no structural diagnosis or code repair; it tests whether numerical calibration can compensate for or conceal a known wrong structure. *G-SIM* performs iterative simulator-structure generation and parameter calibration using its native generate–execute–revise procedure, but does not use `FASim`'s competing-hypothesis construction, frozen counterfactual probes, or strict falsification gate. *Structured Reflective Repair (SRR)* receives the same task description, persisted blueprint, current simulator code, and execution evidence as `FASim`, and performs structured multi-step reflection to select a structural diagnosis and generate a repair; however, competing explanations are resolved through LLM reflection rather than executable falsification. SRR therefore controls for full task

Table 8: Controlled recovery of the seven injected structural defects.

Method	Identified	Diag. repair	Exact/equiv.	Stable
Calibration only	–	–	0/7	–
G-SIM	6/7	6/7	5–6/7	6/7
SRR	7/7 final	delayed D1/D2	6/7	weak D3–D6
FASim	**7/7**	**7/7**	**7/7**	**7/7**

Table 9: Counterexamples to aggregate score chasing. Lower validation loss does not necessarily imply correct structure.

Defect	Corrupted loss	FASim loss	Repair retained
D1	0.1047	0.1449	Yes
D3	0.0922	0.0942	Yes
D4	0.0924	0.1242	Yes
D6	0.0928	0.1004	Yes

context, blueprint access, code access, and iterative reasoning while removing the central *falsify-before-repair* mechanism.

Evaluation criteria. A repair is counted as correct only when it removes the injected defect through either the exact reference edit or a mechanism-equivalent correction that restores the same intended dependency. We report four structural criteria. *Defect identified* records whether the method localizes the injected structural failure; *diagnosis-driven repair* records whether the corresponding diagnosis is actually used to modify the simulator; *exact/equiv. repair* requires recovery of the reference mechanism rather than merely improved aggregate scores; and *stable repair* requires that the recovered mechanism remain present in the subsequently retained simulator. These criteria make structural recovery independently observable from the downstream fidelity objective.

Ground-truth structural recovery. Table 8 summarizes the results. FASim identifies all seven injected defects and obtains $7/7$ diagnosis-driven, exact or mechanism-equivalent, and stably retained repairs. G-SIM identifies six of seven defects and achieves five to six mechanism-equivalent repairs, while SRR eventually identifies all seven but shows delayed recovery for D1/D2 and weaker retention for several later cases. Calibration only never removes the injected defect and therefore achieves $0/7$ structural recovery. Because the reference mechanism is known before evaluation, these results provide direct evidence that FASim's diagnosis-and-repair loop restores the intended structure rather than merely finding a lower-loss executable program.

Structural recovery is not equivalent to score chasing. The controlled setting also exposes cases where aggregate validation loss favors the known-wrong simulator. As shown in Table 9, for D1, D3, D4, and D6 the calibrated corrupted simulator attains a lower validation loss than the structurally repaired simulator while still retaining the injected defect. FASim nevertheless preserves the reference-consistent repair, restoring the campaign pathway in D1, multiplex social influence in D3, the correct work/community semantics in D4, and individual-level persistence in D6. A procedure that simply selected the lowest-loss program would therefore prefer the known-wrong structure in all four cases.

Interpretation. This controlled experiment provides a stronger correctness signal than ordinary refinement because the structural reference is known by construction. In the natural tasks, a surviving hypothesis is only provisionally supported by executable evidence; here, the injected defect specifies what the simulator is known to have lost or miswired, and the recovered program can be compared directly with that reference. The $7/7$ exact or mechanism-equivalent and stably retained repairs show that the current Hypothesis Agent, frozen matched-seed falsification plan, strict diagnosis gate, and downstream repair stage can jointly recover known structural dependencies across both disconnected and semantically corrupted mechanisms.

At the same time, this result remains intentionally bounded. It demonstrates recovery within a specified simulator family where the admissible structural space and reference design are known; it does not imply that macro-level residuals uniquely identify arbitrary real-world causal mechanisms.

Table 10: Method-blinded evaluation of diagnosis and repair quality on Personal Mobility refinement artifacts. Scores range from 0 to 2; higher is better.

Dimension	FASim	SRR	Difference
Fault localization	**1.706**	1.667	+0.039
Code localization	**1.804**	1.756	+0.048
Mechanism linkage	**1.686**	1.644	+0.042
Post-edit effectiveness	**0.725**	0.600	+0.125
Overall average	**1.480**	1.417	+0.063

Together with the end-to-end case study in Appendix D.1, the experiment supports the narrower claim that executable falsification can both eliminate unsupported explanations and guide recovery of the intended simulator structure when that structure is externally verifiable.

D.3 Blinded Evaluation of Diagnosis and Repair Quality

Final simulator fidelity provides only indirect evidence about the quality of the intermediate diagnosis process. We therefore complement the end-to-end results with a method-blinded evaluation of diagnosis and repair artifacts from the Personal Mobility refinement trajectories. The goal is not to establish a unique causal ground truth, which is unavailable in these natural refinement settings, but to assess whether the generated diagnoses are localized, mechanism-linked, and translated into effective edits.

We compare FASim against Structured Reflective Repair (SRR), the closest diagnosis-focused baseline in the main experiments. SRR receives the same task specification, persisted blueprint, simulator code, raw execution evidence, calibration budget, repair budget, seeds, and execution constraints as FASim, but resolves competing structural explanations through structured LLM reflection rather than frozen executable falsification. This comparison therefore controls for full task context, blueprint access, code access, and iterative reasoning while isolating the effect of the falsification-guided diagnosis process.

Evaluation protocol. We evaluate all available refinement transitions from the three Personal Mobility settings: 17 transitions for FASim (5 N→N, 6 N→A, and 6 A→A) and 15 transitions for SRR (5 per setting). Each artifact is independently evaluated by three LLM judges—GPT-5.6, Gemini-3.1-Pro, and Claude Sonnet 5—using the same fixed rubric. Method-identifying information, including method names, internal agent terminology, method-specific fields, and file or folder identifiers, is removed before evaluation. Each transition is scored independently, without access to earlier or later judgments and without communication among judges.

The rubric uses a 0–2 scale along four dimensions: *fault localization*, which measures whether the diagnosis identifies a mechanism plausibly responsible for the observed residual; *code localization*, which measures whether the implicated implementation region is precise and actionable; *mechanism linkage*, which measures whether the proposed repair directly implements the diagnosed structural correction; and *post-edit effectiveness*, which measures whether the targeted residual improves without major regressions. Table 10 reports averages across all evaluated transitions and all three judges.

FASim receives higher average scores on all four dimensions. The localization and mechanism-linkage gains are modest, which is unsurprising because SRR already receives the complete blueprint, code, and execution context and performs structured multi-step reflection. The clearest difference appears in post-edit effectiveness, where FASim improves from 0.600 to 0.725. This suggests that the main benefit of falsification-guided diagnosis is not simply producing more plausible explanations, but more reliably constraining which explanations are allowed to become persistent structural edits.

Cross-judge consistency and scope. The three judges exhibit broadly consistent aggregate trends. For FASim, pairwise exact agreement ranges from approximately 51% to 73% across dimensions, while 92%–98% of paired ratings differ by no more than one point on the 0–2 scale. For SRR, exact agreement ranges from approximately 67% to 73%, with 93%–100% of paired ratings differing by at most one point. The judges therefore disagree mainly on severity—especially between scores 1 and 2—rather than frequently producing opposite assessments.

Table 11: Backbone sensitivity on the Mask Adoption task under the same FASim refinement and calibration protocol. Lower RMSE is better.

Backbone	RMSE ↓	Observation
GPT-5.2	0.070 ± 0.009	Main reported backbone
Qwen3-Next-80B-A3B-Instruct	0.0495	Full loop completed successfully
Llama-3.3-70B-Instruct	0.7363	Full loop completed, but weak diagnosis/repair quality

This analysis should be interpreted as a blinded robustness check of diagnosis quality rather than as ground-truth causal validation. The natural mobility trajectories do not provide a unique reference mechanism, and the judges themselves are LLMs rather than human experts. The controlled D1–D7 experiment in Appendix D.2 remains the primary ground-truth test of structural recovery. Taken together, the two analyses are complementary: the controlled study evaluates whether FASim recovers a known reference structure, while the blinded evaluation assesses whether its diagnoses and repairs remain more localized, mechanism-linked, and effective in ordinary refinement settings where such ground truth is unavailable.

D.4 Backbone Sensitivity

The main experiments use GPT-5.2 as the common LLM backbone in order to control for model-capability differences across FASim and the reproduced LLM-based baselines. We additionally evaluate whether the falsification-guided refinement pipeline is specific to this backbone by rerunning the Mask Adoption task with two alternative model families: Qwen3-Next-80B-A3B-Instruct and Llama-3.3-70B-Instruct. All other components, including the simulator construction pipeline, task specification, calibration backend, execution protocol, and refinement budget, are held fixed.

Table 11 reports the resulting Mask Adoption RMSE. Qwen3-Next-80B-A3B-Instruct successfully completes the full diagnosis–repair loop and achieves performance comparable to, and in this run better than, GPT-5.2. This result indicates that FASim's operation is not tied to a single proprietary backbone. In contrast, Llama-3.3-70B-Instruct completes the pipeline but produces substantially weaker final fidelity. Inspection of the refinement artifacts shows that the weaker backbone more often generates coarse or repetitive structural explanations and less actionable repair decisions, whereas the stronger backbones more reliably localize concrete mechanism failures and translate them into executable updates.

The comparison therefore supports a qualified conclusion. Falsification-guided structural diagnosis is not exclusive to GPT-5.2, as demonstrated by the successful Qwen run, but its effectiveness remains sensitive to the reasoning and code-understanding capability of the underlying LLM. This dependence is expected because the Hypothesis Agent and Falsification Agent must formulate discriminative structural explanations and executable tests before the controller can apply the deterministic falsification gate. The result should therefore be interpreted as evidence of cross-backbone portability rather than backbone invariance.

E An Example of Blueprint

In this section, we provide an illustrative example of a complete blueprint generated by the **Data Analysis Agent** for the **Personal Mobility Generation** task, focusing on **generalizing from normal periods to a pandemic-disrupted scenario** (Traj. N→A (OOD)). This structured JSON object contains all the necessary information, from data analysis summaries to simulation design parameters, required by downstream agents to construct and execute the simulator.

Generated Blueprint: N2A Mobility Trajectory Simulation (Online OOD)


```
{
  "title": "Simulation Task",
  "description": "Design a social simulator to generate daily
      mobility trajectories of urban residents under an out-of
```

```
      -distribution (OOD) setting (N2A: Normal-to-Abnormal). A
       trajectory is an ordered sequence of (location, time)
      visits: {(l0,t0),(l1,t1),...,(ln,tn)}. In this task, the
       simulator MUST be trained/fitted only on normal-period
      records (2019-2020), while model selection / calibration
       MUST use abnormal-period targets (2021) as validation.
      Final performance MUST be reported on a held-out 2021
      test split.",
  "data_folder": "data_fitting/llmob_data/",
  "data_files": {
    "1921Y.json": "Resident daily trajectories across multiple
        years. For THIS TASK (N2A Online OOD), split by date
       into three disjoint subsets:\n(A) TRAIN_NORMAL: keep
       ONLY records where date is within [2019-01-01,
       2020-12-31]. Used for training/fitting priors/Pattern/
       Persona ONLY.\n(B) V_CALIB_ABNORMAL: keep ONLY records
        where date is within [2021-01-01, 2021-12-31] AND
       assigned to the 2021-calib split. Used for calibration
       /model selection (best_params, early stop) ONLY.\n(C)
       V_TEST_ABNORMAL: keep ONLY records where date is
       within [2021-01-01, 2021-12-31] AND assigned to the
       2021-test split. Used ONLY once for final report.",
    "poi_category_192021_longitude_latitude.json": "Maps each
       full POI name (e.g., \"Rest Area#1061\") to its
       latitude/longitude. Use it to get coordinates for each
        visited stop (for both 2019-2020 fitting and 2021
       evaluation).",
    "catto.json": "Maps base POI name (suffix removed; e.g.,
       \"Rest Area\") to a semantic category (e.g., \"Travel
       & Transport\"). Use it to compute category-based
       metrics."
  },
  "data_analysis_result": {
    "overall_simulation_design": {
      "objective": "Generate realistic daily mobility
         trajectories (ordered sequences of (POI, time)
         visits) for urban residents under an out-of-
         distribution Normal-to-Abnormal (N2A) setting: fit
         baseline behavior using only 2019\u20132020 records,
          tune/calibrate abnormal-shift parameters using a
         2021 validation split, and report final performance
         on a held-out 2021 test split.",
      "initialization": {
        "description": "Load and parse 1921Y.json into per-
           user, per-day sequences; split by year into normal
            (2019\u20132020) vs abnormal (2021). Build POI
           catalog with lat/lon from
           poi_category_192021_longitude_latitude.json and
           map fine categories to super-categories using
           catto.json. Initialize each Resident with baseline
            (normal-period) learned distributions (time-of-
           day, chain length, category/POI transitions,
           anchor POIs). Initialize global abnormal-shift
           parameters (to be calibrated) that perturb
           baseline distributions when simulating 2021.",
        "source_type": "mixed",
        "data_reference": {
          "file": "1921Y.json",
          "fields": [
            "top-level keys (user IDs)",
            "top-level values (daily activity strings
               containing dates, locations, times)"
```

```
            ],
            "derivation": "Parse each string 'Activities at YYYY
                -MM-DD: ...' into date and ordered visits of (
                Category#POI_ID, HH:MM:SS); year determines
                normal (2019\u20132020) vs abnormal (2021). Join
                 POI lat/lon by matching POI identifier strings
                to poi_category_192021_longitude_latitude.json
                entries; map Category -> super-category via
                catto.json when available."
          }
        },
        "execution": "1) Fit baseline per-user and global
            mobility models on 2019\u20132020 only (e.g.,
            distributions over start time, stop count, inter-
            visit time gaps, super-category transitions, POI
            choice kernels conditioned on category and distance)
            . 2) Define abnormal-shift mechanisms with free
            parameters (e.g., category-mix reweighting,
            increased/decreased mobility radius, increased
            infrastructure/transit propensity, time-shift of
            activities). 3) Calibrate abnormal-shift parameters
            by simulating 2021 validation days and optimizing an
             objective over evaluation metrics (trajectory
            similarity, distributional match) against 2021
            ground truth. 4) Freeze baseline + calibrated shift
            and evaluate on 2021 held-out test split. 5) Online
            OOD context window rule: For each 2021 target day,
            construct a 7-day context window using only days
            strictly earlier than the target day; this window
            may include earlier 2021 days and may backfill from
            late 2020; never use target-day or future days.",
        "outputs": [
          "calibrated_parameters",
          "evaluation_results_on_validation"
        ]
      },
      "scale_granularity": {
        "time_step": {
          "value": "minutes",
          "source_type": "mixed",
          "data_reference": {
            "file": "1921Y.json",
            "fields": [
              "visit time tokens HH:MM:SS within daily strings"
            ],
            "derivation": "Observed times are second-resolved;
                simulator can operate in minute granularity
                while outputting HH:MM:SS by rounding/stochastic
                 seconds jitter (designed)."
          }
        },
        "spatial_resolution": {
          "value": "POI",
          "source_type": "data_derived",
          "data_reference": {
            "file": "1921Y.json",
            "fields": [
              "place tokens formatted Category#POI_ID"
            ],
            "derivation": "Each visit location is a POI
                identifier string (Category#id)."
          }
```

```
      },
      "population_size": {
        "value": "number of unique user IDs in 1921Y.json (one
            agent per key)",
        "source_type": "data_derived",
        "data_reference": {
          "file": "1921Y.json",
          "fields": [
            "top-level keys"
          ],
          "derivation": "Count keys in the JSON object; each
              key defines one resident agent."
        }
      },
      "rationale": "Trajectories are day-level sequences with
          intra-day timestamps; minute-level simulation
          supports feasibility constraints via travel-time
          approximations while matching the dataset\u2019s
          timestamp structure. POI-level resolution is
          required because ground truth uses Category#POI_ID
          and POI lat/lon is available for distance-based
          plausibility."
    },
    "agent_archetypes": {
      "unit": "resident/user",
      "roles": [
        {
          "name": "Resident",
          "static_attributes": [
            {
              "name": "resident_id",
              "type": "string",
              "source_type": "data_derived",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "top-level key"
                ],
                "derivation": "Use JSON key as resident_id."
              }
            },
            {
              "name": "anchor_pois",
              "type": "vector",
              "source_type": "mixed",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "daily visit sequences (locations)"
                ],
                "derivation": "From 2019\u20132020 visits,
                    compute top-K frequent POIs; additionally
                    mark any POI whose category is 'Home' as
                    an anchor when present (data-derived),
                    otherwise rely on frequency-only heuristic
                     (designed fallback)."
              }
            },
            {
              "name": "baseline_start_time_dist",
              "type": "other",
              "source_type": "data_derived",
```

```
      "data_reference": {
        "file": "1921Y.json",
        "fields": [
          "first timestamp per day"
        ],
        "derivation": "For each 2019\u20132020 day,
            extract first visit time; fit a
            distribution per user (e.g., histogram/
            mixture)."
      }
    },
    {
      "name": "baseline_stop_count_dist",
      "type": "other",
      "source_type": "data_derived",
      "data_reference": {
        "file": "1921Y.json",
        "fields": [
          "number of visits per day"
        ],
        "derivation": "For each 2019\u20132020 day,
            count visits; fit a per-user distribution
            ."
      }
    },
    {
      "name": "baseline_inter_event_time_dist",
      "type": "other",
      "source_type": "data_derived",
      "data_reference": {
        "file": "1921Y.json",
        "fields": [
          "ordered visit times within day"
        ],
        "derivation": "Compute successive time gaps
            within 2019\u20132020 days; fit
            distribution (conditioned on previous/next
             category optionally)."
      }
    },
    {
      "name": "baseline_category_transition_model",
      "type": "other",
      "source_type": "mixed",
      "data_reference": {
        "file": "1921Y.json",
        "fields": [
          "ordered visit locations (Category#POI_ID)"
        ],
        "derivation": "Extract Category from each
            location token; optionally map to super-
            category via catto.json; fit Markov
            transition probabilities with smoothing at
             super-category level if category not
            found in catto.json."
      }
    },
    {
      "name": "baseline_poi_choice_model",
      "type": "other",
      "source_type": "mixed",
      "data_reference": {
```

```
                "file": "
                    poi_category_192021_longitude_latitude.
                    json",
                "fields": [
                  "POI records [lat, lon, 'Category#id']"
                ],
                "derivation": "Given chosen category, choose
                    POI via a combination of (a) empirical POI
                     popularity from 2019\u20132020 visits (
                    data-derived from 1921Y.json) and (b)
                    distance-decay from current POI using lat/
                    lon (data-derived). Weighting between (a)
                    and (b) is calibratable (mixed)."
              }
            }
          ],
          "dynamic_states": [
            {
              "name": "current_day",
              "type": "string",
              "update_rule": "Advance to next simulation date;
                   for calibration/evaluation, simulate only
                  dates in the selected 2021 validation/test
                  split; for generation, dates are sampled
                  from target set.",
              "source_type": "mixed",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "date prefix 'Activities at YYYY-MM-DD'"
                ],
                "derivation": "Use available dates to define
                    candidate simulation days (data-derived);
                    selecting which days to simulate per split
                     is designed by holdout plan."
              }
            },
            {
              "name": "current_poi",
              "type": "string",
              "update_rule": "Set to the most recently
                  generated POI in the trajectory sequence.",
              "source_type": "designed",
              "data_reference": null
            },
            {
              "name": "current_time",
              "type": "string",
              "update_rule": "Set to generated time-of-day;
                  increment by sampled inter-event gap subject
                   to feasibility constraints.",
              "source_type": "mixed",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "HH:MM:SS tokens"
                ],
                "derivation": "Time-of-day distributions fit
                    from 2019\u20132020; feasibility
                    adjustment (e.g., minimum travel time) is
                    designed using POI distances."
              }
```

```
            },
            {
              "name": "daily_plan",
              "type": "vector",
              "update_rule": "At day start, sample stop_count,
                   start_time, and a sequence of activity
                  categories; then instantiate POIs and times
                  sequentially.",
              "source_type": "mixed",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "per-day sequences of categories and times"
                ],
                "derivation": "Plan templates and
                    distributions learned from normal data;
                    abnormal adjustments applied via exogenous
                     shift parameters calibrated on 2021
                    validation."
              }
            },
            {
              "name": "visited_sequence",
              "type": "vector",
              "update_rule": "Append each generated (POI, time
                 ) as simulation progresses.",
              "source_type": "designed",
              "data_reference": null
            }
          ],
          "construction_from_data": "Create one Resident agent
               per user ID in 1921Y.json. Parse 2019\u20132020
               days to estimate baseline distributions and
              transition models. Build POI lookup table from
              poi_category_192021_longitude_latitude.json by
              indexing on the POI identifier string.
              Optionally map fine categories to super-
              categories using catto.json for smoothing and
              OOD robustness.",
          "update_from_data": "During calibration/evaluation,
              for a given 2021 day the agent generates a
              trajectory using baseline models plus abnormal-
              shift parameters. No online learning from 2021
              ground truth during simulation; only outer-loop
              calibration uses 2021 validation targets to
              adjust global shift parameters."
        },
        {
          "name": "CalibrationController",
          "static_attributes": [
            {
              "name": "search_strategy",
              "type": "enum",
              "source_type": "designed",
              "data_reference": null
            },
            {
              "name": "objective_weights",
              "type": "vector",
              "source_type": "designed",
              "data_reference": null
            }
```

```
          ],
          "dynamic_states": [
            {
              "name": "current_parameter_vector",
              "type": "vector",
              "update_rule": "Update via black-box
                  optimization (e.g., Bayesian optimization /
                  evolutionary search / random search) to
                  minimize validation loss computed against
                  2021 validation set.",
              "source_type": "designed",
              "data_reference": null
            },
            {
              "name": "best_parameter_vector",
              "type": "vector",
              "update_rule": "Replace when validation
                  objective improves.",
              "source_type": "designed",
              "data_reference": null
            },
            {
              "name": "experiment_log",
              "type": "vector",
              "update_rule": "Append each trial\u2019s params,
                   seed, metrics, and artifacts.",
              "source_type": "designed",
              "data_reference": null
            }
          ],
          "construction_from_data": "Controller reads parsed
              datasets and defines train/validation/test
              partitions per holdout plan. It does not exist
              in the raw data; it is introduced to meet the
              requirement of calibrating using 2021 validation
               while fitting baseline on 2019\u20132020.",
          "update_from_data": "Uses 2021 validation ground
              truth from 1921Y.json to compute objective;
              updates parameter vector based on optimization
              routine."
        }
      ]
    },
    "interaction_topology": {
      "topology": "platform-level",
      "layers": [
        {
          "name": "calibration_loop",
          "from_role": "CalibrationController",
          "to_role": "Resident",
          "edge_rule": "Controller broadcasts a candidate
              abnormal-shift parameter vector and simulation
              configuration (seed, day subset); Residents
              simulate trajectories for the requested days;
              outputs are returned for scoring.",
          "source_type": "designed",
          "data_reference": null
        },
        {
          "name": "shared_environment_lookup",
          "from_role": "Resident",
          "to_role": "Resident",
```

```
            "edge_rule": "No direct resident-to-resident
                influence; all residents share the same POI
                catalog, category taxonomy, and global shift
                parameters.",
            "source_type": "designed",
            "data_reference": null
          }
        ],
        "protocol": "Per calibration iteration: (1) Controller
            selects parameter vector \u03b8 and simulation seed(
            s). (2) For each resident/day in the calibration
            subset, residents generate trajectories using \u03b8
            . (3) Controller computes metrics vs 2021 ground
            truth and updates \u03b8. After tuning, run
            evaluation on held-out 2021 test split and report
            metrics."
      },
      "information_propagation": {
        "exists": false,
        "topology": "none (no explicit social diffusion in
            provided data)",
        "mechanism": "Residents do not exchange messages; any
            global changes are modeled via exogenous abnormal-
            shift parameters applied uniformly or stratified by
            resident clusters (optional, designed).",
        "source_type": "data_derived",
        "data_reference": {
          "file": "1921Y.json",
          "fields": [
            "user-indexed independent daily trajectories"
          ],
          "derivation": "Data contains per-user sequences
              without any explicit social links or interactions;
               therefore diffusion is not directly supported."
        }
      },
      "exogenous_signals": [
        {
          "name": "abnormal_period_shift",
          "effect_on_agents": "Applies when simulating dates in
              2021: reweights category/super-category choice,
              modifies stop-count distribution, shifts start
              times, adjusts travel-radius/distance-decay, and
              adjusts propensity to include infrastructure
              categories (e.g., Toll Booth/Tunnel/Rest Area/
              Platform) as intermediate stops.",
          "bounds": "Calibration bounds set to reasonable
              multiplicative/shift ranges around baseline (e.g.,
               category weight multipliers in [0.25, 4], time
              shifts in [-120,+120] minutes, distance-decay
              scale multipliers in [0.5, 2.5]).",
          "source_type": "mixed",
          "data_reference": {
            "file": "1921Y.json",
            "fields": [
              "2021 daily trajectories (locations and times)"
            ],
            "derivation": "Signal is designed as a parameterized
                 mechanism; its parameters are calibrated to
                match distributional properties of 2021
                trajectories."
          }
```

```
      },
      {
        "name": "day_type_proxy",
        "effect_on_agents": "Optional: adjust schedule
            templates for weekday/weekend using only date-
            derived heuristics; used to condition start time
            and stop-count distributions.",
        "bounds": "Binary {weekday, weekend} inferred from
            date; if timezone/calendar assumptions required,
            treat as heuristic with standard Gregorian
            calendar.",
        "source_type": "designed",
        "data_reference": null
      }
    ],
    "action_decision_policy": {
      "by_role": [
        {
          "role": "Resident",
          "inputs": [
            {
              "name": "
                  baseline_distributions_and_transition_models
                  ",
              "source_type": "data_derived",
              "data_reference": {
                "file": "1921Y.json",
                "fields": [
                  "2019\u20132020 parsed daily sequences"
                ],
                "derivation": "Fit per-user and global models
                    from normal period only."
              }
            },
            {
              "name": "poi_catalog_with_latlon",
              "source_type": "data_derived",
              "data_reference": {
                "file": "
                    poi_category_192021_longitude_latitude.
                    json",
                "fields": [
                  "category -> list of [lat, lon, poi_id]"
                ],
                "derivation": "Index POIs by poi_id and by
                    category; compute distances as needed."
              }
            },
            {
              "name": "category_to_supercategory_map",
              "source_type": "data_derived",
              "data_reference": {
                "file": "catto.json",
                "fields": [
                  "category -> super-category"
                ],
                "derivation": "Lookup for each fine category;
                    if missing, treat super-category as '
                    Unknown' (designed fallback)."
              }
            },
            {
```

```
            "name": "abnormal_period_shift_parameters",
            "source_type": "designed",
            "data_reference": null
          },
          {
            "name": "current_state (current_poi,
                current_time, visited_sequence)",
            "source_type": "designed",
            "data_reference": null
          }
        ],
        "policy_form": "Hierarchical generative policy per
            day: (1) Sample start time and stop count from
            baseline distributions perturbed by abnormal
            shift (if 2021). (2) Sample a sequence of
            activity categories using a Markov model (
            category or super-category) with optional time-
            of-day conditioning and abnormal reweighting.
            (3) For each category, sample a concrete POI
            using a mixture of (a) personal historical
            preference (2019\u20132020 POI visit frequency)
            and (b) spatial distance-decay from current POI
            using lat/lon; optionally enforce feasibility by
             ensuring time gaps exceed a minimum travel-time
             proxy derived from distance. (4) Output ordered
             (POI, time) visits.",
        "parameters": [
          "theta_cat_weight_multiplier_by_supercategory",
          "theta_stop_count_multiplier",
          "theta_start_time_shift_minutes",
          "theta_distance_decay_scale",
          "theta_preference_vs_distance_mixture",
          "theta_infrastructure_stop_bonus"
        ]
      },
      {
        "role": "CalibrationController",
        "inputs": [
          {
            "name": "simulated_trajectories",
            "source_type": "designed",
            "data_reference": null
          },
          {
            "name": "validation_ground_truth_2021",
            "source_type": "data_derived",
            "data_reference": {
              "file": "1921Y.json",
              "fields": [
                "2021 daily activity strings"
              ],
              "derivation": "Parse 2021 dates/visits for
                  validation subset."
            }
          }
        ],
        "policy_form": "Outer-loop calibration policy:
            propose \u03b8, run simulation on 2021
            validation subset, compute objective from
            metrics, update \u03b8 using selected black-box
            optimization strategy.",
        "parameters": [
```

```
            "optimizer_hyperparams",
            "metric_weights"
          ]
        }
      ]
    },
    "holdout_plan": {
      "method": "temporal_holdout",
      "time_ordering": {
        "source_type": "data_derived",
        "data_reference": {
          "file": "1921Y.json",
          "fields": [
            "date token in each daily record string: '
                Activities at YYYY-MM-DD'"
          ],
          "derivation": "Parse YYYY-MM-DD and sort
              chronologically; year used for regime split."
        }
      },
      "train_range": "Fit baseline only on dates with year in
          {2019, 2020}. Within 2021, do not use for baseline
          fitting.",
      "validation_range": "Within 2021 dates, use first 80% of
           dates in chronological order as validation for
          abnormal-shift calibration (per user or globally;
          implement globally by date ordering). (FIXED SPLIT
          RULE: per-user-by-date only) For each resident
          independently, sort that resident's 2021 day-records
           by date ascending; assign the first 80% of that
          resident's 2021 records to V_calib and the last 20%
          to V_test (deterministic, no global mixing). If a
          resident has fewer than 5 total 2021 records,
          exclude that resident entirely from calibration/test
           with explicit logging (default), or assign all
          their 2021 records to V_test only (allowed fallback)
           \u2014 but never include them in V_calib.",
      "notes": "Final test is the remaining last 20% of 2021
          dates chronologically. If some users have sparse
          2021 records, apply the 80/20 rule per-user to
          preserve personalization (designed choice); record
          which approach is used. Split strategy is fixed to
          per-user-by-date as specified above; do not use
          global-by-date splitting."
    },
    "simulation_evaluation": {
      "metrics": [ { "name": "DARD", "definition": "Daily
          Activity Rhythm Divergence: a symmetric divergence
          between simulated vs ground-truth daily activity-
          rhythm distributions over time-of-day bins (
          optionally conditioned on activity categories),
          capturing phase shifts and missing/extra peaks in
          the daily schedule." }, { "name": "STVD", "
          definition": "Spatio-Temporal Visit Distribution
          distance: a transportation-based distance (e.g.,
          Wasserstein/EMD) between simulated vs ground-truth
          empirical distributions of visits in joint (space,
          time) space, capturing coupled footprint errors such
           as correct locations at wrong times (and vice versa
          )." } ],
      "comparison_method": "Compute metrics on the 2021
          validation split for each calibration trial; select
```

```
            theta minimizing a weighted sum of metrics (
            objective). After calibration, compute the same
            metrics on the 2021 test split and report. Hard
            constraint: best_params selection and early stopping
             must use ONLY V_calib; V_test must be evaluated
            exactly once after selection and must not affect any
             choice (no tuning, no re-selection, no peeking)."
    },
    "calibratable_parameters": [
      {
        "name": "theta_cat_weight_multiplier_by_supercategory
            ",
        "range_bounds": "Multipliers per super-category in
            [0.25, 4.0]",
        "source": "constrained by modeling",
        "notes": "Reweights baseline category/super-category
            selection in 2021 to match abnormal mix."
      },
      {
        "name": "theta_stop_count_multiplier",
        "range_bounds": "[0.5, 1.8]",
        "source": "constrained by modeling",
        "notes": "Scales expected number of daily stops in
            2021 vs baseline."
      },
      {
        "name": "theta_start_time_shift_minutes",
        "range_bounds": "[-120, 120]",
        "source": "constrained by modeling",
        "notes": "Shifts start-time distribution in 2021."
      },
      {
        "name": "theta_distance_decay_scale",
        "range_bounds": "[0.5, 2.5]",
        "source": "constrained by modeling",
        "notes": "Controls travel radius by scaling distance
            sensitivity in POI choice."
      },
      {
        "name": "theta_preference_vs_distance_mixture",
        "range_bounds": "[0.0, 1.0]",
        "source": "constrained by modeling",
        "notes": "Mixture weight: 1.0=personal preference only
            ; 0.0=distance-only within category."
      },
      {
        "name": "theta_infrastructure_stop_bonus",
        "range_bounds": "[0.0, 3.0]",
        "source": "constrained by modeling",
        "notes": "Boosts probability of selecting travel/
            infrastructure-like categories as intermediate
            stops in 2021."
      }
    ]
  }
}
```

F PROMPTS IN FASIM

In this section, we present the prompts used to trigger and guide each agent in FASim to perform specific functions or generate targeted outputs.

F.1 CHAIN-OF-STRUCTURE PROMPT

Chain-Of-Structure Prompt

You are an expert data scientist and simulation modeler. Your task is to analyze data for designing a simulation model. Overall simulation design is: automatically generate and calibrate the simulator. This requires defining the classes and functions within the simulator, as well as how these classes and functions interact. Next, define the parameters used by these functions and classes. The inputs are the data together with the initialization of parameters. The execution involves tuning parameters through data calibration. The final outputs are the calibrated parameters and the evaluation results on the validation dataset.

TASK DESCRIPTION:
{task_description}

DATA SUMMARIES:
{file_summaries_text}

Based on the provided task description and data summaries, please analyze the data and provide guidelines for simulation model construction.

Your analysis must cover:

1. **Overall simulation design:** State the primary objective of the simulation. Specify how the simulation initializes from inputs. Specify the execution of the simulation (should involve tuning parameters through data calibration). Specify what artifacts it outputs after execution (should be the calibrated parameters and the evaluation results on the validation dataset).
2. **Scale & Granularity:** Specify time step, spatial resolution (or explicitly "non-spatial"), and population size (agents), with brief rationale.
3. **Agent Archetypes:** Define the agent unit and roles; list static attributes and dynamic states; explain how the input data construct static attributes and how dynamic states update from data-derived signals.
4. **Interaction Topology:** Describe how agents interact; explain how to build interactions (layers/edges/protocols) from the input data.
5. **Information propagation:** Clarify whether information diffusion exists, its topology/mechanism, and how to parameterize/drive it using inputs.
6. **Exogenous Signals:** Identify any external signals/interventions, how they are derived from inputs, and how they affect agent decisions.
7. **Action Decision Policy:** Describe actions taken by agents and the decision policy mapping observations/signals to actions; include role-specific inputs and policy forms.
8. **Holdout:** Propose a training / validation data split plan if needed. For time series, prefer first 80% of days as train and last 20% as validation; state exact ranges or the rule to compute them.

9. **Simulation Evaluation:** Define evaluation metrics and how to compare simulator output artifacts against validation ground truth.
10. **Calibratable Parameters:** List tunable parameters with bounds/ranges that will be used for configuring the simulation.

Provide your response in the following JSON format (valid JSON only, no extra text):

```
{
  "overall_simulation_design": {
    "objective": "what is this simulation about",
    "initialization": "How the simulation starts from inputs (states, seeds, signals)",
    "execution": "How to tune parameters through data calibration",
    "outputs": ["List of simulation artifacts to produce (e.g., the calibrated parameters and the evaluation results on the validation dataset)"]
  },
  "scale_granularity": {
    "time_step": "seconds|minutes|hours|days",
    "spatial_resolution": "non-spatial|grid|POI|road network|other:<...>",
    "population_size": "integer or description tied to data",
    "rationale": "Why these scales are appropriate"
  },
  "agent_archetypes": {
    "unit": "e.g., resident",
    "roles": ["role1", "role2"],
    "static_attributes": ["list of static fields mapped from inputs"],
    "dynamic_states": ["list of dynamic states tracked over time"],
    "construction_from_data": "How input data map to static attributes (IDs, demographics, risk, initial states, etc.)",
    "update_from_data": "How dynamic states update using observations/signals derived from data"
  },
  "interaction_topology": {
    "topology": "graph|hybrid|broadcast|market|spatial",
    "layers": ["e.g., family", "work_school", "community", "all"],
    "protocol": "p2p on layers|group broadcast|platform-level rules",
    "construction_from_data": "How to build edges/layers from social_network / counts / metadata"
  },
  "information_propagation": {
    "exists": true,
    "topology": "layer(s) used for diffusion",
    "mechanism": "e.g., neighbor fraction, threshold, broadcast rate",
    "construction_from_data": "Which file/field drives info intensity or schedule",
    "parameters": {"names": ["e.g., info_rate", "neighbor_weight"], "notes": "how to estimate or tune"}
  },
  "exogenous_signals": [
    {
      "name": "signal_name",
```

```
      "construction_from_data": "which file/field defines or constrains it",
      "observability": "which roles can observe; delay/noise if any",
      "effect_on_agents": "how it enters decision function",
      "bounds": "[low, high]"
    }
  ],
  "action_decision_policy": {
    "by_role": {
      "role1": {
        "inputs": ["observations/signals used by this role"],
        "policy_form": "e.g., logistic with peer/self/policy terms; thresholds; inertia",
        "parameters": ["list of per-role parameters"]
      },
      "role2": {
        "inputs": ["..."],
        "policy_form": "...",
        "parameters": ["..."]
      }
    }
  },
  "holdout_plan": {
    "method": "temporal_holdout|rolling_backtest",
    "train_range": "e.g., day 0-23 or first 80% rule",
    "validation_range": "e.g., day 24-29 or last 20% rule",
    "notes": "any agent-level holdout or rolling details"
  },
  "simulation_evaluation": {
    "metrics": [
      {"name": "RMSE_aggregate", "definition": "RMSE between simulated and observed daily adoption rates"},
      {"name": "MAE_aggregate", "definition": "MAE between curves"},
      {"name": "Brier", "definition": "Brier score for per-agent probabilities if available"},
      {"name": "TransitionFit", "definition": "error on P10/P11/P01 rates vs observed"}
    ],
    "comparison_method": "how to compute metrics on validation set and report (tables/plots)"
  },
  "calibratable_parameters": [
    {
      "name": "influence_weight_family|work_school|community",
      "range_bounds": "[0,1] or rationale",
      "source": "constrained by network layers / train_data dynamics",
      "notes": "tie to topology and adoption speed"
    },
    {
      "name": "threshold|inertia|policy_weight|info_rate|memory_decay",
      "range_bounds": "[0,1] or problem-appropriate bounds",
      "source": "estimated from transitions; refined via optimization",
      "notes": "regularize to avoid overfitting"
    }
  ]
}
```

```
Return only valid JSON that can be parsed.  Do not include any other
explanation or text outside the JSON.
```

Prompt: *This is the prompt for the* ***Chain of Structure****.*

F.2 Data Analysis Prompt

Data Analysis Prompt

```
You are an expert data scientist and simulation modeler.  Your task
is to analyze the provided task description and data summaries to
design a simulation model blueprint for subsequent code generation.

Overall simulation design:  automatically generate and calibrate the
simulator.  This requires defining the classes and functions within
the simulator, as well as how these classes and functions interact.

IMPORTANT GLOBAL REQUIREMENT (capability realization):
- The simulator is potentially multi-agent.  You must decide, from
the task description and data, what agent roles are needed.  Do not
assume fixed role names; derive them.

- For every agent role you define, you must specify how the agent's
capability is realized.  Capability can be realized in one or more of
the following ways:
1.  Heuristic / rule-based logic (e.g., if-else, scoring, priority
rules).
2.  Tool / mature component / retrieval calls (e.g., look up
JSON/DB/index, call an external API wrapper, load a local model).
3.  LLM-calling with prompt injection (the agent assembles a
structured prompt from task inputs, context data, and other agents'
outputs, then calls an LLM).

- If an agent uses LLM-calling as part of its capability, you must
leave a placeholder for the prompt, such as "llm_prompt_skeleton",
and specify what fields must be injected (IDs, profiles, metadata,
history, plan/context, domain/platform hints, etc.), so that later
stages can fill or modify the actual prompt.

- The blueprint must produce clear slots/placeholders for the
subsequent Code Generation Agent, e.g., llm_run(<prompt_skeleton>,
temperature), tool_fetch_x(...), or rule_decide(...), so the code can
be generated mechanically.

CRITICAL GROUNDING POLICY (IMPORTANT):
- You must be faithful to the provided DATA SUMMARIES: do not invent
files or fields that do not exist there.

- However, not every design choice must come from data.
- If a design element can be reasonably derived or mapped from data,
you should specify the file/field mapping.
- If a design element is not directly supported by data, or no
corresponding field exists, you must mark it as "designed" and
provide a brief rationale.

- For every key design element (agent attribute/state, interaction
edge, exogenous signal, metric target), you must provide a "source_-
type" label:
- "data_derived":  directly from one or more file fields;
- "designed":  invented or assumed by the modeler/LLM, with no direct
file support;
- "mixed":  partially from data and partially designed;
- "unknown":  unclear from the summaries and requiring later
```

```
clarification.

- If source_type is "data_derived" or "mixed", include an explicit
data_reference containing the file, fields, and derivation rule.

- If source_type is "designed" or "unknown", set data_reference=null
and explain the rationale or assumption.

TASK DESCRIPTION:
{task_description}

DATA SUMMARIES:
{file_summaries_text}

Based on the provided task description and data summaries, please
analyze the data and provide guidelines for simulation model
construction.

Your analysis must cover:
1.  Overall simulation design:
- State the primary objective of the simulation.
- Specify how the simulation initializes from inputs.
- Specify the execution of the simulation; it must involve tuning
parameters through data calibration.
- Specify what artifacts it outputs after execution; these must
include calibrated parameters and evaluation results on validation.
- If any initial states, seeds, or signals are data-driven, specify
their data_reference; otherwise mark them as designed.

2.  Scale & Granularity:
- Specify time step, spatial resolution (or explicitly "non-spatial"),
and population size.
- Provide a brief rationale and source_type for each choice.

3.  Agent Archetypes:
- Define the agent unit and the set of agent roles inferred from the
task.
- For each role, list static attributes and dynamic states.
- Explain how static attributes are constructed and how dynamic states
update.
- For each attribute/state, provide source_type and, where applicable,
data_reference.

4.  Interaction Topology:
- Describe how the defined agent roles interact.
- Explain how layers, edges, or interaction protocols are constructed
from the input data.
- For each layer or edge-construction rule, provide source_type and,
where applicable, data_reference.

5.  Information Propagation:
- Clarify whether information diffusion or message passing exists.
- Specify its topology, mechanism, and how it is parameterized or
driven by inputs.
- If data-driven, provide data_reference; otherwise mark it as
designed or unknown.

6.  Exogenous Signals:
- Identify external signals or interventions.
- Explain how they are derived from inputs and how they affect agent
decisions.
- For each signal, provide source_type and, where applicable, data_-
reference.
```

```
7.  Action Decision Policy:
- Describe the actions taken by agents and the policy mapping
observations/signals to actions.
- Include role-specific policy inputs and policy forms.
- For each role's policy inputs, provide source_type and, where
applicable, data_reference.

8.  Capability Realization (CRUCIAL):
- For each agent role, specify supported realization modes from
["heuristic_rules", "tool_calls", "llm_calls"].
- If "llm_calls" is used, provide an "llm_prompt_skeleton" showing what
fields should be inserted later.
- Specify tool/data dependencies, such as indexed user/item/review
stores, feature stores, or external APIs.
- Specify a fallback strategy if the LLM or tool is unavailable.
- Specify how intermediate reasoning/results are logged back to shared
memory or execution logs.

9.  Holdout:
- Propose a training/validation split plan if needed.
- For time series, prefer the first 80% of time as training data and
the last 20% as validation data unless the task requires another
split.
- Explicitly reference the file(s) and field(s) defining temporal
ordering where applicable.

10.  Simulation Evaluation:
- Define evaluation metrics and how simulator outputs are compared
against validation ground truth.
- Explicitly name the ground-truth file(s) and field(s) associated
with each metric where applicable.

Please structure your response as a valid JSON object using the
following schema.  Return valid JSON only and do not include any
additional text.

{
  "overall_simulation_design": {
    "objective": "what is this simulation about",
    "initialization": {
      "description": "How the simulation states, seeds, signals",
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": {
        "file": "filename",
        "fields": ["field_a"],
        "derivation": "how derived"
      }
    },
    "execution": "How to tune parameters through data calibration",
    "outputs": [
      "calibrated_parameters",
      "evaluation_results_on_validation"
    ]
  },
  "scale_granularity": {
    "time_step": {
      "value": "seconds|minutes|hours|days",
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": null
    },
    "spatial_resolution": {
      "value": "non-spatial|grid|POI|road network|other:<...>",
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": null
```

```
    },
    "population_size": {
      "value": "integer or description tied to data",
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": null
    },
    "rationale": "Why these scales are appropriate"
  },
  "agent_archetypes": {
    "unit": "e.g., simulated_entity|user|household|device",
    "roles": [
      {
        "name": "role_name_derived_from_task",
        "static_attributes": [
          {
            "name": "attr_name",
            "type": "int|float|string|enum|vector|other",
            "source_type": "data_derived|designed|mixed|unknown",
            "data_reference": {
              "file": "filename",
              "fields": ["field_name"],
              "derivation": "mapping/aggregation rule"
            }
          }
        ],
        "dynamic_states": [
          {
            "name": "state_name",
            "type": "int|float|string|enum|vector|other",
            "update_rule": "how it updates",
            "source_type": "data_derived|designed|mixed|unknown",
            "data_reference": {
              "file": "filename",
              "fields": ["field_name"],
              "derivation": "mapping/aggregation rule"
            }
          }
        ],
        "construction_from_data":
          "Explain construction; if designed, explain assumption",
        "update_from_data":
          "Explain updates; if designed, explain assumption"
      }
    ]
  },
  "interaction_topology": {
    "topology": "graph|hybrid|broadcast|platform-level",
    "layers": [
      {
        "name": "layer_name",
        "from_role": "role_name",
        "to_role": "role_name",
        "edge_rule": "how nodes/edges/events are formed",
        "source_type": "data_derived|designed|mixed|unknown",
        "data_reference": {
          "file": "filename",
          "fields": ["src_id", "dst_id"],
          "derivation": "edge construction rule"
        }
      }
    ],
    "protocol": "describe message passing/order among these roles"
  },
```

```
  "information_propagation": {
    "exists": true,
    "topology": "which roles/channels are used for diffusion",
    "mechanism": "how diffusion works",
    "source_type": "data_derived|designed|mixed|unknown",
    "data_reference": {
      "file": "filename",
      "fields": ["field_name"],
      "derivation": "drive intensity/schedule"
    }
  },
  "exogenous_signals": [
    {
      "name": "signal_name",
      "effect_on_agents": "how it enters decision function",
      "bounds": "[low, high] or rationale",
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": {
        "file": "filename",
        "fields": ["field_name"],
        "derivation": "how derived"
      }
    }
  ],
  "action_decision_policy": {
    "by_role": [
      {
        "role": "role_name_derived_from_task",
        "inputs": [
          {
            "name": "obs_or_signal_name",
            "source_type": "data_derived|designed|mixed|unknown",
            "data_reference": {
              "file": "filename",
              "fields": ["field_name"],
              "derivation": "how computed"
            }
          }
        ],
        "policy_form": "policy description",
        "parameters": ["list of per-role parameter names (if any)"]
      }
    ]
  },
  "capability_realization": [
    {
      "role": "role_name_derived_from_task",
      "modes": [
        "heuristic_rules",
        "tool_calls",
        "llm_calls"
      ],
      "llm_prompt_skeleton":
        "If llm_calls is used, provide a high-level prompt",
        "template with placeholders",
      "tool_dependencies": [
        "list of tool/data lookup functions"
      ],
      "fallback_strategy":
        "what to do if LLM/tool is unavailable",
      "logging":
        "what intermediate results are written back to shared log"
    }
```

```
  ],
  "holdout_plan": {
    "method": "temporal_holdout|random_split|rolling_backtest",
    "time_ordering": {
      "source_type": "data_derived|designed|mixed|unknown",
      "data_reference": {
        "file": "filename",
        "fields": ["time_field"],
        "derivation": "ordering rule"
      }
    },
    "train_range": "rule to compute train set",
    "validation_range": "rule to compute validation set",
    "notes": "split notes"
  },
  "simulation_evaluation": {
    "metrics": [
      {
        "name": "metric_name",
        "definition": "how to compute it",
        "ground_truth": {
          "source_type": "data_derived|designed|mixed|unknown",
          "data_reference": {
            "file": "filename",
            "fields": ["target_field"],
            "derivation": "target extraction"
          }
        }
      }
    ],
    "comparison_method":
      "how to compute metrics on validation set and report"
  },
  "calibratable_parameters": [
    {
      "name": "parameter_name",
      "range_bounds": "[low, high] or rationale",
      "source": "constrained by data or modeling",
      "notes": "tie to interaction or decision speed"
    }
  ],
  "llm_and_tool_specs": {
    "llm_required": true,
    "llm_calls": [
      {
        "name": "llm_call_name_derived_from_role",
        "inputs": ["list of fields/placeholders to inject"],
        "output": "expected JSON/structured output"
      }
    ],
    "tool_wrappers": [
      {
        "name": "tool_function_name",
        "input_type": "string|object",
        "output_type": "structured_json"
      }
    ]
  }
}

Return only valid JSON that can be parsed.  Do not include any other
explanation or text outside the JSON.
```

Prompt 2: *This is the prompt for the **Data Analysis Agent**.*

F.3 Code Generation Prompt

Code Generation Prompt

```
========================
SYSTEM:
========================
You are the Code Generation Agent in a system that generates social simulations.
Transform the provided Blueprint into production-grade, executable Python code.
You MUST use the Repair Strategies as an external tool: read it first, then selectively apply only the relevant bullets.
Treat the Repair Strategies as a tool. Use relevant parts, do NOT force irrelevant parts into the solution.
If the Repair Strategies conflicts with the Task Spec, ALWAYS follow the Task Spec.

========================
USER:
========================
You will be given:
(1) A BLUEPRINT (ground truth requirements)
(2) A REPAIR STRATEGIES (reusable best practices, pitfalls, templates)

Your job: as an expert software engineer and simulation modeler, you need to generate production-grade Python code for a multi-agent simulator that follows the Blueprint below. You should implement data ingestion, agent modeling, multilayer interactions, information propagation, exogenous signals, temporal holdout, a pluggable calibration algorithm, forward simulation on the validation window, and metric evaluation.

Transform the Blueprint below into production-grade executable Python code.

When generating the code following the Blueprint, please also achieve the following requirements.

According to the blueprint's guidance, we need to:
- Read the input data files from absolute paths.
- Following the blueprint's guidance, construct the simulator code. Based on the data files, model the various elements of the simulator as designed in the blueprint, and generate the outputs specified by the blueprint.
- According to the blueprint's guidance, partition the training data (Holdout).
- Develop a calibration algorithm aimed at fitting (calibrating) the simulator's parameters. The evaluation interface should remain stable, and the calibration algorithm should be modular and replaceable.
- Using the training portion of the data, apply the calibration algorithm to estimate/fit the simulator's parameters, so that the simulator aligns as closely as possible with the real data on this segment.
- Using the validation portion of the data, allow the model to forward simulate (roll out). Compare the simulation results against the real data according to the evaluation metrics defined in the blueprint, and compute the error values.

```

```
========================
ADDITIONAL CONSTRAINTS
========================
1. Global Requirements:
 - Write clean, modular, PEP-8 compliant code with complete docstrings (triple-quoted, not truncated).
 - Provide full class/function bodies (no stubs).
 - The output must be a single standalone Python program that runs end-to-end without manual edits.
 - Ensure deterministic behavior via a global random seed.
 - Validate all inputs; raise clear exceptions with actionable messages.

2. Path Handling Instructions
When generating code, setup data file paths as follows:
```python
PROJECT_ROOT = os.environ.get("PROJECT_ROOT")
DATA_PATH = os.environ.get("DATA_PATH")
DATA_DIR = os.path.join(PROJECT_ROOT, DATA_PATH)

3. Orchestrator (Non-Empty Main Flow):
 - The generated code must include a non-empty main() function that acts as the orchestrator.
 - main() must call, in order: parse_cli() (which is optional) -> load_data() -> build_network_and_agents() -> holdout_split() -> calibrator.fit() -> simulator.rollout() -> evaluator.compute_metrics() -> save_results().
 - main() must not be empty, must not contain only pass, and must allow the program to be executed end-to-end directly with python simulate.py ....

========================
INSTRUCTIONS
========================
1) You need to list which Repair Strategies bullet IDs you will use, and for each, write a one-line justification (why relevant). Therefore, first output a Python comment block starting with:

REPAIR_STRATEGIES_USAGE_JSON = '''<one-line valid JSON enclosed in triple quotes>'''

The JSON must include: {"used_bullets":[{"id":"...","why":"..."}]}
CRITICAL: Wrap the entire JSON content in triple quotes (''') to make it a Python string literal.
Example: # REPAIR_STRATEGIES_USAGE_JSON = '''{"used_bullets":[{"id":"strategy-1","why":"reason here"}]}'''

2) If you need to inject meta JSON (e.g., CHANGE_SUMMARY_JSON or REPAIR_STRATEGIES_USAGE_JSON), if the program includes any from future import ... statements (e.g., from future import annotations), those future-import lines MUST appear before any other executable statements. Therefore, place the two JSON variables immediately after the future-import line(s) and after the module docstring (if any), but before the rest of the code.

3) Do not copy the Repair Strategies verbatim; apply it.

========================
BLUEPRINT (AUTHORITATIVE)
========================
```
```

```
{blue_print}

========================
REPAIR STRATEGIES (READ FIRST)
========================
{repair_strategies}
```

Prompt: *This is the prompt for the **Code Generation Agent**.*

F.4 Code Patch Prompt

In our design, the **Code Generation Agent (CGA)** operates in two distinct regimes. In *iteration 0*, the simulator is synthesized directly from the mechanistic Blueprint: the **Code Generation Prompt** (Appendix F.3) instructs the CGA to instantiate a complete executable simulator together with the required data, calibration, execution, and evaluation interfaces. Subsequent iterations perform *diagnosis-gated structural refinement* rather than unconstrained regeneration. After the current simulator is calibrated and executed, directly verifiable implementation issues are routed to $\mathcal{D}_t^{\text{dir}}$, whereas structurally ambiguous issues are passed through the Hypothesis Agent, frozen falsification plan, executable probe, and strict diagnosis gate to form $\mathcal{D}_t^{\text{fal}}$. The controller combines these sources into the actionable repair set $\mathcal{D}_t = \mathcal{D}_t^{\text{dir}} \cup \mathcal{D}_t^{\text{fal}}$ and converts the admissible changes into the iteration-specific `coding_patch` supplied to the CGA.

The **Code Patch Prompt** below therefore does not ask the CGA to independently diagnose the residuals or choose among competing structural explanations. Its role is downstream: given the previous executable program, the Blueprint, the current simulation evidence, and the controller-approved patch specification, it implements minimal persistent edits while preserving unaffected simulator interfaces. The **Playbook** provides reusable implementation guidance for realizing these admitted repairs, but it is not itself a source of structural validity and cannot authorize a diagnosis that has been falsified or withheld by the preceding gate. In this way, diagnosis and implementation remain separated: falsification determines *which* ambiguous structural explanations are sufficiently supported to justify modification, while the CGA determines *how* the corresponding supported changes should be realized in executable code.

Code Patch Prompt

```
========================
SYSTEM:
========================
# ROLE & OBJECTIVE
You are the Code Patch Agent for a social simulation system.
Your Goal: FIX and IMPROVE the `PREVIOUS_CODE` to satisfy the `
    BLUEPRINT` and resolve `SIMULATION_RESULTS`.
You MUST use the Playbook as an external tool: read it first,
    then selectively apply only the relevant strategies.
Treat the Playbook as a tool. Use relevant parts, do NOT force
    irrelevant parts into the solution.

# CRITICAL "PATCH-LEVEL" PROTOCOL (HARD CONSTRAINTS)
1. **Refinement, Not Rewrite**: Do NOT rewrite the entire
    program from scratch. Make minimal, targeted edits.
2. **Single File**: You must output a single updated standalone
    Python program based on PREVIOUS_CODE.
3. **Preserve Interfaces**: Do NOT rename existing public
    functions/classes or reorder major blocks unless Blueprint
    explicitly requires it.
4. **Deterministic**: Ensure deterministic behavior via a global
     random seed.
```

```

# OUTPUT REQUIREMENT (STRICT, but position-aware):
You must output the following two variables as triple-quoted
    JSON strings somewhere near the top of the file, but do NOT
    break Python syntax.

1. `PLAYBOOK_USAGE_JSON = '''...'''` (Triple-quoted JSON string)
Schema: {"used_bullets":[{"id":"<strategy_id>","why":"<relevance
    >"}]}

2. `CHANGE_SUMMARY_JSON = '''...'''` (Triple-quoted JSON string)
Schema: {"touched_symbols":[{"symbol":"<name>","reason":"<why
    >"}],
       "applied_strategies":[{"id":"<id>","applied":true}]}

Placement rules (to avoid SyntaxError):
- If the program includes any from __future__ import ...
    statements
  (e.g., from __future__ import annotations), those future-
      import
  lines MUST appear before any other executable statements.
  Therefore, place the two JSON variables immediately after the
  future-import line(s) and after the module docstring (if any),
  but before the rest of the code.
- If there is no future-import, place the two JSON variables at
  the top of the file after an optional module docstring.
- Do not wrap these JSON strings in Markdown fences.
  The rest of the output must be pure Python code.

========================
USER:
========================
Here is the context for the current iteration.

# PART 1: CONTEXT & DIAGNOSTICS (Reference Material)
========================
PREVIOUS_CODE (Baseline to Patch)
========================
{previous_code}

========================
SIMULATION RESULTS (Symptoms & Metrics)
========================
{simulation_results}

# PART 2: IMPLEMENTATION CONSTRAINTS (MUST FOLLOW)
*Attention: The following rules determine the correctness of the
     system integration.*

1. **Path Handling Instructions (COPY EXACTLY)**:
   Keep the path setup exactly as follows:

   ```python
   import os
   PROJECT_ROOT = os.environ.get("PROJECT_ROOT")
   DATA_PATH = os.environ.get("DATA_PATH")
   DATA_DIR = os.path.join(PROJECT_ROOT, DATA_PATH)

2. Orchestrator Pipeline (MUST PRESERVE MAIN FLOW):
   main() must call, in strict order:
   parse_cli() (optional) ->
   load_data() ->
   build_network_and_agents() ->
```
```

```
  holdout_split() ->
  calibrator.fit() ->
  simulator.rollout() ->
  evaluator.compute_metrics() ->
  save_results()

  You may add logging/checks, but do not remove these steps.

3. Global Requirements:
  Write clean, modular, PEP-8 compliant code.
  Complete docstrings (triple-quoted).
  Full class/function bodies (NO stubs, NO pass without reason).

  Validate all inputs; raise clear exceptions.

4. Coding patch:
  {coding_patch}

# PART 3: TARGET & STRATEGY (High-Attention Region)
CRITICAL: Read the Blueprint and Playbook immediately before
    coding.

========================
BLUEPRINT (AUTHORITATIVE)
========================
{blue_print}

========================
PLAYBOOK (The Solution Strategy)
Usage Policy:
Prioritize: blocker > high > medium > low.
Select only strategies relevant to the current Blueprint or
    Simulation Results.
If a strategy suggests a change, you MUST implement it.
========================
{playbook}

# FINAL INSTRUCTION:
1. Review the Playbook and Blueprint above.
2. Check the Simulation Results to identify what broke.
3. Apply the Playbook strategies to fix PREVIOUS_CODE while
    strictly adhering to the Implementation Constraints (Path &
    Orchestrator).

Generate the response now, starting strictly with the first line
    :
PLAYBOOK_USAGE_JSON = '''
```

Prompt: *This is the code patch prompt used by the **Code Generation Agent** during diagnosis-gated structural refinement.*

F.5 Simulation Execution Prompt

Simulation Execution Prompt

```
You are the Simulation Execution Agent in a system that generates
social simulations.  Your job is to run the generated simulation code
and collect the results.
```

```
Task Specification:
{task_spec}

Code Path:
{code_path}

Data Path (if available):
{data_path}

Please execute the simulation code and provide the results.
Consider:
1.  Running the simulation with appropriate parameters
2.  Collecting metrics and outputs
3.  Capturing any runtime errors or issues
4.  Generating visualizations if appropriate

Please structure your response as a JSON object with the following
format:

{
  "execution_status": "success|partial_success|failure",
  "runtime_errors": [
    {
      "error_type": "Type of error",
      "message": "Error message",
      "location": "Where the error occurred"
    }
  ],
  "performance_metrics": {
    "execution_time": time_in_seconds,
    "memory_usage": memory_in_mb
  },
  "simulation_metrics": {
    "metric1": value1,
    "metric2": value2
  },
  "time_series_data": [
    {
      "time_step": step_number,
      "metrics": {
        "metric1": value1,
        "metric2": value2
      }
    }
  ],
  "visualizations": [
    {
      "type": "Type of visualization",
      "path": "Path to the saved visualization",
      "description": "Description of what the visualization shows"
    }
  ],
  "summary": "Summary of the simulation execution and results"
}
```

Prompt: *This is the prompt for the **Simulation Execution Agent**.*

F.6 HYPOTHESIS AGENT PROMPT

The **Hypothesis Agent** is invoked only for issues that have been routed as `STRUCTURALLY_AMBIGUOUS`. Its role is intentionally restricted to hypothesis construction rather than diagnosis selection or repair generation. Given one ambiguous issue, the current simulator, execution evidence, task specification, and a dynamically verified capability manifest, the agent constructs the smallest coherent set of genuinely competing explanations that can be discriminated through executable

evidence. Each hypothesis must refer to an existing pathway in the current simulator, explain the same observed symptom at a comparable level of abstraction, and imply a distinct observable response under a shared feasible intervention family. The agent does not determine which hypothesis is correct; it only produces a falsifiable contrast set for the downstream Falsification Agent. Explanations that cannot be connected to runtime-verified observables or temporary access points are excluded from the executable hypothesis set rather than retained as untestable repair suggestions.

Hypothesis Agent Prompt

```
========================
SYSTEM:
========================
You are the Hypothesis Agent for falsification-before-repair in
    automated simulator construction.

For exactly one STRUCTURALLY_AMBIGUOUS issue, propose a compact
    set of genuinely competing explanations of the current
    executable simulator. Do not diagnose a winner, propose a
    repair, modify code, or introduce mechanisms absent from the
     simulator.

# HARD REQUIREMENTS

1. Return 2 to 4 hypotheses.
   - Dynamically choose the smallest coherent set.
   - Never pad the set with weak or redundant alternatives.

2. Hypotheses compete only within this issue.
   - Every hypothesis must explain the same observed symptom.
   - Do not mix unrelated issues into the same contrast set.

3. Each hypothesis must concern an existing executable pathway.
   - Do not invent mechanisms that are absent from the current
       simulator.
   - Do not introduce hypothetical modules, latent pathways, or
       new structures.

4. Construct a contrast set, not a list of generally plausible
    explanations.
   - Every hypothesis must operate at the same mechanism
       abstraction level.
   - Every hypothesis must name a distinct existing defect.
   - Every hypothesis must predict a distinct observable
       signature under a shared feasible intervention family.

5. Do not mix incompatible explanation types in one set.
   In particular, do not combine:
   - runtime mechanism defects,
   - calibration-search failures,
   - missing-mechanism claims,
   - downstream symptoms,
   - broad model inadequacy.

6. Each hypothesis must include:
   - an abstract falsifiable implication;
   - a compact contrast_signature;
   - a verified observable or access point;
   - the existing mechanism path;
   - defect type;
   - code locus.

7. The contrast_signature must discriminate hypotheses.
```

```
   - Its observable and/or predicted direction must differ from every other hypothesis in the set.
   - All hypotheses must be testable under the same shared intervention family.

8. The capability manifest has already been dynamically preflighted.
   - Use only observables and temporary access points marked runtime verified.
   - Never retain an important but inaccessible explanation in this executable contrast set.
   - Inaccessible alternatives belong in limitations, not in hypotheses.

9. required_evidence must name only:
   - exact verified observable names; or
   - verified point_ids from the capability manifest.

10. Do not propose repairs.
    - Do not suggest code edits.
    - Do not recommend parameter changes.
    - Do not state which explanation should be preferred.

11. If fewer than two executable competing explanations exist, return:
    {"hypotheses": []}

12. Hypothesis revision protocol:
    - HYPOTHESIS REVISION 0 is the initial hypothesis set.
    - At most one revision is permitted.
    - On revision 1, inspect PRIOR FROZEN HYPOTHESIS RESULTS and respond specifically to the recorded trigger.
    - If the prior complete set was falsified, generate genuinely new explanations that remain consistent with that evidence.
    - If the prior set received NO_EXECUTABLE_SHARED_PROBE, replace it with a new contrast set whose members can all be discriminated by one shared intervention available in the verified manifest.
    - Explicitly avoid the access limitation stated in plan_reason.
    - Do not rename, paraphrase, or superficially rephrase the rejected hypothesis set.

13. Return valid JSON only.
    - No markdown.
    - No prose outside the JSON object.
    - No chain-of-thought.

========================
USER:
========================

# ISSUE
{issue}

========================
TASK SPECIFICATION
========================
{task_spec}

========================
SIMULATION RESULTS
```

```
========================
{simulation_results}

========================
CURRENT SIMULATOR CODE
========================
{simulator_code}

========================
PROBE CAPABILITY MANIFEST
========================
{probe_capabilities}

========================
HYPOTHESIS REVISION
(0 is the initial set; at most 1)
========================
{hypothesis_revision}

========================
PRIOR FROZEN HYPOTHESIS RESULTS
========================
{prior_hypothesis_results}

========================
OUTPUT
========================
{
  "hypotheses": [
    {
      "hypothesis_id": "H1",
      "explanation": "structural explanation",
      "mechanism_path": "existing source -> state -> outcome path
          ",
      "defect_type": "misconnection|misrouting|sensitivity|
          state_update|persistence|other_existing_defect",
      "falsifiable_implication": "observable implication if this
          explanation holds",
      "contrast_signature": {
        "shared_intervention_family": "one verified intervention
            usable across the set",
        "observable": "verified observable or point_id",
        "direction": "INCREASE|DECREASE|NO_MEANINGFUL_CHANGE|
            ACTIVATES|DOES_NOT_ACTIVATE",
        "distinctive_reason": "why this differs from every other
            row"
      },
      "required_evidence": [
        "observable or point_id required"
      ],
      "access_assessment": "RUNTIME_VERIFIED_EXISTING_HOOK |
          RUNTIME_VERIFIED_TEMPORARY_ACCESS; explain briefly",
      "code_refs": [
        {
          "symbol": "function_or_class",
          "lines": "Lx-Ly or unknown"
        }
      ]
    }
  ]
}
```

Prompt: *This is the prompt used by the **Hypothesis Agent** to construct executable competing explanations for one structurally ambiguous issue.*

F.7 Falsification Agent Probe Planning Prompt

The **Falsification Agent** acts as a challenger to structurally ambiguous diagnoses proposed by the Hypothesis Agent. For each active hypothesis set, it designs one executable counterfactual probe that maximizes discrimination among the still-plausible explanations while preserving the current calibrated simulator as the object being diagnosed. The agent is not permitted to repair simulator code, change calibrated parameters, or introduce a missing mechanism. Instead, it must operate through the simulator's declared probe capabilities, select a valid intervention and observable evidence, and specify for each hypothesis a predicted signature together with a falsification condition that can be evaluated after execution.

The probe plan is frozen before the execution outcome is exposed. This includes the intervention, observables, evidence regime, matched seeds where applicable, predicted signatures, falsification conditions, and any threshold requirements. For stochastic quantitative responses, the Falsification Agent specifies only the expected direction of change; the workflow controller derives the meaningful-effect cutoff from baseline variability using the pre-defined robust thresholding rule. Probe evidence is subsequently evaluated against the frozen plan by the controller and adjudication function Γ, yielding FALSIFIED, SURVIVES, or INCONCLUSIVE verdicts. The agent therefore designs the experiment but does not retrospectively redefine the test after observing its result.

The workflow allows at most three sequential probes for an issue and at most one hypothesis-set revision. Probe 1 is required to target the complete active hypothesis set. A later probe is permitted only when the preceding frozen probe yields usable but only partially discriminating evidence and the strict diagnosis gate remains unresolved; uniformly surviving or uniformly inconclusive outcomes do not automatically authorize additional probing. If the original hypothesis set cannot be challenged through any executable shared contrast, the controller may instead invoke the one-time hypothesis-revision path. These restrictions prevent repeated probing from becoming an unconstrained search procedure and preserve the falsify-before-repair principle.

Falsification Agent Prompt

```
========================
SYSTEM:
========================
You are the Falsification Agent acting as an experiment designer
and challenger. Design the requested executable counterfactual
probe for the still-active competing hypotheses. Do not repair
code and do not change calibrated parameters or invent a missing
mechanism.

This workflow permits at most three sequential probes and at
    most
one hypothesis-set revision. Probe 1 should maximize
    discrimination
across the complete hypothesis set. A later sequential probe is
conditionally approved only when the preceding executed probe
produced usable, partially discriminating evidence (different
verdict classes across hypotheses) but did not satisfy the
    strict
diagnosis gate. Uniform SURVIVES or uniform INCONCLUSIVE
    outcomes
do not authorize another paid probe. An unexecutable shared
    contrast
may instead trigger the separate one-time hypothesis-set
    revision
path.

```

```
An approved later probe receives the active revised or non-falsified
hypotheses and must target only the distinctions or missing evidence
left unresolved. If HYPOTHESIS REVISION is 1, a single new contrast
set has been generated because the original set was fully falsified
or admitted no executable shared probe; do not reuse rejected
explanations, the documented inaccessible intervention, or an
identical probe. Never repeat an earlier probe unless the new plan
explicitly fixes earlier unavailable evidence. Each probe is
independently frozen before execution; prior predictions may not
be changed.

For any later probe, an observable with zero valid pairs in PRIOR
FROZEN PROBE RESULTS is forbidden as required evidence. Choose a
different reachable observable/name and access point. The controller
rejects repeated unavailable requirements before execution.

The simulator exposes the capability manifest below. Use existing
hooks when sufficient; otherwise choose statically discovered
temporary access points. If none permits a discriminating experiment,
return only:

{"probe_id": "UNRESOLVED",
 "reason": "specific missing intervention/observation and why it
     is needed"}

This no-probe outcome is not a Probe Plan.

Access modes:

1. EXISTING_HOOK:
  Choose an intervention from manifest.interventions and declared
  observables.

2. OBSERVATION_INSTRUMENTATION:
  Use an existing declared intervention and add read-only observations
  at manifest.temporary_access.points.

3. TEMPORARY_INTERVENTION_ADAPTER:
  Set one existing scalar at a temporary point whose operations
  include "set", optionally adding observations. Observation-only
  points cannot be intervened upon. intervention.target is the exact
  point_id; operation is "set"; value is a finite builtin scalar of
  the same type as the original state. adapter_occurrence is "all"
  (default) or a positive 1-based occurrence index. Every original
  RHS executes once, preserving its random draws, then the chosen
  state is overridden only in probe runs.
```

```
66
67 Temporary observations use an instrumentation list. Copy real
       point_ids
68 and context roots from the manifest. All instrumentation names
       must be
69 included in observables and start with "instrument:".
       Observation
70 occurs after any requested override.
71
72 Supported bounded evidence types:
73
74 - SCALAR:
75   reducer is last, mean, sum, count, min, max, any, or all.
76
77 - HISTOGRAM:
78   provide explicit numeric bins.
79
80 - QUANTILES:
81   provide probabilities and max_samples within the manifest
         bound.
82
83 - CATEGORICAL_COUNTS:
84   provide max_categories.
85
86 - GROUPED_HISTOGRAM:
87   provide bins, context field-to-accessor mappings, group_by
         context
88   fields, and max_groups.
89
90 - PAIRED_RECORDS:
91   provide context mappings, key_by context fields, and
         max_records.
92   Use this for resident-day or other stable paired keys.
93
94 - EVENT_TRACE:
95   provide max_records and any available context mappings.
96   Empty context is valid for a bounded ordered scalar-value
         trace.
97
98 Context accessors may only be a discovered local root or its
       read-only
99 attributes; no calls, subscripts, or arbitrary expressions.
       Structured
100 collectors receive the scalar value at point_id plus the
       declared
101 context. They summarize or bound records inside the runtime;
       never
102 request unbounded raw trajectories, arrays, candidate
       collections, or
103 parameter histories.
104
105 A conditional observation point may be reported as structured
106 UNAVAILABLE without failing the run. An intervention point that
       is
107 never executed still invalidates the probe.
108
109 Every dynamically verified temporary point includes
       runtime_value_type
110 and compatibility flags. HISTOGRAM, QUANTILES, and
       GROUPED_HISTOGRAM
111 require numeric_evidence_compatible=true. Boolean or string
       points may
```

```
use SCALAR, CATEGORICAL_COUNTS, PAIRED_RECORDS, or EVENT_TRACE, but
never numeric histograms. Treat these flags as hard execution
constraints.

Hard requirements:

- Exactly one probe in this planning response, using exactly one of the
  three access modes above.

- Existing-hook runtime paths come from the manifest; temporary targets
  must match a discovered point_id.

- Observables must be declared native names/prefixes or explicitly
  defined instrumentation names.

- Hold simulator structure, calibrated parameters, data, topology,
  timing, initial state, and matched seeds fixed except for the declared
  intervention.

- evidence_mode is CONTROLLED_STRUCTURAL for clear trace/path predicates,
  otherwise MATCHED_SEED.

- Every predicted signature and falsification condition must be
  executable/decidable.

- Every hypothesis_test must list evidence_requirements. Each requirement
  names a requested observable and the exact evidence_type supplied by
  the plan. Do not claim a histogram, grouping, pairing, or event
  relationship from SCALAR evidence.

- At least one hypothesis must have a unique frozen prediction; otherwise
  return UNRESOLVED because the strict diagnosis gate can never select
  one survivor.

- BOOLEAN_OR_TRACE signatures use explicit predicates.

- QUANTITATIVE signatures name the observable and an effect_direction,
  but must not invent numeric cutoffs. The controller freezes a
  BASELINE_VARIABILITY/ROBUST_MAD threshold formula and computes its
  cutoff from baseline evidence before verdict evaluation. Use
  INCREASE, DECREASE, or NO_MEANINGFUL_CHANGE.

- A BOOLEAN_OR_TRACE signature may combine several quantitative effect
  classifications, but then it MUST declare threshold_requirements with
  one requested observable and effect_direction for every INCREASE,
```

```
 DECREASE, or NO_MEANINGFUL_CHANGE claim. The controller compiles every
declared cutoff before evaluation.

- Do not invent a derived metric in predicate prose. Every quantitatively
 classified quantity must be an exact requested observable that the
runtime returns. Use a directly collected histogram, quantile, count,
 grouped distribution, paired record, or event trace instead of asking
the evaluator to derive an undeclared quantity from truncated samples.

- MATCHED_SEED must freeze an explicit unique integer seed list and
 stability_rule.

- For stochastic probes, normally use five matched seeds. Probe 1 should
 not consume the entire 20-call case budget when a second probe may be
 needed.

- Return valid JSON only.

========================
USER:
========================
PROBE SEQUENCE NUMBER:
{probe_sequence}

HYPOTHESIS REVISION:
{hypothesis_revision}

REMAINING SIMULATOR-CALL BUDGET:
{remaining_simulation_call_budget}

PRIOR FROZEN PROBE RESULTS (empty for Probe 1):
{prior_probe_results}

========================
ISSUE
========================
{issue}

========================
HYPOTHESES
========================
{hypotheses}

========================
TASK SPECIFICATION
========================
{task_spec}

========================
CURRENT SIMULATION RESULTS
========================
{simulation_results}

========================
```

```
CURRENT SIMULATOR CODE
========================
{simulator_code}

========================
EXECUTABLE PROBE CAPABILITY MANIFEST
========================
{probe_capabilities}

========================
OUTPUT
========================
{
  "probe_id": "short-semantic-id",
  "goal": "what the probe discriminates",
  "access_mode": "EXISTING_HOOK",

  "intervention": {
    "target": "existing variable/path/input",
    "operation": "set|inject|disable",
    "value": 0
  },

  "hold_fixed": [
    "calibrated_parameters",
    "initial_state",
    "topology",
    "timing"
  ],

  "observables": [
    "observable_name"
  ],

  "instrumentation": [
    {
      "point_id": "Simulator.rollout:123:distance",
      "observable": "instrument:distance_by_transition",
      "evidence_type": "GROUPED_HISTOGRAM",
      "bins": [0, 1, 2, 5, 10, 20, 50],
      "context": {
        "origin": "current_category",
        "destination": "selected_category"
      },
      "group_by": [
        "origin",
        "destination"
      ],
      "max_groups": 50
    }
  ],

  "evidence_mode": "CONTROLLED_STRUCTURAL|MATCHED_SEED",

  "seeds": [11, 23, 37],

  "stability_rule": {
    "minimum_directional_fraction": 0.8,
    "description": "pre-execution rule"
  },

  "hypothesis_tests": [
    {
```

```
    "hypothesis_id": "H1",

    "predicted_signature": {
      "kind": "BOOLEAN_OR_TRACE|QUANTITATIVE",
      "predicate": "decidable condition"
    },

    "falsification_condition": {
      "predicate": "decidable contradictory condition"
    },

    "effect_direction":
      "INCREASE|DECREASE|NO_MEANINGFUL_CHANGE (required for
          QUANTITATIVE)",

    "threshold_requirements": [
      {
        "observable": "mean_stop_count",
        "effect_direction": "INCREASE"
      }
    ],

    "evidence_requirements": [
      {
        "observable": "instrument:distance_by_transition",
        "evidence_type": "GROUPED_HISTOGRAM",
        "required": true
      }
    ]
  }
 ]
}
```

Prompt: *This is the falsification-planning prompt used by the **Falsification Agent** to construct and freeze executable counterfactual probes before outcome evaluation.*

F.8 Falsification Agent Evidence Evaluation Prompt

The **Falsification Agent** is invoked a second time after the frozen probe has been executed, but its role at this stage is strictly limited to evidence adjudication. The agent receives only the previously frozen probe plan and the resulting execution evidence bundle, and must judge each hypothesis against the criteria that were fixed before execution. It is not allowed to modify the probe, introduce new observables, reinterpret the predicted signatures, alter thresholds, add new falsification conditions, select a final diagnosis, or propose a structural repair.

This separation is important because the experiment-design and evidence-evaluation stages must not collapse into post-hoc reasoning. For quantitative signatures, threshold construction is owned by the workflow controller: the frozen plan specifies the threshold formula, and the realized evidence bundle contains the corresponding controller-computed threshold values. The adjudicator must apply those values exactly rather than inventing or relaxing a cutoff after observing the result. If required evidence or a required threshold is unavailable, the appropriate verdict is INCONCLUSIVE. The resulting per-hypothesis verdicts are subsequently consumed by the controller, which applies the strict unique-survivor diagnosis gate described in Section 3.2.

Falsification Adjudication Prompt

```
========================
SYSTEM:
```

```
========================
You are the Falsification Agent acting only as an evidence
adjudicator.

The probe plan below was frozen before execution. You may not
    change,
relax, reinterpret, or add predictions, thresholds, observables,
seeds, or falsification conditions. Judge each hypothesis only
    against
its frozen criteria and the supplied real execution evidence.

Verdicts:

- FALSIFIED:
  Stable evidence clearly contradicts the frozen signature.

- SURVIVES:
  Stable evidence is consistent with the frozen signature.

- INCONCLUSIVE:
  Evidence is unstable, incomplete, unavailable, or does not
      decide
  the frozen predicates.

SURVIVES never means true. Do not select a final diagnosis and
    do not
propose a repair. Return exactly one verdict per hypothesis as
    valid
JSON only.

For every declared threshold_requirement, including those used
    inside
composite BOOLEAN_OR_TRACE signatures, the controller-owned
    threshold
formula in the frozen plan and the realized
    controller_thresholds in
the evidence bundle override any informal or numeric wording in
    the
LLM-authored predicate.

Apply each frozen effect_direction against the corresponding
meaningful_effect_threshold. Do not create, move, relax, or
    reinterpret
a threshold after seeing results.

If a required controller threshold is unavailable, return
    INCONCLUSIVE
for every hypothesis that requires it.

========================
USER:
========================
FROZEN PROBE PLAN:
{frozen_probe_plan}

========================
EVIDENCE BUNDLE:
{evidence_bundle}

========================
OUTPUT
========================
{
```

```
  "verdicts": [
    {
      "hypothesis_id": "H1",
      "verdict": "FALSIFIED|SURVIVES|INCONCLUSIVE",
      "evidence": "specific frozen predicate and observed result"
    }
  ]
}
```

Prompt: *This is the post-execution adjudication prompt used by the **Falsification Agent** to evaluate each hypothesis strictly against its frozen prediction and the recorded execution evidence.*

G Limitations and Future Work

While FASim demonstrates promising capabilities in automated simulator construction with falsification-guided structural diagnosis, several limitations remain:

1. **Dependence on LLM-generated hypotheses and probe designs.** FASim relies on LLMs to construct mechanistic blueprints, propose competing structural hypotheses, design executable falsification probes, and implement accepted repairs. Although the workflow constrains these stages through structured schemas, executable capability checks, frozen probe plans, and diagnosis gating, the quality of the resulting refinement process still depends on the reasoning and code-understanding capabilities of the underlying LLM. In particular, a relevant structural explanation may never be tested if it is not proposed in the active hypothesis set, and a weakly designed probe may fail to discriminate among otherwise valid alternatives. Stronger hypothesis-coverage strategies, automated diversity constraints, and more systematic probe-generation procedures are therefore important directions for future work.
2. **Falsification is limited by simulator observability and intervention access.** Executable falsification can only test hypotheses whose implications can be exposed through available interventions, internal access points, or observable execution evidence. Some structural mechanisms may be difficult to isolate because the relevant internal state is not reachable, the required intervention cannot be applied without changing simulator semantics, or multiple mechanisms produce indistinguishable responses under the available probes. In such cases, FASim deliberately returns inconclusive evidence rather than forcing a diagnosis. Extending the probe interface with richer but still auditable intervention and observation capabilities would broaden the class of structural failures that can be discriminated.
3. **Structural survival does not establish real-world causal truth.** A hypothesis that SURVIVES a frozen executable probe is only provisionally supported relative to the current simulator, competing hypothesis set, and available evidence. The procedure does not establish that the surviving mechanism is the unique or true causal explanation of the real-world system. This limitation is especially important when several structurally different simulators can reproduce similar macro-level behavior. Our controlled defect-recovery experiments provide ground truth only because the corrupted mechanism is known by construction; such supervision is generally unavailable in natural simulation tasks. Future work could incorporate stronger external structural evidence, domain constraints, or independent intervention data where available.
4. **Execution cost and orchestration complexity.** The current implementation uses a centralized synchronous outer loop in which simulator calibration, hypothesis generation, probe execution, diagnosis gating, repair, and recalibration are performed sequentially. Falsification may additionally require repeated matched-seed baseline/probe executions, making the process more expensive than reflection-only refinement. Although baseline reuse and bounded probe budgets reduce redundant executions, cost can still grow with simulator complexity, stochasticity, the number of ambiguous issues, and the number of candidate hypotheses. Parallel probe execution, adaptive probe budgeting, and asynchronous orchestration are promising directions for improving scalability.

5. **Scope of simulator families and agent representations.** Our experiments focus on agent-based social simulators with explicit micro-level states, interaction rules, and micro-to-macro evaluation. The falsification-guided diagnosis principle is not inherently restricted to this setting, but applying it to other simulator families—such as ODE/PDE systems, physics engines, or hybrid learned simulators—would require suitable structural representations, intervention interfaces, and executable evidence contracts. Likewise, `FASim` currently targets agents with explicit executable state-transition mechanisms rather than long-horizon communicative agents whose behavior is primarily generated through dialogue, persistent memory, or unconstrained LLM interaction. Extending structural falsification to such settings remains an open problem.
6. **Evaluation breadth and long-horizon robustness.** We evaluate `FASim` across user modeling, intervention-driven mask adoption, and personal mobility under both ID and OOD settings, but these benchmarks still cover a limited subset of possible simulator-construction tasks. Broader evaluation is needed across longer temporal horizons, larger populations, richer intervention types, more strongly coupled multi-agent systems, and domains with independently verifiable structural knowledge. Such settings would provide stronger tests of whether falsification-guided diagnosis continues to prevent unsupported structural edits as simulator scale and mechanism complexity increase.

H Societal Impact

`FASim` advances automated simulator construction by translating natural-language task briefs and heterogeneous data into executable, calibrated simulators whose *micro-entity* rules are evaluated through *micro-to-macro* evidence. This capability can support scientific discovery and decision-making in domains such as public health, mobility planning, and human-centered recommender systems. In particular, it enables faster prototyping of mechanistic hypotheses, uncertainty-aware what-if analysis, and more reproducible simulation pipelines.

At the same time, such systems carry potential risks. Simulators may be misused for surveillance, behavioral manipulation, or overconfident policy guidance if their outputs are treated as ground truth rather than model-based approximations. They may also amplify bias when the input data encode historical inequities or underrepresent affected populations. `FASim` partially mitigates these risks through explicit mechanism specification, calibration against observed evidence, and diagnostic feedback that surfaces residual mismatches rather than obscuring them. For deployment in high-stakes settings, we recommend domain-expert review, transparent reporting of data provenance and uncertainty, and external validation. These review procedures are safeguards for responsible use rather than components of the automated `FASim` pipeline evaluated in this paper.

I Ethics Statement

The experiments reported in this paper **do not involve human-in-the-loop** intervention, human-subject data collection during the optimization loop, crowdsourcing, or external expert feedback injected into blueprint generation or repair-directive generation. All reported results are produced by the automated `FASim` pipeline under fixed task specifications, datasets, calibration budgets, and evaluation protocols.

`FASim` includes **optional interfaces** for domain-expert inspection of generated blueprints and repair directives. These interfaces are designed as deployment-time engineering safeguards rather than as components of the core algorithm. They may be useful in rare cases where the automatically generated blueprint substantially deviates from the intended simulator design, or where feedback-guided code optimization repeatedly fails to converge despite metric-grounded repair attempts. In the reported experiments, these optional expert-intervention interfaces were disabled, and we did not observe such failure cases requiring external intervention.

Although these interfaces were not activated in the reported experiments, enabling them in future deployments could introduce ethical considerations, including participant consent, privacy protection, data management, and appropriate compensation for expert labor. In addition, as part of the broader development and continuous improvement of the `FASim` system, we collected human feedback

from computer science researchers and social scientists on the quality, interpretability, and practical usefulness of generated simulators and system outputs. This feedback was used only to inform future system design and implementation, not to modify the experimental blueprints, repair directives, simulator code, or reported benchmark results.

For these reasons, we completed the relevant ethics approval process in advance to mitigate potential risks in future human-facing use of the system. The participant instructions, consent materials, and disclaimers described below were prepared under this precautionary framework and apply to optional human-feedback or expert-inspection studies, rather than to the automated experimental results reported in the main paper.

The released artifacts do not contain personally identifiable information. The datasets and benchmark artifacts are either synthetic, publicly available, or derived from task settings described in the paper. We discuss potential misuse risks of automated social simulation, including surveillance, behavioral manipulation, bias amplification, and overconfident policy guidance, and recommend external validation and domain review before any high-stakes deployment.

1. **Ethics Approval and Oversight.** The optional human-feedback and expert-inspection procedures associated with this project were reviewed and approved by the authors' institution (Approval ID anonymized for blind review). The approved protocol covers future or auxiliary human-facing studies in which domain experts or researchers may inspect generated blueprints, review repair directives, evaluate simulator quality, or provide feedback on system usability and output fidelity. These procedures were not used to alter the automated optimization loop or the benchmark results reported in this paper. Recruitment details are anonymized for double-blind review and will be disclosed in the camera-ready version if accepted.
2. **Participant Recruitment and Consent.** For auxiliary system-development feedback, we recruited participants including computer science researchers and social scientists to comment on the quality, interpretability, and practical usefulness of generated simulators and system outputs. Prior to participation, all participants were provided with a Participant Information Statement and provided written informed consent. Participation was strictly voluntary, with a clear option to withdraw at any time without penalty. No participant feedback was injected into the reported experimental runs as blueprint edits, repair directives, code patches, or calibration decisions.
3. **Data Privacy and Management.** To protect participant privacy, all feedback collected through optional expert-inspection or system-development activities was de-identified before analysis. Personally identifiable information (PII) was removed prior to storage or use. Data is stored on secure, password-protected institutional servers with access restricted solely to the research team. No PII is contained in the artifacts, datasets, prompts, generated code, or models released with this work.
4. **Potential Risks and Mitigation.** Participation in optional expert-inspection or feedback activities is considered low risk. Potential discomforts are limited to minor psychological stress associated with using a new software interface, reviewing automatically generated simulator outputs, or reflecting on professional judgments about simulator fidelity. There may also be a time burden associated with evaluation or feedback tasks. To mitigate these risks, all tasks were designed to be non-invasive, participants were informed of their right to pause or withdraw at any time without penalty, and technical support was provided to minimize frustration during interaction with the `FASim` system.
5. **Participant Instructions and Disclaimers.** To ensure transparency and informed engagement, we prepared detailed participant information and consent documents for optional human-feedback and expert-inspection activities. These documents explain the workflow, the distinction between automated experimental runs and optional human-facing evaluation, data handling procedures, privacy protections, and potential risks such as technical frustration or time burden. Participants are required to review these materials and acknowledge the relevant disclaimers before providing feedback.
6. **Recruitment Channels and Payment Adequacy.** Recruitment for auxiliary feedback activities was conducted primarily through institutional mailing lists and professional networks affiliated with the authors' institution and partner institutions, targeting both technical and

domain-oriented cohorts such as computer science researchers and social scientists. All annotators are English speakers, and the recruitment process did not target any specific racial, ethnic, or demographic groups. Participants were compensated for their time via prepaid gift cards. Participants, if involved in future deployments or expert-inspection studies, should be compensated according to applicable local guidelines.

7. **Broader Impact.** While `FASim` aims to automate simulator construction, we emphasize that the system is designed to augment, not replace, human domain expertise. The core method evaluated in this paper is an automated evidence-grounded pipeline, and the reported experiments do not rely on human-in-the-loop intervention. At the same time, optional expert-inspection interfaces are retained as responsible-use safeguards for future deployments, especially in high-stakes domains where generated simulators may influence policy, planning, or institutional decisions. Such interfaces can help detect misaligned blueprints, non-convergent repair loops, hallucinated mechanisms, or ethically problematic simulation assumptions before deployment.