AUTOMATICALLY BUILDING MOLECULAR PROPERTY PREDICTION MODELS WITH AN EXPERT-GUIDED AI AGENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Molecular property prediction guides compound prioritisation in drug discovery, but turning domain expertise into an effective model takes machine-learning engineering and repeated cycles of design, implementation, training and evaluation. We present a hierarchical agent pipeline that automates this workflow. Given a task description, the data and optional written expert guidance, it runs an agent loop that designs a model, writes and debugs the code, trains and tunes it, evaluates the result and, when the results call for it, revises the design, then selects and delivers a final model, so a domain expert steers development in prose rather than code. A controlling language-model agent assigns these steps to specialised sub-agents and decides each next step from their reports and training results; the pipeline code is the same for every task. We evaluate it retrospectively on the OpenADMET pregnane X receptor (PXR) induction challenge, a recent blind competition, using guidance we wrote from published challenge reports and earlier runs of this pipeline after the competition ended. Across three runs, the guided pipeline achieves a mean relative absolute error of 0.561, compared with 0.627 without guidance; the published winning score of 0.563, obtained using proprietary data, provides an external reference. On two further benchmarks, hepatocyte clearance and plasma protein binding, the same pipeline with task-specific guidance scores better on average than the published leaderboard bests. Given guidance that names no model, the pipeline's run fell within the range of three AIDE runs and scored better than three MLEvolve runs.

1 INTRODUCTION

Predicting a molecular property well takes domain knowledge, modelling judgement and many rounds of implementation, training and checking. The developer has to decide how to represent molecules when labelled data are scarce, whether related measurements on the same compounds can help and how to use them when most of their labels are missing, how to validate when the compounds of interest are close analogues of each other rather than a random sample, and how to diagnose where the model fails. In the pregnane X receptor (PXR) induction challenge, for example, the evaluation compounds are analogues of potent hits, chosen to mimic lead optimisation (OpenADMET, 2026b; Swain et al., 2026).

Domain knowledge helps with these decisions, but it does not become a working model by itself. A recommendation such as "the single-concentration screening readouts carry the same biology as the potency label" still has to be turned into a representation, a training objective that tolerates missing labels, a validation scheme that respects the analogue structure, training runs, and a selection among the results, and that engineering must be redone for every task. Conventional AutoML optimises models and hyperparameters within a predefined search space (Feurer et al., 2015; Erickson et al., 2020), and language-model agents now generate, debug and improve modelling code from task descriptions and data (Jiang et al., 2025; Chi et al., 2024; Chan et al., 2024), some drawing on curated knowledge bases (Du et al., 2026; Ou et al., 2025), and any of them can be handed written guidance. We ask how such a system should be organised to take over this engineering for a domain expert, from design and implementation through training and model selection to a delivered model.

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.

We present a hierarchical agent pipeline for this purpose. A controlling agent assigns setup, design, implementation and training, and selection to specialised sub-agents, reads each report and the training results, and decides whether to tune further, revise the design or select and deliver a model; every agent reads the guidance as text, so the pipeline code is the same for every task. PXR is our main case. To test whether the same pipeline carries to other molecular properties, we also evaluate it on two absorption, distribution, metabolism, excretion and toxicity (ADMET) benchmarks, hepatocyte clearance and plasma protein binding, where its guided runs score better on average than the published leaderboard bests without any change to the pipeline code.

Our contributions are threefold. First, a hierarchical pipeline that automates molecular property model development, in which a controlling agent directs role-specific sub-agents from their reports, revises the design from training feedback and ends with a selected, delivered model (Section 3). Second, its evaluation: on PXR the guided runs reach a mean relative absolute error (RAE) of 0.561 across three runs, against 0.627 without guidance, beside a published winning score of 0.563 obtained with proprietary data (Section 5). Third, an analysis of where the performance comes from, through execution records, component ablations and comparisons with a single coding agent, AIDE and MLEvolve (Sections 5.3 to 5.5).

2 Related Work

Automated Machine Learning and Coding Agents. Conventional AutoML searches a fixed space of models and hyperparameters (Feurer et al., 2015; Erickson et al., 2020); agents built on large language models instead write and evaluate code. AIDE searches a tree of single-file solutions with operators that draft, debug and improve them (Jiang et al., 2025); SELA searches pipeline configurations with Monte Carlo tree search (Chi et al., 2024); MLEvolve searches a graph of solutions with cross-branch references, a domain knowledge base and a global memory (Du et al., 2026); and AutoMind grounds a tree search in a curated expert knowledge base (Ou et al., 2025). Written knowledge is therefore not new to such systems, and AIDE and MLEvolve take the same guidance as our pipeline in Section 5.5. What differs here is how development is organised, by a controlling agent that directs separate role sessions from their reports (Section 3); general-purpose coding agents instead work in one session with shell and file tools (Yang et al., 2024; Wang et al., 2025), as our single-agent baseline does.

Domain Knowledge and Agents for Molecular Prediction. Domain knowledge commonly enters molecular models through pretrained representations (Burns et al., 2025), auxiliary tasks trained jointly with the target (Heid et al., 2024) and hand-built features, and LLM4SD turns rules synthesised from the literature and data into interpretable features (Zheng et al., 2025). Here a pipeline of agents turns such knowledge, given as prose, into code and a trained model. MolAgent automates molecular property modelling with stacked ensembles and nested cross-validation on ADMET benchmarks from the Therapeutics Data Commons (TDC) (Gómez-Tamayo et al., 2025; Huang et al., 2021). Ning et al. (2026) run a closed research loop over representations, estimator code and external measurements on 36 tasks and find that validation-selected gains often fail to transfer to held-out data; they study the reliability of autonomous search, while we vary the supplied guidance and the system that executes it.

3 Method

3.1 System Overview

The design separates what to build, which changes with each task and arrives as text, from how to build it, agent code that is the same for every task and reads that text at every step, so a new task needs its own data, description and guidance but no changes to the pipeline code.

The pipeline takes three inputs: a task directory, holding the data and fold-assignment files; a task description stating what is predicted, how it is scored and what file to deliver; and the optional guidance. A long-lived controlling agent runs the development loop and invokes specialised sub-agents as tools, one to set up the environment, one to produce or revise a model design, one to implement and train it, and one to select among trained artifacts and write the predictions. The

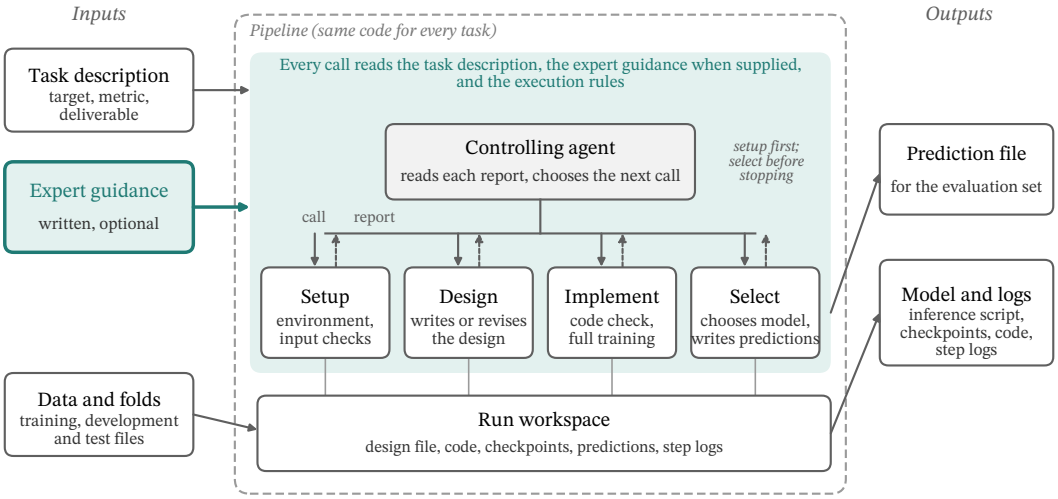

Figure 1: Pipeline overview. The controlling agent calls setup, design, implementation and selection sub-agents, reads each report and chooses the next call. Every call receives the task description, the expert guidance when supplied (teal) and the execution rules; data and fold files are read from the task directory, and every call writes to the run workspace. The order beside the controlling agent is instructed, not enforced; Section 3.5 gives the limits and when a run ends.

outputs are the prediction file, a standalone inference script with the model files it loads, and the code, checkpoints and logs of the agents' tool calls (Figure 1). Each sub-agent is a full language-model session that runs code, reads the results and corrects itself over many turns, then returns a report. All agents have the same built-in tools, among them a shell, file editing, web search and a tool that starts a helper agent; the sub-agents differ in their instructions, their output schema and, except for setup, where their file-editing tools may write. The controlling agent keeps one session for the whole run, and its instructions confine it to reading files and calling sub-agents, although in six of the 21 runs it ran shell commands, in two to inspect them. In the reported runs each sub-agent call starts a new session, so it learns what earlier calls did only from the workspace files and the controlling agent's instruction for it.

These mechanisms serve the aim of carrying a task to a delivered model without the user writing code: each stage has one responsibility and a report; the controlling agent turns reports and training results into tuning, redesign or selection; files hand work between stages; and selection ends with a delivered model rather than code alone. These are design reasons, not measured benefits; Section 5.5 compares the division of work with a single agent, a comparison that cannot test adaptive control.

3.2 Task Specification and Expert Guidance

The task description defines the task, namely the data and their columns, the target, the metric, the fold assignment and the deliverable, and adds working rules such as fixing the numbers of epochs and folds before training. The guidance adds knowledge that the task materials lack, general or specific to the task. We call it expert guidance for its content, not its author: the authors wrote each document used here from published challenge reports and earlier runs (Section 4.2), and what we measure is the effect of supplying it against withholding it, not expert against non-expert authorship. Both texts are assembled into one block of instructions that the controlling agent and every sub-agent receive in full: the description under its own heading, and the guidance verbatim under a heading that ranks it above every rule except error reporting. No code routes parts of the document to particular steps or extracts settings from it, and a run without guidance receives the block with that section absent.

Guidance can be written at three levels of specificity, which the PXR auxiliary measurements illustrate. A *domain judgement* says what may matter: the single-concentration screening readouts reflect the same biology as the potency label on a different scale. A *modelling suggestion* names a direction to evaluate: learn those readouts jointly with the target. An *implementation prescription* fixes the

decision: heads for the target, the three readouts and lipophilicity, weighted 4:1:1:1:1. The further a document is from prescription, the more of the design it leaves to the system. Section 4.2 places each document used in this study on this scale.

3.3 Agent-Controlled Model Development

A run starts with setup, which builds the software environment and checks the inputs. The design sub-agent then writes a design covering the representation, model, training objective, validation and the numbers of epochs and folds, drawing on the instruction block and any earlier training reports; when the task names a dataset or benchmark, its instructions ask it to search the web. The implementation sub-agent turns the design into code in two modes. In implementation mode it writes the code and checks it end to end on a reduced test run; in tuning mode it trains, may adjust training settings within the plan, and writes checkpoints, predictions and internal scores, with every random seed fixed at 42.

The controlling agent links these steps. Each call it makes carries a free-text instruction, which the sub-agent treats as non-binding background and which may restate parts of the guidance, and each call returns a report of what was done, the key metrics, the issues met and suggested next steps. From these reports and the files it reads, the controlling agent chooses what comes next: further tuning, a revised design followed by reimplementation, or selection (Section 3.4). Its instructions fix the order: setup comes first; every implementation call is followed by a tuning run, since the test run says nothing about model quality; a design is revised only after at least one tuning run; an issue a sub-agent cannot solve is passed to a different one; a call that keeps failing for the same reason prompts a change of course or a stop; and selection precedes stopping. These are instructions only, with no set number of repetitions and no check in the code. Within its own call, the implementation sub-agent is told to fix and rerun a failed training run and to restart a diverging one.

The guidance enters this loop at every step. Because each call's instructions rank it above every rule except error reporting, it can fix the design, the code and training settings, and the selection rule. The design instructions, for example, forbid numeric hyperparameters other than epochs and folds, but guidance that prescribes values, such as task weights of 4:1:1:1:1, takes precedence, and every training command of the three guided PXR runs carried those weights; Section 5.3 traces the other recommendations.

3.4 Validation and Model Selection

The task supplies a fixed fold assignment as a file, unless the guidance names another, and asks that every configuration be cross-validated on it. Followed, this keeps out-of-fold predictions row-aligned across configurations; the pipeline does not enforce it, so we read fold use from each run's artifacts. The task text also reserves a labelled development set, separate from the training folds, for selection and post-processing.

At the end of a run the selection sub-agent chooses among trained artifacts by their out-of-fold or development-set scores. The code can develop several candidates in parallel, each with its own design history and directory of training attempts, and the selection instructions define one way to combine them: a non-negative weighted stack fitted on out-of-fold predictions, kept only if it beats the best single candidate on the development set. These instructions name the PXR metric, the one task-specific item in the pipeline code, but stacking needs two candidates and each reported run had one. With one candidate the instructions ask for the best attempt, retrained on all training data when that is cheap. An ensemble inside the candidate, such as the PXR average over ten fold models, is built by the candidate's own code as the guidance directs, although several selection steps without guidance blended attempts of their one candidate.

To deliver, the selection sub-agent places a standalone inference script beside the chosen model files, instructed to load every fitted component rather than refit it, and runs the script to write and check the prediction file; in several runs without guidance the script instead looks up stored test-set predictions. Post-processing is fitted where the guidance directs, the PXR activity gate on the development set, and otherwise where the run chooses. No part of the pipeline is given the evaluation labels; what the logs show about web access is reported in Section 4.3.

3.5 Execution Control and Logging

A run normally ends when the controlling agent stops after selection. The instructions state an overall time budget, 5 h 30 m in the reported runs, and nothing enforces it. The code caps the number of setup, design and implementation calls, at 30 in the reported runs, refusing them past the cap with a message to run selection and stop, and gives each such call a time limit, 240 minutes; selection is not counted, and is told that limit but not held to it. A setup, design or implementation call that fails or times out returns the error or a timeout notice as its report, and the controlling session is resumed after an error result up to three times. The code also checks the paths given to the Write and Edit tools of the design, implementation and selection sub-agents, keeping them inside their assigned directories and out of the data directory, and filters some shell commands by pattern; these are pattern checks, not filesystem isolation, which comes from the experiment harness (Section 4.4). The instructions ask for a failed step to be reported rather than replaced with a fallback.

Step logs keep each tool call with its full input, including shell commands and web searches, but not the requests that programs make themselves, and a retried call can overwrite its earlier log. The PXR expert document asks for each training command line to be logged; Section 5.3 reads that log.

4 Experimental Setup

4.1 Benchmarks and Datasets

PXR activity. The OpenADMET PXR blind challenge concerns induction of the pregnane X receptor, which regulates drug-metabolising enzymes, chiefly CYP3A4, so compounds that induce it can cause drug-drug interactions (OpenADMET, 2026b). We chose it because domain knowledge has a concrete use on it and its difficulty is documented independently of us. Two thirds of its training compounds carry only single-concentration screening readouts, whose relation to potency the challenge teams' published reports describe (Swain et al., 2026; Castellanos et al., 2026); the evaluation compounds are analogues found by similarity search around 63 potent hits, the organisers report activity cliffs that proved challenging for models, and of the 100 entries on the final leaderboard the top 28 could not be statistically separated from the winner (OpenADMET, 2026b; Swain et al., 2026; Castellanos et al., 2026). The released split of the screening data (OpenADMET, 2026a) gives 4,139 training compounds with measured pEC_{50}, 8,135 more with only single-concentration screening readouts, a 253-compound development set with labels, and 260 evaluation compounds; the task files add 4,200 compounds with only a public lipophilicity value, from the MoleculeNet lipophilicity set. Published entries on the same 260 compounds give three references: 0.5631 for first place, whose team declared proprietary data, 0.5676 for the strongest entry built from public data, and 0.6043 for the last of the top 28 (OpenADMET, 2026b).

Hepatocyte clearance and plasma protein binding. The second and third tasks are ADMET regression benchmarks from TDC (Huang et al., 2021), with no auxiliary measurements and a public leaderboard on a fixed scaffold split, test whether the approach carries to properties of a different kind: Clearance_Hepatocyte_AZ, intrinsic clearance in human hepatocytes (848 training, 122 development and 243 test compounds), and PPBR_AZ, the percentage of a compound bound to plasma proteins (1,952, 279 and 559). Their leaderboard bests are 0.536 Spearman correlation and 7.440 mean absolute error (MAE). These published values, like the PXR ones, are external references, not baselines rerun under our conditions.

4.2 Baselines and Guidance Conditions

Every task is run with the task description alone and with the expert document added, three end-to-end runs each. On plasma protein binding the document is the only input that differs; on PXR it also names a ten-fold assignment where the task description names a five-fold one; on hepatocyte clearance the guided condition's task description also names an external microsomal assay and ships a frozen copy of it, so a third condition supplies that description and file without the document. Only these document-only comparisons attribute a difference to the document alone. Earlier runs with general modelling guidance were not matched to these batches and appear only in Appendix B.

On the scale of Section 3.2, the PXR document is dominated by prescriptions: it names the pretrained encoder, the auxiliary heads and their weights, the fold file, the seed and an activity-gating procedure, and asks for that configuration to be reproduced. The clearance and plasma protein binding documents share a general set of if-then rules, 78% and 85% of their text, written before any result from either task existed. They recommend concrete defaults, such as fine-tuning the same pretrained encoder, that a run may leave when its own diagnostics disagree, and each fixes a few further steps for its task, so they are less prescriptive than the PXR document but not open suggestions. The authors wrote all three from published challenge reports and earlier runs of this pipeline and froze each before the reported runs (Section 6). The task descriptions carry a few hints of their own, such as a target score on PXR and a suggested target transform on clearance. The PXR document as first written contained twenty-three statements that could only have come from the evaluation set, most of them attached to the activity gate. We replaced each with a prespecified instruction carrying no outcome, and the batch in Table 1 uses the rewritten document, which on the runs compared reproduces the original's predictions to 2×10^{-7} per molecule.

As a baseline for the pipeline itself, a single Claude Code session, driven through the same agent SDK and Claude Code version as the pipeline's agents, runs the PXR task with the same guidance, language model, instruction block, data and deliverable, and the same built-in tools except those through which it could start, schedule or hand work to another agent. It works alone, with no controlling agent, role instructions, call cap or per-call limit, under different budget controls. A second comparison gives the pipeline, the single agent, AIDE 0.2.2 and MLEvolve the same revised task text, guidance and language model (Section 5.5).

4.3 Evaluation Protocol and Metrics

Each task ships a labelled development set (for PXR the 253 released compounds, for the other two the official TDC validation split) that the task text reserves for selection and post-processing; the evaluation split is used only for scoring. The PXR runs use a supplied fold file: the clustering it derives from was chosen with reference to the development labels, and its ten-fold version does not fully preserve the original clusters. The PXR evaluation labels were never on the training machine; the TDC test labels are public, were held on the training host only outside the sandbox, and are forbidden by the task texts. The agents could reach the internet, and the design instructions urge web search for a named benchmark; no recorded tool call fetches evaluation labels, but one PXR run given the task description alone looked up the challenge's winning approach (Appendix B).

A run is one execution from task directory to prediction file, and nothing is combined by hand across runs. Each cell of Table 1 is a batch of three runs, all reported; within a batch, a run is replaced only when it delivered nothing to score. Which batch fills a cell is a separate choice, and for the PXR guided cell the reported batch replaced an earlier one after both had been scored (Section 6).

Each task is scored by its benchmark's metric: relative absolute error (RAE, lower is better) for PXR, the challenge's announced primary metric (OpenADMET, 2026c), Spearman correlation for clearance and MAE for plasma protein binding. RAE is the total absolute error divided by that of predicting the mean of the scored labels. The main-table scorers align each delivered file with the evaluation compounds and refuse one with missing compounds; the ablation scorer also checks, before opening any label, that per-fold predictions average to the delivered file and that post-processing matches the run's record. We score all 260 PXR compounds once, whereas the leaderboard values are means over bootstrap resamples. We report means, sample standard deviations, the standard error of each difference over runs and every individual run, without significance tests.

4.4 Implementation and Resource Settings

Every run uses the same language model, `claude-opus-5`, and one NVIDIA A100 80 GB GPU on a shared host, inside a filesystem sandbox that exposes the task directory and a private workspace and hides the launcher and every other run. Runs execute one at a time. Before launch the harness refuses to start if the task text or guidance differs from its registered version, and for 12 of the 21 pipeline runs it also records content hashes of the pipeline code and task files. It places the pretrained CheMeleon weights in each run's home directory in every condition. Every pipeline run has a cap of 30 setup, design and implementation calls, a limit of 240 minutes on each such call, one candidate set by the launcher, and a stated budget of 5 h 30 m; no hard stop enforced it in the reported pipeline

Task	Metric	Condition	Mean ± sd	Diff. (se)
PXR activity	RAE ↓	+ expert document	0.5609 ± 0.0036	0.0660 (0.0089)
		task description alone	0.6269 ± 0.0150	
		leaderboard first / best public-data	*0.5631 / 0.5676*	
Clearance_ Hepatocyte_AZ	Spearman ↑	+ expert document	0.5870 ± 0.0075	0.0172 (0.0066)
		+ external file only	0.5698 ± 0.0086	0.1173 (0.0081)
		task description alone	0.4525 ± 0.0110	
		leaderboard best	*0.536*	
PPBR_AZ	MAE ↓	+ expert document	7.137 ± 0.137	0.183 (0.156)
		task description alone	7.321 ± 0.232	
		leaderboard best	*7.440*	

Table 1: The pipeline under each input condition: 21 runs, three per cell, each scored on the prediction file it delivered; mean and sample standard deviation. The last column is the difference from the row below, from unrounded means, positive when the condition given more input scored better, with its standard error over runs. Adjacent conditions can differ in more than one input (Section 4.2). Italic rows are published leaderboard values, shown as external references. Individual runs are in Table 4.

runs, whereas the harness caps the single-agent run at 420 minutes. Every run of the prescriptive comparison records Claude Code version 2.1.218. The run records do not record the exact model build behind the model name.

5 Results and Analysis

5.1 Performance Across Molecular Prediction Tasks

Each of the 21 reported runs took its task from the data to a delivered prediction file with no modelling code written by hand. Table 1 gives every condition with the published leaderboard values, and Table 4 in Appendix C every one of the twenty-one runs. With guidance, the PXR mean of 0.5609 lies below the two strongest references, 0.5631 and 0.5676; two of its three runs are below first place and one is 0.0008 above it. Three runs of one pipeline support neither parity nor an advantage over a leaderboard whose top 28 entries could not be statistically separated from the winner (Castellanos et al., 2026). On clearance the guided mean of 0.5870 exceeds the TDC leaderboard best of 0.536. On plasma protein binding both conditions have lower error than the leaderboard best of 7.440.

5.2 Effect of Expert Guidance

The observed differences between conditions vary by task (Table 1): large on PXR, small on clearance and unresolved on plasma protein binding. On PXR every guided run is better than every unguided one, but the two arms also differ in the fold file read and were run eleven days apart; an earlier guided batch run interleaved with the unguided arm gives a difference of 0.0632 rather than 0.0660 (Section 6). On clearance only the guided and external-file conditions differ in the document alone; their difference of 0.0172 Spearman is between two and three standard errors, with slightly overlapping ranges. The larger 0.1173 between the external-file and unguided conditions reflects the task description and the assay file together and cannot be attributed to either. On plasma protein binding the fall of 0.183 in mean error is about one standard error and the ranges overlap, so three runs per condition neither establish the improvement nor show that it is absent.

5.3 Adherence to the Guidance

Four PXR recommendations become concrete settings: a pretrained encoder, issued as `--from-foundation CheMeleon` (Burns et al., 2025); five prediction heads, so a compound missing the scored label still contributes; four times the weight on the scored head; and an activity gate fitted on the development set only. The first three appear in the training commands of all three reported guided runs, and all three fitted the gate on the development set only. The guided runs also

Prescription	ΔRAE	Comparison
Retrained against one reused reference		
Pretrained initialisation	0.0878	retrained without it against the reused reference, ungated, one run at seed 42
Auxiliary supervision + supporting rows	0.0482	as above; removes the auxiliary targets and the rows carrying them together
217 computed descriptors	0.0233	as above
Primary-task weight 4x	0.0054	the designated equal-weight run against the same reused reference, ungated, seed 42
Recomputed inside the three existing expert runs, so these carry a spread		
Ten-fold prediction mean	0.0394	mean RAE of ten single models minus RAE of their mean prediction
Activity gate	0.0128	ungated minus gated RAE inside each run, gate frozen

Table 2: Component ablations of the delivered PXR model on all 260 blind-test compounds; positive favours the retained component. The upper group compares one run per arm, at seed 42, with a fixed reference and has no error bar, because no independent paired repeat was run; two further equal-weight runs were made only to check that the weight arm reproduces. Each difference is measured against the full guided model, not against the unguided runs, so the differences should not be summed or read as a decomposition of the gap between the guided and unguided conditions.

proceeded differently: each made four counted calls (setup, design, implementation, full training) followed by selection and revised no design, whereas each unguided run revised its design once after training and made six to nine counted calls.

The record also shows where execution departed from the document. One guided run outside Table 1 departed from the document in one line and scored 0.0049 worse than the two runs it is compared with, which followed the document and delivered near-identical predictions. Where the document asks for the training library's internal split for early stopping (Heid et al., 2024), that run built its own split file with the correct ratio but different rows, so early stopping chose a different epoch on each fold. We did not test whether this departure caused the gap.

5.4 Ablations of the Delivered PXR Model

Table 2 measures what each prescribed component contributes to the delivered PXR model, each by a comparison of its own; these are not ablations of the pipeline. Removing the pretrained initialisation costs most, then the auxiliary targets with their rows and the descriptors, while equal task weights cost little. The retrained arms use our own trainer, which reproduces the delivered command line.

5.5 Comparison with Other Agents

Guidance	System	Runs	RAE ↓	Individual runs
Prescriptive document	The pipeline	3	0.5609 ± 0.0036	0.5569, 0.5618, 0.5639
	Single coding agent	1	0.5624	0.5624
Modelling suggestions	The pipeline	1	0.6057	0.6057
	Single coding agent	1	0.6163	0.6163
	AIDE	3	0.6092 ± 0.0038	0.6088, 0.6132, 0.6056
	MLEvolve	3	0.6264 ± 0.0038	0.6299, 0.6268, 0.6224

Table 3: PXR activity for each executing system, all driven by `claude-opus-5`. The two blocks differ in task text and guidance and are not compared with each other; in the second, the pipeline and the single agent had a stated 330-minute budget and MLEvolve's first run a 180-minute stop.

Given the same PXR task, data and prescriptive guidance, the single coding agent delivered a complete run scoring 0.5624 (Table 3), inside the pipeline's range of 0.5569 to 0.5639. Before the activity gate its predictions score 0.5727 against 0.5737 for each of the pipeline's three, and they correlate at 0.994 to 0.996 with those of the three pipeline runs.

Once the document fixes the model, objective, validation and post-processing, what remains is implementation, which in its one run a single agent carried out as well as the pipeline. Under this guidance we therefore observe no accuracy advantage from dividing the work by role; as both systems choose their own next step, adaptive control is not tested.

Under a revised task text with no target score and guidance written as modelling suggestions, given identically to four systems (Table 3), the pipeline's one run, 0.6057, lies within the range of three AIDE runs and scores better than all three MLEvolve runs, the first of which had a shorter budget; the single agent's run, 0.6163, shared its host's CPUs with another job. This places the pipeline at the level of existing agents under open guidance, without showing that its organisation improves accuracy.

6 Discussion and Limitations

The evidence is small. Each condition has three runs, the single-agent baseline one, and each retrained component ablation one run at one seed against a reused reference, too few for significance tests or to resolve the plasma protein binding result. All runs come from one implementation and one language model, and our spreads describe run-to-run variation, not sampling variation of the evaluation sets.

The PXR evaluation is retrospective and shaped by hindsight. We ran on the released blind split after the competition closed, which is not the same as having competed even though the labels never entered the machine that trained. We wrote the guidance documents after the challenge (Section 4.2). Removing the twenty-three statements of Section 4.2 dealt with quantities that leaked, not with judgement shaped by knowing the outcome. The task description used by both PXR arms and by the single agent also stated a target RAE of about 0.57 and a range for top entries, so every condition was told roughly what a good score looks like. The agents could also search the web, and one of the three PXR runs given the task description alone received a search summary of a winning approach that names the encoder the guidance prescribes, so that arm was not kept from the published solutions (Appendix B).

The reported PXR batch was chosen after its scores were known. The earlier batch it replaced, under the original document, gives a difference of 0.0632 rather than 0.0660 and sits below the 0.5676 public-data entry but not below the 0.5631 first place; it was run interleaved with the unguided arm, the reported batch eleven days later.

The guidance comparisons are not uniform one-factor comparisons (Section 4.2). In the ablations, one prescribed descriptor is unbounded in molecule size, so its scale shifts between arms, and one arm produced a single extreme out-of-fold prediction that we have not traced to it (Appendix B). Among the clearance runs, one failed an isolation audit by listing another run's identifier and is kept (Appendix B).

Finally, the main guidance is largely prescriptive, so it tests how well a prescribed solution is carried out rather than whether the pipeline finds one; under open guidance the pipeline and the single agent have one run each and budgets differ, so that comparison places the systems but cannot rank them.

7 Conclusion

We presented a pipeline that automates molecular property model development from design to a delivered model, steered by written guidance instead of code, with the same code on three tasks. With guidance, PXR error was lower, clearance correlation higher where only the document changed and plasma protein binding unresolved, and on both ADMET tasks the guided runs beat the published leaderboard bests. Under guidance that names no model, its one run scored within the range of three AIDE runs and better than three MLEvolve runs, which does not establish an advantage for its organisation; the evidence covers only molecular property prediction.

REFERENCES

Jackson W. Burns, Akshat Shirish Zalte, Charlles R. A. Abreu, Jochen Sieg, Christian Feldmann, Miriam Mathea, and William H. Green. Deep learning foundation models from classical molecular descriptors. *arXiv preprint arXiv:2506.15792*, 2025. URL https://arxiv.org/abs/2506.15792.

Maria Castellanos, Jon Swain, and Hugo MacDermott-Opeskin. Don't look back in error: What we learned predicting PXR induction (part i). OpenADMET, 2026. URL https://doi.org/10.5281/zenodo.22103683.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024. URL https://arxiv.org/abs/2410.07095.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. SELA: Tree-search enhanced LLM agents for automated machine learning. *arXiv preprint arXiv:2410.17238*, 2024. URL https://arxiv.org/abs/2410.17238.

Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, Yifan Zhou, Xin Li, Jie Zhou, Liang He, Bo Zhang, and Lei Bai. MLEvolve: A self-evolving framework for automated machine learning algorithm discovery. *arXiv preprint arXiv:2606.06473*, 2026. URL https://arxiv.org/abs/2606.06473.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-Tabular: Robust and accurate AutoML for structured data. *arXiv preprint arXiv:2003.06505*, 2020. URL https://arxiv.org/abs/2003.06505.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, 2015.

Jose Carlos Gómez-Tamayo, Joris Tavernier, Roy Aerts, Natalia Dyubankova, Dries Van Rompaey, Sairam Menon, Marvin Steijaert, Jörg Kurt Wegner, Hugo Ceulemans, Gary Tresadern, Hans De Winter, and Mazen Ahmad. MolAgent: Biomolecular property estimation in the agentic era. *Journal of Chemical Information and Modeling*, 65(20):10808–10818, 2025. doi: 10.1021/acs.jcim.5c01938.

Esther Heid, Kevin P. Greenman, Yunsie Chung, Shih-Cheng Li, David E. Graff, Florence H. Vermeire, Haoyang Wu, William H. Green, and Charles J. McGill. Chemprop: A machine learning package for chemical property prediction. *Journal of Chemical Information and Modeling*, 64(1):9–17, 2024. doi: 10.1021/acs.jcim.3c01250.

Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W. Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2102.09548.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025. URL https://arxiv.org/abs/2502.13138.

Jingjie Ning, Xiaochuan Li, Ji Zeng, Chenyan Xiong, and Guolin Ke. Closed-loop Auto Research for molecular property prediction: Discovering and certifying generalizable improvements. *arXiv preprint arXiv:2606.22731*, 2026. URL https://arxiv.org/abs/2606.22731.

OpenADMET. PXR challenge train/test dataset. Hugging Face dataset, 2026a. URL https://huggingface.co/datasets/openadmet/pxr-challenge-train-test. CC BY 4.0, accessed 2026-09-24.

OpenADMET. OpenADMET PXR blind challenge. Hugging Face Space, 2026b. URL https://huggingface.co/spaces/openadmet/pxr-challenge. Challenge description and final activity leaderboard, accessed 2026-09-24.

OpenADMET. PXR challenge tutorial. GitHub repository, 2026c. URL https://github.com/OpenADMET/PXR-Challenge-Tutorial. Scoring code in evaluation/config.py, accessed 2026-09-24.

Yixin Ou, Yujie Luo, Jingsheng Zheng, Lanning Wei, Zhuoyun Yu, Shuofei Qiao, Jintian Zhang, Da Zheng, Yuren Mao, Yunjun Gao, Huajun Chen, and Ningyu Zhang. AutoMind: Adaptive knowledgeable agent for automated data science. *arXiv preprint arXiv:2506.10974*, 2025. URL https://arxiv.org/abs/2506.10974.

Jon Swain, Maria Castellanos, and Hugo MacDermott-Opeskin. It's the end of the PXR challenge as we know it (and I feel fine). OpenADMET, 2026. URL https://doi.org/10.5281/zenodo.21786099.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024.

Yizhen Zheng, Huan Yee Koh, Jiaxin Ju, Anh T. N. Nguyen, Lauren T. May, Geoffrey I. Webb, and Shirui Pan. Large language models for scientific discovery in molecular property prediction. *Nature Machine Intelligence*, 7(3):437–447, 2025. doi: 10.1038/s42256-025-00994-z.

A Use of Large Language Models

The authors used large language models to aid and polish the writing of this paper. The pipeline studied here is itself driven by large language models (Section 3).

B Supplementary Analyses

This appendix gives supplementary measurements referred to in the main text.

Generic-guidance runs. Earlier runs gave the pipeline general modelling guidance in place of a task-specific document. On PXR activity three such runs delivered RAE 0.6191, 0.6187 and 0.613, mean 0.6169, against 0.5609 for the expert document in Table 1. On hepatocyte clearance a long document carrying no task-specific thresholds was run three times, for Spearman 0.4694, 0.5047 and 0.4834 against 0.4839 for a run with no document. We do not report either set as the generic-guidance control because neither is condition matched: all six were run before, and separately from, the batches in Table 1. On clearance the three repeats also carry different run budgets written into the instruction itself, their no-document comparator differs in code commit as well as in guidance, and its 0.4839 is not the same condition as the 0.4525 in Table 1.

Scale of the unbounded descriptor. The descriptor is the information content of the molecular graph. It is unbounded and grows steeply with molecule size: 7.30e+02 for aspirin against 7.70e+16 for cyclosporin A. Read from the scalers the models themselves saved, on fold 7: in the full configuration that column's scale is 3.015e+29, under which that pool's 16060 molecules, excluding its single largest, occupy a standardised range of 4.0×10^{-5}, and cyclosporin A sits at -8.3×10^{-3}, the smallest value in the pool. In the arm that drops the rows carrying only auxiliary labels, one molecule reaching 3.6e+31 is dropped with them and the saved scale becomes 3.740e+12; that arm's own pool of 3725 molecules, again excluding its largest, occupies a range of 11.7 under its own

scaler, and cyclosporin A standardises to 2.1×10^4. The two ranges describe two pools under two scalers, not the same molecules measured twice. That arm predicts -437.7 for cyclosporin A against a true value of 3.27, where the other two arms predict 6.65 and 3.57. What is measured is a scale that moves by seventeen orders of magnitude between arms and an extreme prediction observed alongside it; we did not retrain with the column removed, so the causal step is not closed.

Isolation audit failure. While checking that it had left no training processes running, one run in the clearance external-file condition listed its own processes with a pattern that matched two of the study's, and the three-line output named another run's identifier and the paths of the waiter and launcher scripts. Our audit reports this as a failure on two tests. What it could have learned from that output is that the batch had two identifiers and where the machinery sits. What it could not learn from it is its own condition, since the launcher's arguments carry an identifier, a device and a log path and nothing else; any guidance text, since none was on any command line and no other run was executing; or anything about the compounds or the target. We keep the run and disclose it rather than removing it, and we note that we already knew its score when we decided to keep it.

C Individual Run Results

Task	Metric	Condition	Runs, in execution order
PXR activity	RAE ↓	+ expert document	0.5618 / 0.5639 / 0.5569
		task description alone	0.6420 / 0.6121 / 0.6267
Clearance_ Hepatocyte_AZ	Spearman ↑	+ expert document	0.5918 / 0.5908 / 0.5783
		+ external file only	0.5695 / 0.5614 / 0.5785
		task description alone	0.4432 / 0.4497 / 0.4647
PPBR_AZ	MAE ↓	+ expert document	7.124 / 7.280 / 7.008
		task description alone	7.088 / 7.321 / 7.553

Table 4: Every one of the 21 runs behind Table 1, in the order it executed. No value is dropped, trimmed or replaced by a best-of, and there is no outlier rule: with three runs per cell an outlier rule would be a licence to delete the inconvenient one. One run in the clearance external-file condition failed an isolation audit and is kept and disclosed rather than removed; Appendix B says what it saw.