AI RESEARCH PREFERENCE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

AI research agents (AIRAs) can drive machine learning experiments from proposal to implementation and evaluation. Yet their progress on frontier tasks is often throttled by the cost of evaluation, which can consume days of GPU time. When an agent can propose far more candidates than it can afford to run, progress depends on its *research preference*: how it allocates a fixed execution budget across many candidates. We introduce AI Research Preference Models (RPMs) that predict which candidate solution is most promising, without needing to run them all. We build two variants of RPMs from pretrained language models: an inference-only model that reasons over candidate plans, code, and previously executed solutions, and an agentic model that also runs small-scale pilot experiments. Integrated into the AIRA-dojo research agent and evaluated on the machine learning research benchmark AIRS-BENCH, the inference-only variant consistently provides gains over the unguided baseline with the agentic variant delivering the best overall performance. Across four LLM backbones (GPT-5, Gemma 4-31B, Qwen3.6-27B and Muse Glimmer), both variants improve the average normalized score over the baseline unguided agent. On average, they match the unguided agent's 24-hour performance in roughly 14 hours, using less than 60% of its execution budget, and they set new state-of-the-art results on three AIRS-BENCH tasks.

1 INTRODUCTION

Language model agents have advanced rapidly in domains such as mathematics, coding, and computer use, where candidate actions can be evaluated accurately and efficiently: a mathematical answer can be verified, a program run against a test suite, and a computer-use task checked against its target state. These evaluation functions provide the reward that lets agents iterate and improve, driving progress on benchmarks such as SWE-bench (Jimenez et al., 2024) and Terminal Bench (Merrill et al., 2026).

Progress has been slower for AI research agents (AIRAs) that autonomously propose, implement, and evaluate their own experiments, despite recent efforts such as AIDE (Jiang et al., 2025), AIRA-dojo (Toledo et al., 2025), AIRA_2 (Hambardzumyan et al., 2026) and MARS (Chen et al., 2026). Frontier machine learning research lacks cheap feedback: proposing or modifying candidate code can be quick, but executing it to train a model and measuring its performance can consume hours to days or decades of GPU time. Considering an agent can propose far more candidates than it can afford to run, the primary lever on research progress becomes *research preference*: deciding which research directions are promising enough to allocate compute budget to, and which directions to drop.

We address this allocation problem directly by introducing a dedicated **AI Research Preference Model (RPM)** into an AIRA. The RPM receives the full context of previously executed solutions, and uses it to select the most valuable candidate from a newly generated set of solutions to run next.

We summarize our contributions below:

- We introduce AI Research Preference Models, a framework for efficiently allocating compute to the most promising candidate solutions in AI research agents.
- We develop *Inference-only RPMs*, frozen-weight LLMs that reason over candidate plans, code and previous solutions, and demonstrate that it raises performance of the AIRA-dojo agent on AIRS-BENCH over the unguided baseline.
- We further propose *Agentic RPMs*, an extension of Inference-only RPMs with the ability to run small-scale pilot experiments, and demonstrate that it further raises performance over the Inference-only RPM and the unguided baseline.

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.

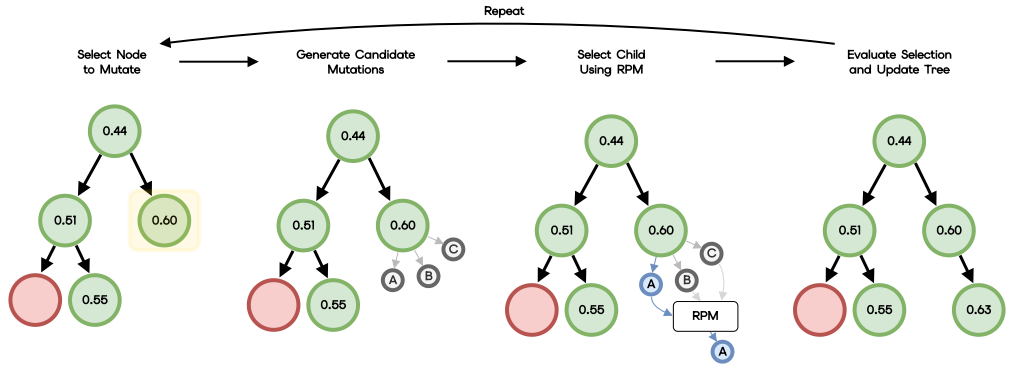

Figure 1: **Overview of an AI research agent with Research Preference Model (RPM) augmented child creation.** Each node represents a solution within the search tree, labeled with its evaluation score. The agent (1) selects a promising parent node from the active tree (here, parent selection is random for illustrative purposes), (2) generates a batch of candidate child solutions, (3) utilizes the RPM to select the most promising candidate, using the full context of previously executed solutions and (4) executes and scores only the selected candidate to expand the tree, to reduce the compute overhead of running unpromising candidates (Section 3.3).

- On AIRS-BENCH, agents equipped with our best RPMs yield new state-of-the-art results on two tasks and match the unguided agent's 24-hour performance in roughly 14 hours on average, using less than two-thirds of its execution budget (Fig. 2).

2 BACKGROUND

In line with the broader literature, we view an agent as a computer system that acts autonomously in some environment to achieve its design objectives (Wooldridge & Jennings, 1995). In our setting, an AI research agent acts by generating and executing code. The objective is to produce an artifact (such as model weights, an optimised code snippet, or an answer to a question) that, when evaluated by some task-specific reward function, achieves a high score.

2.1 AI RESEARCH AGENT BENCHMARKS

Recent benchmarks evaluate agentic large language models (LLMs) across complex, long-horizon workflows. For software engineering, popular suites like SWE-bench (Jimenez et al., 2024) assess an agent's ability to resolve real GitHub issues and modify multi-file codebases. Within the data science domain, benchmarks like MLE-bench (Chan et al., 2025) evaluate agent capabilities through structured machine learning engineering competitions.

We experiment in AIRS-BENCH (Lupidi et al., 2026), a suite of 20 machine learning tasks sourced from state-of-the-art (SOTA) papers. These tasks span diverse domains, including language modeling, mathematics and bioinformatics. AIRS-BENCH assesses agentic capabilities over the research lifecycle (including idea generation, experimentation, and iterative refinement) without providing baseline code. Each task has a problem specification, dataset, target metric, and SOTA score.

Here the agents act for 24 hours in a containerised environment, with access to a training dataset, test inputs, and an H200 GPU. The AIRA must write code to train a model, and produces a `submission.csv` with predictions on the test inputs. The agent may choose to guide its search using a validation score (e.g., from using cross-validation). However, the true test score (produced by comparing the `submission.csv` to the ground truth labels) is hidden from the agent.

To aggregate performance across heterogeneous task metrics, AIRS-BENCH defines a Normalized Score for agent a on task t:

$$\mathrm{NS}_t^a = \frac{\phi_t(s_t^a) - \phi_t(s_t^{\min})}{\phi_t(s_t^{\mathrm{sota}}) - \phi_t(s_t^{\min})}, \tag{1}$$

where $s_t^{\min}$ is the worst score observed across all agents, s_t^{sota} is the most recent public SOTA score as of the benchmark's publication, and ϕ_t is a non-linear log transform, defined as $\phi_t(s) = -\log_{10}(|s - s_t^{\mathrm{opt}}|)$ to properly weight exponential progress near optimal bounds (s_t^{opt}). Under this

metric, NS $= 0$ corresponds to the minimum baseline and NS $= 1$ to the public SOTA, while invalid or failed submissions receive a score of 0. The Average Normalized Score reported on this benchmark averages the Normalized Score uniformly across all tasks and seeds.

2.2 AI RESEARCH AGENT SCAFFOLDS

For completeness, we decompose an AIRA into an LLM backbone and an algorithmic *scaffold*. Whereas a backbone provides core knowledge and reasoning abilities, a scaffold defines the search strategy (e.g., simple linear loops or complex tree search), governing how candidates are generated and refined. Scaffolds like AIRA-dojo (Toledo et al., 2025) or Claude Code (Anthropic, 2025) can support different backbones, such as Claude Opus (Anthropic, 2026) or GPT-5 (Singh et al., 2025).

These candidates can be viewed as nodes within an evolving solution graph, each storing artifacts like code scripts, execution logs, and metric scores. An AIRA explores this graph by selecting a parent node, mutating it, and producing a child node. We characterize the core design space of these evolutionary search scaffolds along three axes: **Parent Selection:** The strategy used to identify which historical node is most promising to build upon next; **Child Creation:** The process of prompting an LLM to generate a candidate child solution from the selected parents. Prompts used to drive *child creation* are referred to as *operators*; **Final Solution Selection:** The criteria used to evaluate the accumulated bank of solutions and determine which to return as the final output.

Existing AIRAs implement these mechanisms differently. For example, MLGYM (Nathani et al., 2025) is a linear search scaffold, simplifying parent selection by always choosing the most recent node and relying on a single mutation operator for iterative refinement. In contrast, we build upon AIRA-dojo, a tree-search framework that uses greedy parent selection, mutating whichever previous node has the highest validation score. In child creation, AIRA-dojo employs three mutation operators tailored to distinct engineering phases: DRAFT, IMPROVE, and DEBUG. For final solution selection, AIRA-dojo submits the node that achieved the highest validation score across the entire search.

The AIRA-dojo loop produces an expanding bank of solutions, each paired with its evaluation score. It is over this growing set that a preference model can intervene, deciding which candidates are worth the expense of execution. It also serves as a valuable dataset for training and evaluating an RPM.

3 AI RESEARCH PREFERENCE MODELS

Frontier machine learning research lacks low-cost feedback: while *proposing* candidate solutions is fast, *executing* them by training a model can consume hours or days of GPU compute. Faced with this bottleneck, research progress heavily relies on predicting the value of pursuing research directions.

To address this, we introduce RPMs that guide experimental allocation within research agents. Through initial experimentation, we observed that LLMs are unreliable in forecasting absolute metrics. Consequently, an RPM reformulates experimental allocation as a preference ranking problem: ranking candidates to select the most promising paths before dedicating compute to pursuing them.

We explore RPMs leveraging varying ranges of test-time compute:

- **Inference-only RPMs** (Section 3.1): Rank candidate solutions using lightweight reasoning over search history and code diffs.
- **Agentic RPMs** (Section 3.2): Allocate additional compute to run rapid sandbox pilot experiments prior to ranking.

Designed as a scaffold-agnostic component, RPMs can interface with a wide variety of AIRA architectures. In this work, we investigate integrating RPMs into the AIRA-dojo evolutionary tree-search scaffold (Section 3.3). We target the child-creation phase, where creating a single child mutation is replaced with generating N candidate modifications in parallel and using an RPM-guided tournament to select the most promising solution before committing GPU compute.

3.1 INFERENCE-ONLY RPMS

The "LLM-as-a-Judge" paradigm (Zheng et al., 2023) demonstrates that LLMs can rank technical solutions with reasonable fidelity via intuition and reasoning. Motivated by this, we experiment by

querying pretrained LLMs as an inexpensive RPM for selecting between ideas. To understand how visibility into the AIRA's search space affects the RPM's selection quality, we vary the count of previously explored solutions visible to the RPM, and the count of suggestions it selects between.

Furthermore, to improve the RPM prompt, we leverage MIPROv2 from DSPy (Opsahl-Ong et al., 2024), a widely adopted baseline for robust prompt optimization. This generated an instruction set that directed the RPM to conduct more structured analyses of each solution, while remaining tolerant of minor, fixable issues. Further details of this prompt are provided in Appendix A.1.

3.2 Agentic RPMs

Motivation A core practice in software engineering, and machine learning research is rapid prototyping. To comprehensively understand and validate the potential or the feasibility of a novel idea, researchers tend to quickly run small-scale pilot experiments before launching a full-volume large-scale experiment. Inspired by how pilot experiments inform the possible outcome and assist decision making, we develop Agentic RPMs where an agent can use multiple predefined tools in a sandbox environment to conduct pilot experiments before making a decision.

Agentic Workflow A pilot experiment is conducted via multi-turn interaction between the RPM and the environment, interleaving chain-of-thought reasoning, tool calling, and environment feedback (Yao et al., 2022). The sandbox environment for the Agentic RPM is identical to that of the AIRA, including access to a single H200, training data, unlabelled test data and pre-installed python packages. The agentic RPM interacts with this environment using a set of tools: `python`, `bash`, and `submit_solution`. The `python` and `bash` tools can execute arbitrary code. Based on the pilot experiment, the agent can summarize and submit its experimental findings with `submit_solution`. Notably, the agent need only submit the summary and analysis of the *pilot* experiment, not complete final solution selection. The `submit_solution` tool will return the remaining time budget. If this budget surpasses a defined threshold, the agent is prompted to run further experiments, maximizing its use of time and computation. Finally, the candidate solution, the task description, and all submitted pilot experiment findings are input to an LLM, which selects the most promising solution. Full implementation details on agentic RPM are provided in Appendix B.

3.3 AIRA Integration

While an RPM could be applied multiple stages of an agent's search scaffold, we focus on *child creation*, as illustrated in Fig. 1. In an evolutionary AIRA, child creation encompasses two sub-steps: *candidate creation* and *candidate selection*. We modify AIRA-dojo by expanding candidate creation: a chosen parent node is mutated by applying operators N times, independently in parallel, yielding N unexecuted child candidate solutions. During candidate selection, an RPM evaluates these N candidates alongside historical trajectory context, conducting pairwise comparisons in a tournament knockout structure to select the single candidate for full execution.

To ground each comparison in the search so far, we also provide the RPM selected context from the tree. Context nodes are pulled via a BFS traversal of the already-explored tree starting from the parent, collecting up to K non-buggy nodes from earlier in the search; each of these previously-evaluated solutions is presented alongside the validation score it obtained.

4 Experimental Setup

4.1 End-to-End Evaluation

We integrate RPMs into child-creation for AIRA-dojo, and evaluate it against the 20 publicly released AIRS-Bench tasks with text and tabular modalities. Following the AIRS-Bench evaluation protocol, each AIRA-dojo evaluation is repeated for 10 seeds. To avoid the generalization gap present in AIRA-dojo, we integrate the Hidden Consistent Evaluation (HCE) protocol from AIRA_2. To contextualize these results, we compare against the default No-RPM baseline, where child selection is uniformly random among candidates; equivalent to vanilla AIRA-dojo in expectation.

Both the child candidate generation operator and the RPMs use a the same LLM backbone to ensure that observed improvements are driven by the framework, rather than a stronger selection backbone.

We conduct our evaluations with Qwen3.6-27B (Qwen Team, 2026), GPT-5 (Singh et al., 2025), Gemma 4-27B (Abd et al., 2026) and Muse Glimmer (Meta Superintelligence Lab, 2026), a set of high-performing proprietary and open-weights models for code reasoning.

4.2 Offline Evaluation

Our full end-to-end (E2E) evaluations, detailed in Section 4.1, are computationally expensive, taking 200 H200 GPUs for 24 hours each. To enable quicker insight and guide development of our RPMs before full E2E evaluations, we produce an offline evaluation dataset compiled from AIRA-dojo runs on a set of 40 AIRS-Bench-like tasks from image, video and audio modalities. We separate development and evaluation sets by modality to avoid contamination. These AIRA-dojo runs followed the standard AIRS-Bench evaluation protocol from section 4.1, using gpt-oss-120b (OpenAI, 2025), GPT-4o (Hurst et al., 2024) and CWM (Copet et al., 2025) as the LLM backbone.

From this data, we extract 1,000 sibling node pairs, complete with their plans, code, and search tree history. We exclude near-ties, with a normalized test-metric gap below 0.01, to prevent run-to-run metric noise from confounding the evaluation signal. The RPM is evaluated on its accuracy in selecting the node with the highest test score in its subtree. This metric ensures that RPMs look past immediate performance, explicitly rewarding candidates with strong long-term potential, though we acknowledge that the original greedy search policy biases performance towards nodes with stronger early scores that are selected more frequently for mutation. Random selection gives a baseline of 50% accuracy. For offline evaluations, GPT-5 (Singh et al., 2025) is used as the backbone for the RPMs.

5 Results

5.1 End-to-End Evaluations

We present the E2E performance of integrating our RPMs within AIRA-dojo, in Fig. 2. In these evaluations, AIRA-dojo generates 15 child candidate suggestions per operator step, based on the offline finding that larger candidate pools consistently improve selection performance (Section 5.2.1). The Inference-only RPM evaluates these pairs using the "LLM-as-a-judge" reasoning prompt in Section 3.1, including the scores of historical nodes from the search tree. We only deploy the Agentic RPM, detailed in Section 3.2, for the Draft and Improve operators to balance overall compute budgets; we revert to random selection during Debug steps.

As shown in Fig. 2, our methods balance decision quality and compute time in fundamentally different ways. As it incurs no candidate-execution cost, the Inference-only RPM provides immediate and steady gains over the No-RPM baseline. Conversely, the Agentic RPM exhibits a slow rise early in the run; its small-scale proxy experiments require additional per-step compute at every step, slowing early progress. However, this more rigorous evaluation eventually triggers a sharp performance acceleration, ultimately exceeding both the Inference-only RPM and the No-RPM baseline. These trends hold across all evaluated LLM backbones, with the sole exception of the Inference-only RPM using Muse Glimmer, which only matches the baseline because a single seed on the SICK textual similarity task (Marelli et al., 2014) produced a buggy solution that degraded its average performance.

To further contextualize these results, we also compare our E2E evaluations with Qwen3.6-27B against Test and Validation Oracles, where all candidates are executed per step, and the best is selected. Both oracles outperform all other methods, however we observe that AIRA-dojo with the Agentic RPM closes the gap significantly. Further details are provided in Appendix C.1.

State-of-the-Art Breakthroughs These guided search capabilities translate directly to three new state-of-the-art (SOTA) milestones on established benchmarks, to the best of our knowledge. On WinoGrande (Sakaguchi et al., 2020), AIRA-dojo with the Agentic RPM and Qwen3.6-27B backbone achieves an accuracy of 94.1%, surpassing the previous agentic SOTA of 90.4% reported by Hambardzumyan et al. (2026). During this run, the agent fine-tunes a Qwen2.5-14B-Instruct model (Qwen Team, 2024) via LoRA on data with shuffled labels, then averages prediction logits across original and shuffled label orderings at inference to eliminate position bias. Similarly, on SVAMP (Patel et al., 2021), AIRA-dojo with an Inference-only RPM and Qwen3.6-27B backbone reaches 95.7% accuracy, beating the prior human SOTA of 94.2% (Zhong et al., 2024). Here, the agent designs few-shot prompts that instruct the model to cleanly isolate relevant numerical data from distracting

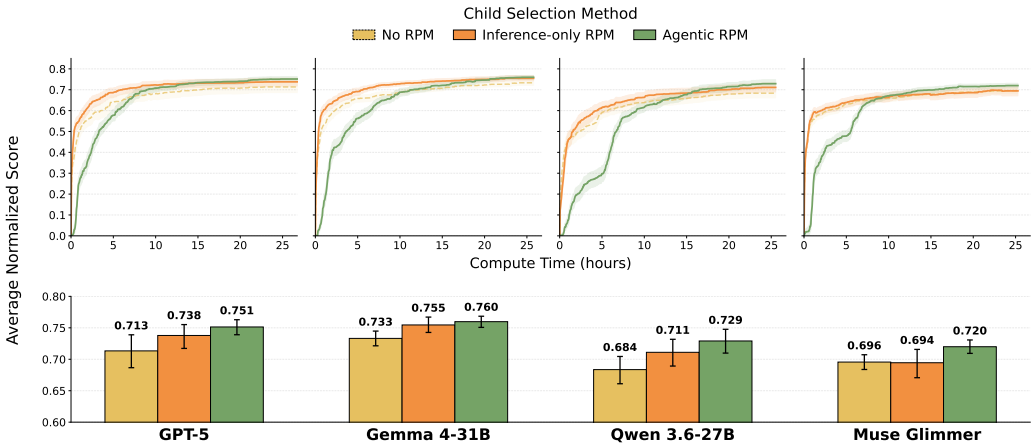

Figure 2: **Evaluation of RPM-augmented AIRA-dojo.** Average normalized scores over time (top) and final performance (bottom) for AIRA-dojo with RPM-augmented child selection on AIRS-BENCH, with 95% confidence intervals. Across all models, the Inference-only RPM yields steady gains over the No-RPM baseline. While per-step proxy experiments slow its early trajectory, the Agentic RPM eventually significantly surpasses other methods.

context, then generates ten independent reasoning paths by sampling Qwen2.5-7B-Instruct with increased temperature, and resolves the final prediction using a majority vote. Finally, on SQuAD 1.1 (Rajpurkar et al., 2016), AIRA-dojo with an Inference-only RPM and GPT-5 backbone reaches 91.0% exact match, surpassing the previous human SOTA of 90.6% (Jun et al., 2022). The agent fine-tunes DeBERTa-v3-large using multi-answer span supervision, layer-wise learning-rate decay, R-Drop, adversarial training, and exponential moving averages, and aggregates span predictions across overlapping context windows at inference.

Research Efficiency Both preference models accelerate search by reaching the No-RPM baseline's 24-hour score in less time. Averaged across the four model backbones, the Inference-only and Agentic RPMs match this baseline in 14.20 and 14.06 hours, respectively, using less than 60% of the compute budget. Individual model speedups range from $1.45\times$ to $2.97\times$, with the GPT-5 Inference-only RPM using only 8.09 hours. The only exception is the Inference-only RPM with Muse Glimmer, which finishes marginally below the baseline, after initially matching the baseline score in 21.68 hours.

Impact of Selection Quality To confirm how decision quality drives performance, we retrospectively analyze the selection advantage of each RPM throughout the Qwen3.6-27B runs. Selection advantage measures the average difference between the chosen candidate's score and the overall batch mean. As expected, random selection yields an advantage of roughly 0.0, the Inference-only RPM achieves significantly higher selection quality, and the Agentic RPM yields the highest advantage on average. We find a moderate positive correlation between selection advantage and final normalized score (Pearson $r = 0.55$, Spearman $\rho = 0.56$), confirming that more accurate candidate selection translates to better AIRA performance. Further details are provided in Appendix C.2.

5.2 Offline Evaluation and Further Analysis

We guide our development and research into RPMs using the offline evaluation detailed in Section 4. Full 24-hour online runs are too computationally expensive for quick iteration, and thus this offline setup allows us to tune models, analyze their core behaviors, and derive clear trends.

We find that Inference-only RPMs are surpassed in predictive accuracy by our Agentic RPMs, with both exceeding the random baseline. This matches the trends ultimately seen in E2E evaluations.

5.2.1 Inference-only RPM

Inference Scaling We structure our offline analysis around three key dimensions that dictate the RPM's visibility and analytical capacity: *context size* (the number of historical code solutions and validation scores provided from the tree), *suggestion count* (the number of candidate child solutions), and *reasoning budget* provided to the LLM.

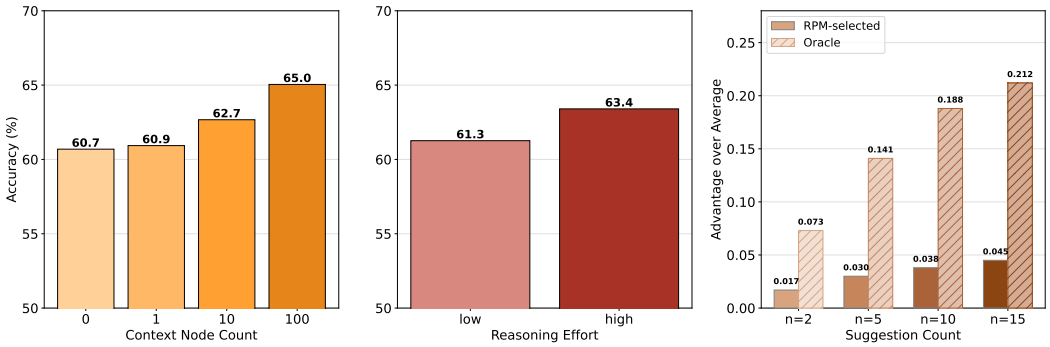

Figure 3: **Offline evaluation of the Inference-only RPM's scaling properties.** Increasing search-tree context provided gives improved predictive accuracy (left). Allocating a higher reasoning budget yields better selection (middle). Expanding the candidate suggestion pool reliably improves both the oracle-selection limit and the RPM's selection performance (right), measured by the average difference between the selected solution's score and the average score of all candidates.

Offline testing reveals three scaling behaviors: *context scaling* demonstrates that feeding the RPM more historical tree nodes, with validation scores, consistently improves selection judgment; *suggestion scaling* shows that expanding the candidate pool increases the selection advantage; *reasoning scaling* reveals that increasing the reasoning budget for the LLM judge improves its selection accuracy.

These results motivated using a large suggestion count (15 suggestions) and to maximize the context nodes provided (by providing the maximum amount that can fit within the LLM's context window), and high reasoning budget parameters for the RPM in our end-to-end evaluation.

Ensembling We evaluate three frontier models, GPT-5 (Singh et al., 2025), Claude Opus 4.8 (Anthropic, 2026) and Gemini 3.1 Pro (Google DeepMind, 2026), individually and via two aggregation techniques: a majority vote; and an LLM-Arbiter that ingests the reasoning traces of all three models before making a final decision. Whereas individual models attain between 64.66% to 67.44% accuracy, aggregation improves performance: majority vote to 68.04%, and the LLM-Arbiter to the highest achieved offline accuracy of 69.35%. A full breakdown of these configurations and their corresponding results is compiled in Appendix A.2.

Reasoning Analysis To get a better understanding of the RPM's decision-making, we analyze its generated reasoning traces. We find that referencing prior evidence, correctness of implementation or the pretrained backbone in the justification yields higher selection accuracy compared to when these are omitted. We also find, that citing more unique values from historical context also yields improved selection accuracy. Further details on the reasoning analysis are provided in Appendix A.3.

5.2.2 Agentic RPM

Compute Scaling We analyze how agentic RPMs scale with execution time limits on our offline benchmark. Increasing their available budget consistently improves selection accuracy, scaling from 78.52% under a 5-minute constraint, to 82.78% at 30 minutes and 84.02% with a 4-hour allocation. This demonstrates that giving agentic RPMs more time to run, observe, and debug directly translates to higher-fidelity estimations. However, given the pilot-experiment agent shares its time budget with the AI research agent, a 4-hour run of the Agentic RPM is prohibitively expensive and the marginal returns obtained from increasing the time budget are minimal. Therefore, we use a time budget of 5 minutes for E2E evaluation.

Proxy Strategy Analysis To understand agent behavior for identify promising candidates, we use keyword matching to analyze generated Python code. Since full execution is impossible within 5- or 30-minute limits, we find that agents simplify validation to use a single-split (though this is less common in the 30-minute agent), subsample data sampling, and adapt training (e.g., by removing ensembles, reducing epochs, and lowering batch sizes), as shown in Fig. 4. We note that our prompt

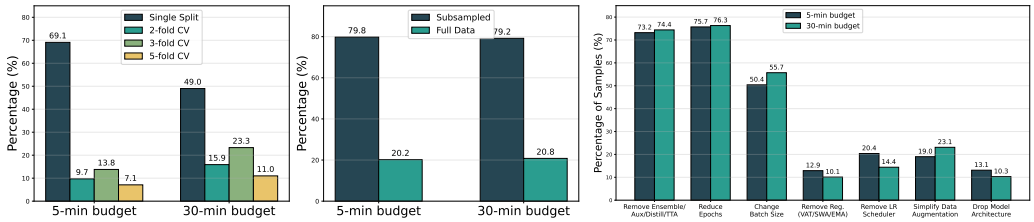

Figure 4: **Agentic RPM's proxy experiment strategies.** The Agentic RPM frequently runs simplified versions of the original candidates. Agents save time by running reduced cross-validation split counts (left) and sub-sampling the training data (middle). Both agents employ similar training adaptations, such as removing ensembling, reducing epochs and lowering the batch size.

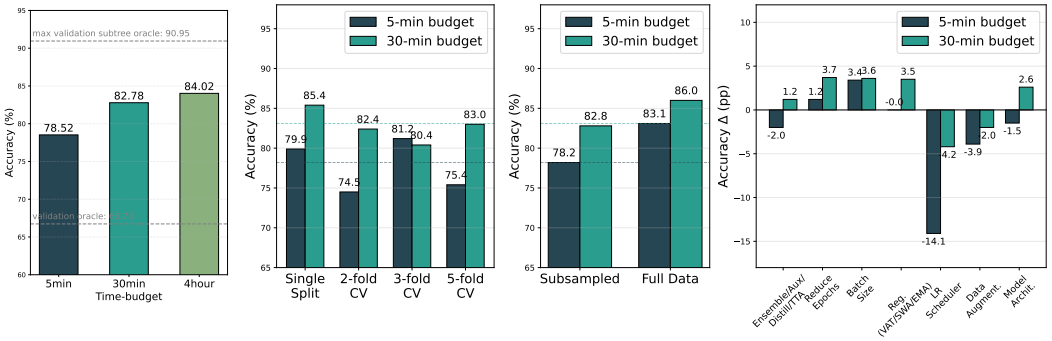

Figure 5: **How the Agentic RPM's strategy affects performance.** Increasing the time budget leads to better performance, with diminishing returns (leftmost). Using more cross-fold validation splits does not correlate with better selection accuracy (middle left). Using full data improves over sub-sampling data (middle right). Most training adaptations improve selection accuracy, except for removing the learning rate scheduler and simplifying the data augmentation approach (rightmost).

provide some hints on possible proxy strategies for the pilot experiment, as shown in Appendix Fig. 13, so some of these strategies are not wholly developed by the RPM.

Our behavioral analysis in Fig. 5 reveals three key insights: (1) multi-fold cross-validation does not consistently improve accuracy due to frequent execution timeouts; (2) evaluating on full datasets provides significantly more reliable quality estimations than using subsampled data; and (3) removing learning rate schedulers or data augmentations triggers severe performance drops, as the underlying AIRA-dojo tasks often rely on these exact training-level optimizations to succeed.

6 Related Work

AI research agents and automated ML research. Early work in automating machine learning research with LLMs used models to generate ideas, write code, run experiments, analyze results, and draft papers (Lu et al., 2026; Yamada et al., 2025; Schmidgall et al., 2025; Baek et al., 2025; Ren et al., 2025; Zheng et al., 2025). In some cases, AI research agents are treated as search policies over a tree of candidate solutions, where a node corresponds to an executed or proposed solution and edges correspond to mutations, refinements, or other improvement operators (Toledo et al., 2025; Hambardzumyan et al., 2026; Lange et al., 2026). This formulation directly motivates our work: RPMs target the decision problem induced by such search trees, predicting which candidate solution to execute before committing expensive compute. Other recent works improve the agent scaffold by adding modular search, targeted refinement, ideation agents, multi-agent specialization, or improved code interfaces (Nam et al., 2025; Chen et al., 2026; Zhang et al., 2026; Li et al., 2024; Yang et al., 2024). While these works improve agents, scaffolds, or benchmarks, none of them address the dominant cost in this setting, namely *executing* the candidate solutions that the search proposes.

Reducing the cost of evaluation. In reinforcement learning (RL), world models learn environment dynamics and can be used for planning (Schrittwieser et al., 2020) or policy learning from model-generated rollouts (Sutton, 1990; Ha & Schmidhuber, 2018; Hafner et al., 2023) without requiring

expensive environment interaction. At the meta-level, Metz et al. (2022) favor task configurations predicted to expedite optimizer meta-training, Oh et al. (2020); Goldie et al. (2024) each meta-learn over simple RL environments before transferring to more complex ones, Goldie et al. (2025) use a distillation objective as opposed to online evaluation due to the time-cost of training models during meta-learning, and Wolf et al. (2026) present model-based meta-learning for algorithm discovery. While these works share our goal of reducing expensive evaluation, none introduce preference ranking over a solution's code to decide which *proposed algorithm* to evaluate next.

Preference, judge, and reward models. Reward models trained from human preferences are central to reinforcement learning from human feedback (Xue et al., 2024; Stiennon et al., 2020; Ouyang et al., 2022), and their objectives frequently connect to paired-comparison and ranking formulations (Bradley & Terry, 1952; Liu, 2009). Related preference supervision can also be constructed from expert trajectories without reward-model training or online environment rollouts (Chen & Yuille, 2026). Furthermore, strong LLM judges have been shown to approximate human preferences on open-ended responses, subject to known biases (Zheng et al., 2023). Recent work derives continuous scores from scoring-token logits to reduce ties and rank candidates without additional verifier training (Kwok et al., 2026), while Wang et al. (2026) argue that fixed coding-agent rewards can saturate as generators improve. These methods mostly evaluate completed responses, trajectories, or agent outputs, whereas the RPM ranks candidate solutions before expensive full training and evaluation.

Closest to our setting, Zheng et al. (2026) predict pairwise preferences between unexecuted ML solutions, but condition on a separately prepared, static data report and the candidate code snippets, rather than on nodes from the surrounding search tree. The RPM instead grounds comparisons in the live search tree and the validation scores of executed solutions. Goldie et al. (2026) propose training a judge or reward model to select promising leaves in tree-search research agents, but do not implement the idea. The RPM realizes this proposal and extends it: where agentic verifiers probe code or user interfaces (Wang et al., 2026), agentic RPMs run small-scale *pilot ML experiments*, partial training and evaluation runs, and select based on their measured results.

7 Limitations

While our evaluations assume that LLM inference calls will incur a negligible cost, current LLM inference incurs real-time latency in practice. For the Inference-only RPM which uses a self-hosted Qwen3.6-27B, inference latency totals 0.660 hours per 24-hour end-to-end run. Adjusting for this yields a normalized score of 0.708 in 23.34 hours, a negligible drop from 0.711. Nevertheless, latency and monetary overhead will vary by infrastructure and LLM.

We limit our main RPM integration to the *child creation* stage, and report only initial results of RPMs for *final solution selection* in Appendix D. There, we observe no significant improvement over validation-based selection, given the HCE protocol's strong validation-test generalization (Hambardzumyan et al., 2026). We leave further RPM integration variations to future work.

We describe the RPM as scaffold-agnostic in principle, but demonstrate it only for AIRA-dojo, on a single benchmark suite. While we cannot foresee reasons why RPMs would not transfer, portability to other scaffolds and backbones is left to future work.

8 Conclusion

An AI research agent can propose a candidate solution far faster than it can evaluate it. We introduced the **AI Research Preference Model (RPM)**, which ranks unexecuted candidates so that the agent can direct its execution budget toward promising candidates, avoiding requiring the model to forecast absolute outcomes. Our results show that RPM-guided candidate selection improves *both* final performance and research efficiency without changing the agent backbone or search operators. Furthermore, we demonstrate that candidate selection is a useful target for test-time compute: agents can invest computation into deciding which candidates are worth executing.

AI research often lies on the less favorable side of the asymmetry of verification: candidate solutions are easy to generate, but hard to verify (Wei, 2025). RPMs are designed to navigate this. While they do not reduce the cost of a single verification, they reallocate it, spending their budget on preferred candidates to reach better solutions sooner. By integrating this allocation into research agents, we hope to encourage work into candidate solution *selection*, in addition to candidate solution *design*.

Ethics Statement

By improving the efficiency of AI research agents, our work helps democratize access to such systems and support faster scientific experimentation. We hope these advances accelerate scientific discovery, though we recognize these systems could be misused for malicious purposes, a risk inherent to autonomous research systems broadly. Thus, we emphasize the importance of appropriate human oversight and safeguards when deploying such systems.

Reproducibility Statement

We plan to release all of the code involved in our study upon institutional approval. The implementation details of Inference-only RPM and Agentic RPM are elaborated on in Sections A and B, respectively, including the exact prompts used and the full Agentic RPM workflow pseudocode and hyperparameters (Appendix B).

AI Use Statement

In this work, we used generative AI tools for both coding assistance and paper writing assistance: to aid in codebase development, iterate on LLM prompts, improve paper readability, identify related literature, summarize the findings of the ensembling logic in Section A.2, and aid reasoning trace analysis in Section A.3. Generative AI models were used within the AIRA-dojo scaffold to generate the search trajectories from which the offline dataset is composed, as detailed in Section 4.2. Generative AI models were also used as the backbone for the RPMs and the AIRA-dojo scaffold in our online and offline evaluations.

References

Gemma Team Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, C. Brick, Victor Cuarbune, Michelle Cas-bon, Mayank Chaturvedi, et al. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Anthropic. Claude code overview. https://code.claude.com/docs/en/overview, 2025.

Anthropic. Introducing claude opus 4.8, 2026. URL https://www.anthropic.com/news/claude-opus-4-8.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative research idea generation over scientific literature with large language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6709–6738, Albuquerque, New Mexico, apr 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.342. URL https://aclanthology.org/2025.naacl-long.342/.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324, 1952. doi: 10.2307/2334029.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *arXiv.org*, 2025. doi: 10.48550/arXiv.2410.07095. URL https://openreview.net/forum?id=6s5uXNWGIh.

Jie-Feng Chen, Bhavana Dalvi, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. MARS: Modular agent with reflective search for automated AI research. In *arXiv.org*, 2026. doi: 10.48550/arXiv.2602.02660. URL https://openreview.net/forum?id=qWE13uu99a.

Yixiong Chen and Alan Yuille. Agentic-DPO: From imitation to agentic policy optimization on expert trajectories, 2026. URL https://arxiv.org/abs/2607.10601.

Fair CodeGen team. Jade Copet, Quentin Carbonneaux, Gal Cohen, Jonas Gehring, Jacob Kahn, Jannik Kossen, F. Kreuk, Emily McMilin, Michel Meyer, Yuxiang Wei, et al. CWM: An open-weights LLM for research on code generation with world models. *arXiv.org*, 2025. doi: 10.48550/arXiv.2510.02387.

A. Goldie, Zi-Lin Wang, Jaron Cohen, J. Foerster, and Shimon Whiteson. How should we meta-learn reinforcement learning algorithms? In *arXiv.org*, 2025. doi: 10.48550/arXiv.2507.17668.

A. Goldie, Zi-Lin Wang, A. Hayler, Deepak Nathani, Edan Toledo, Ken Thampiratwong, Aleksandra Kalisz, Michael Beukman, Alistair Letcher, Shashank Reddy, et al. Procedural generation of algorithm discovery tasks in machine learning. In *arXiv.org*, 2026. doi: 10.48550/arXiv.2603.17863. URL https://openreview.net/forum?id=0Mvm3lqLjF.

Alexander D. Goldie, Chris Lu, Matthew T. Jackson, Shimon Whiteson, and Jakob N. Foerster. Can Learned Optimization Make Reinforcement Learning Less Difficult? In *Advances in Neural Information Processing Systems 37*, volume 37, pp. 5454–5497. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-0177.

Google DeepMind. Gemini 3.1 pro: A smarter model for your most complex tasks, 2026.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Neural Information Processing Systems*, volume 31, 2018.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv.org*, 2023. doi: 10.48550/arXiv.2301.04104.

Karen Hambardzumyan, Nicolas Baldwin, Edan Toledo, Rishi Hazra, Michael Kuchnik, Bassel Al Omari, Thomas Simon Foster, Anton Protopopov, Jean-Christophe Gagnon-Audet, Ishita Mediratta, et al. Aira_2: Overcoming bottlenecks in ai research agents, 2026. URL https://arxiv.org/abs/2603.26499.

OpenAI Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o System Card, 2024. URL https://cdn.openai.com/gpt-4o-system-card.pdf. Accessed: 2024-06-07.

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.org*, 2025. doi: 10.48550/arXiv.2502.13138.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024. doi: 10.48550/arXiv.2310.06770. URL https://openreview.net/forum?id=VTF8yNQM66.

Changwook Jun, Hansol Jang, Myoseop Sim, Hyun Kim, Jooyoung Choi, Kyungkoo Min, and Kyunghoon Bae. Anna: enhanced language representation for question answering. In *Proceedings of the 7th Workshop on Representation Learning for NLP*, pp. 121–132. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.repl4nlp-1.13.

Jacky Kwok, Shulu Li, Pranav Atreya, Yuejiang Liu, Yixing Jiang, Chelsea Finn, Marco Pavone, Ion Stoica, and Azalia Mirhoseini. LLM-as-a-Verifier: A general-purpose verification framework, 2026. URL https://arxiv.org/abs/2607.05391.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *arXiv.org*, 2026. doi: 10.48550/arXiv.2509.19349. URL https://openreview.net/forum?id=lKEdGCoDNC.

Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, et al. AutoKaggle: A multi-agent framework for autonomous data science competitions, 2024. URL https://arxiv.org/abs/2410.20424.

Tie-Yan Liu. Learning to rank for information retrieval. *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 3(3):225–331, 2009. doi: 10.1007/978-3-642-14267-3.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919, 2026. doi: 10.1038/s41586-026-10265-5.

Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, et al. Airs-bench: a suite of tasks for frontier ai research science agents. *arXiv (Cornell University)*, 2026. doi: 10.48550/arxiv.2602.06855.

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (eds.), *International Conference on Language Resources and Evaluation*, pp. 216–223. European Language Resources Association (ELRA), 2014. doi: 10.63317/39qdhuevzbqa.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, H. Raj, I. Bercovich, Lin Shi, J. Shin, Thomas Walshe, Estefany Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *arXiv.org*, 2026. doi: 10.48550/arXiv.2601.11868. URL https://openreview.net/forum?id=a7Qa4CcHak.

Meta Superintelligence Lab. Muse glimmer 30b, 2026.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up, 2022. URL https://arxiv.org/abs/2211.09760.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan O Arik, and Tomas Pfister. MLE-STAR: Machine learning engineering agent via search and targeted refinement. In *Advances in Neural Information Processing Systems 38*, pp. 129290–129310. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2025. doi: 10.52202/085713-3889. URL https://openreview.net/forum?id=vS1M06Px6u.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bash-lykov, A. Menon, Vincent Moens, A. Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, et al. MLGym: A new framework and benchmark for advancing ai research agents, 2025. URL https://arxiv.org/abs/2502.14499.

Junhyuk Oh, Matteo Hessel, Wojciech M. Czarnecki, Zhongwen Xu, H. V. Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Neural Information Processing Systems*, 33:1060–1070, 2020.

OpenAI. Introducing gpt-oss, 2025. URL https://openai.com/index/introducing-gpt-oss/. Accessed: 2026-08-12.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 9340–9366. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.525.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, S. Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Neural Information Processing Systems*, pp. 27730–27744. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-2011.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *North American Chapter of the Association for Computational Linguistics*, pp. 2080–2094. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.NAACL-MAIN.168.

Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1264.

Shuo Ren, Can Xie, Pu Jian, Zhenjiang Ren, Chunlin Leng, and Jiajun Zhang. Towards scientific intelligence: A survey of llm-based scientific agents, 2025. URL https://arxiv.org/abs/2503.24047.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and E. Barsoum. Agent laboratory: Using LLM agents as research assistants. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Conference on Empirical Methods in Natural Language Processing*, pp. 5977–6043. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-emnlp.320.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. doi: 10.1038/s41586-020-03051-4.

Aaditya K. Singh, A. Fry, Adam Perelman, A. Tart, A. Ganesh, Ahmed El-Kishky, A. McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv*, 2025.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *Neural Information Processing Systems*, 2020.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. In *SGAR*, volume 2, pp. 160–163. Association for Computing Machinery (ACM), 1990. doi: 10.1145/122344.122377.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, et al. AI research agents for machine learning: Search, exploration, and generalization in MLE-bench. In *Advances in Neural Information Processing Systems 38*, pp. 39717–39756. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2025. doi: 10.52202/085713-1186. URL https://openreview.net/forum?id=RwfrdKSgCE.

Bing Wang, Chenlong Zhang, Dayiheng Liu, Jiajun Zhang, Jia-Wei Chen, Mouxiang Chen, Rongyao Fang, Siyuan Zhang, Xu-Wu Wang, Yu-Heng Jing, et al. The verification horizon: No silver bullet for coding agent rewards, 2026. URL https://arxiv.org/abs/2606.26300.

Jason Wei. Asymmetry of verification and verifier's rule, 2025. URL https://www.jasonwei.net/blog/asymmetry-of-verification-and-verifiers-law. Accessed: 2026-08-19.

Theo Wolf, Alexander David Goldie, Jarek Luca Liesen, Uljad Berdica, Mattie Fellows, and Jakob Nicolaus Foerster. Model-based meta-learning for algorithm discovery. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*, pp. 8–12. IEEE, 2026. doi: 10.1109/icdke.2011.6053918.

Michael J. Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge engineering review (Print)*, 10(2):115–152, 1995. doi: 10.1017/S0269888900008122.

Wanqi Xue, Bo An, Shuicheng Yan, and Zhongwen Xu. Deep reinforcement learning from human preferences. In *Proceedings of the Thirty-ThirdInternational Joint Conference on Artificial Intelligence*, pp. 5298–5306. International Joint Conferences on Artificial Intelligence Organization, 2024. doi: 10.24963/ijcai.2024/586.

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, 2025. URL https://arxiv.org/abs/2504.08066.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37*, pp. 50528–50652. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2024. doi: 10.52202/079017-1601. URL https://openreview.net/forum?id=mXpq6ut8J3.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2022.

Yunxiang Zhang, Kang Zhou, Zhichao Xu, Kiran Ramnath, Yun Zhou, Sangmin Woo, Haibo Ding, and Lin Lee Cheong. Learning to ideate for machine learning engineering agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, Volume 2: Short Papers*, pp. 436–447. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.eacl-short.32.

Jingsheng Zheng, Jintian Zhang, Yujie Luo, Yuren Mao, Yunjun Gao, Lun Du, Huajun Chen, and Ningyu Zhang. Can we predict before executing machine learning agents? In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3941–3974. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.182.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Da-Cheng Li, Eric P. Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Neural Information Processing Systems*, 36:46595–46623, 2023. doi: 10.52202/075280-2020.

Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17744–17761. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.895.

Qihuang Zhong, Kang Wang, Ziyang Xu, Juhua Liu, Liang Ding, Bo Du, and D. Tao. Achieving> 97% on gsm8k: Deeply understanding the problems makes llms better solvers for math word problems. *Frontiers of Computer Science*, 20(1):1–3, 2024. doi: 10.1007/s11704-025-41102-z.

A INFERENCE-ONLY RPM

A.1 PROMPT OPTIMIZATION

We optimize the Inference-only RPM's prompt using Automatic Prompt Optimization (APO) with no underlying weight updates.

A.1.1 OPTIMIZATION CONFIGURATION

We base our prompt optimization on MIPROv2 from the DSPy framework (Opsahl-Ong et al., 2024). Starting from the handwritten baseline template presented in Fig. 6, the meta-proposer generates 10 candidate prompt variations. The search space is explored via Thompson sampling over per-candidate Beta$(1, 1)$ accuracy posteriors across 40 Bayesian minibatch trials. Underperforming templates are progressively pruned, while top-performing survivors undergo deeper validation runs to mitigate optimization-to-holdout shrinkage. We use GPT-5 as both the meta-proposer and the Inference-only RPM. We evaluate against an offline dataset of 990 samples, drawn similarly to the dataset presented in Section 4.2 but without discarding near-ties.

A.1.2 RESULTS

The optimization routine converges on a *Principal Investigator* persona structured around a five-criterion evaluation rubric. This optimized prompt introduces three main shifts from the initial prompt:

- **Structural Evaluation:** Moves from a flat list of decision rules to a structured walkthrough forcing the model to sequentially score problem-model fit and extensibility.
- **Shifting Bug Tolerance:** Replaces strict penalization of code bugs with an evaluation of core ideas, accepting code bugs if the candidate provides a more promising performance ceiling.
- **Strategic Context Utility:** Sharpens context node usage into an explicit mandate to discover open search gaps and actively penalize redundant directions.

The optimized prompt improves selection accuracy from 57.7% to 59.0%.

A.1.3 PROMPT TEMPLATES

We present the complete prompt templates for the Inference-only RPM before and after the automated prompt optimization procedure in Figures 6 and 7 respectively.

```
You are a strict judge selecting between TWO candidate solutions to
    the SAME machine learning task.

Your goal is to choose the candidate whose direction of exploration
    is more likely to eventually lead to a better long-term best test
    score, even if further refinements are needed. Focus on long-term
    potential rather than immediate performance.

Task description:
```markdown
{task_desc}

Context from various solutions to the same machine learning task.
    These are NOT the candidates you are judging.
{context_text}

Candidate A -- Plan:
{plan_A}
Candidate A -- Code:
{code_A}
```
```

```
Candidate B -- Plan:
{plan_B}
Candidate B -- Code:
{code_B}

Decision rules:

Think about which candidate opens up a more promising search
    direction for future iterations.

Prefer the candidate that lays better groundwork for eventually
    achieving the best possible long-term best test score, not just
    the one that looks better right now.

Consider whether the approach is extensible, modular, and amenable to
    iterative improvement.

Prefer correctness and robustness -- a buggy candidate has no
    long-term potential.

Use context nodes as evidence of what has already been tried and what
    directions have shown promise.

Do not assume the context nodes are optimal; the new candidates may
    open better paths.

Output format (STRICT):

Think step by step and provide your reasoning before giving a final
    answer.

Give a final answer of A for Candidate A and B for Candidate B.

Provide your answer inside a \boxed{{}}, ie \boxed{{A}} or
    \boxed{{B}}.
```

Figure 6: **Un-optimized Inference-only RPM prompt prior to the APO procedure.** `task_desc`, `context_text`, `plan_i`, `code_i` are placeholders.

```
You are a principal investigator allocating compute budget to one of
    two branches. Decide which branch is more likely to yield the
    best eventual test score after several iterations. Emphasize
    extensibility, fixability, and promise relative to what has
    already been tried.

Task description:
{task_desc}

Context from other solutions and their scores (not the candidates).
    Use this to identify promising gaps and avoid redundant
    directions:
{context_text}

Candidate A -- Plan:
{plan_A}
Candidate A -- Code:
{code_A}
```

```
Candidate B -- Plan:
{plan_B}
Candidate B -- Code:
{code_B}

Step-by-step evaluation:

Problem-model fit:

Does each candidate's formulation and objective align with the task?
    Any risks of leakage or misalignment? Note which issues are
    trivially fixable vs. fundamental.

Extensibility and upgrade path:

How modular is the code? How straightforward is it to add stronger
    models, features, or training strategies in 1 to 3 iterations?

Learning curve projection:

Based on current choices, estimate how performance might improve over
    the next few iterations. Identify low-hanging fruit (data
    cleaning, features, hyperparameters, regularization, architecture
    changes).

Context-informed novelty:

Relative to {context_text}, does the candidate explore a fresh,
    promising region or iterate intelligently on a proven one? Avoid
    branches that mirror underperforming context without new leverage.

Risk-adjusted potential:

Balance upside (ceiling) against effort/risk to realize it. Bugs are
    acceptable if the approach is sound and fixes are clear; penalize
    only for hard-to-remedy conceptual flaws.

Decision policy:

Choose the candidate whose direction offers higher expected long-term
    best test score and a credible path to get there.

Output format (STRICT):

Provide reasoning following the steps above.

End with a single final answer: A for Candidate A or B for Candidate
    B.

Provide your answer inside a \boxed{A} or \boxed{B}.
```

Figure 7: **Optimized Inference-only RPM prompt after the APO procedure.** `task_desc`, `context_text`, `plan_i`, `code_i` are placeholders.

A.2 Model Ensembling

We evaluate three frontier LLMs, GPT-5 (Singh et al., 2025), Claude Opus 4.8 (Anthropic, 2026) and Gemini 3.1 Pro (Google DeepMind, 2026), individually and ensembled, as baselines on the offline ranking evaluation detailed in Section 4.2.

A.2.1 SINGLE-MODEL CONFIGURATION

We leverage the same prompt produced by our prompt optimization method, and detailed in Appendix A.1. Following the insights from our offline evaluations, detailed in section 5.2.1, we use the maximum reasoning effort for each model and provide the maximum number of context nodes that fit within each model's context window.

A.2.2 ENSEMBLING CONFIGURATION

For both ensembling configurations, we perform 3 independent rollouts for each baseline model prior to aggregating their predictions.

Majority Vote This strategy employs a mechanical aggregation over the baseline models. For each rollout, the final vote of each model's response is extracted. A simple majority vote determines the final ensemble prediction, with ties broken uniformly at random.

LLM Arbiter The arbiter ensemble replaces mechanical vote counting with a high-level consensus call, using Claude Opus 4.8 as the final arbiter. The arbiter receives the original ranking payload (the task description, historical context, and the candidate pair) alongside the anonymized reasoning traces from the three base models. It is instructed to critically evaluate the logical quality of each expert's argument rather than blindly deferring to the majority choice. The exact template is presented in Fig. 8.

```
{individual_ranking_prompt}

============================
You are the FINAL ARBITER.

Above is the exact task that was given to multiple independent expert
judges: a machine-learning task description, context from prior
solutions, and the candidate solutions.

Each expert independently picked the candidate they believe has the
higher long-term potential. Their votes and full reasoning are given
below. The experts are anonymized and may disagree with one another;
any individual expert may be wrong. Do NOT defer to a majority and
weigh the quality of the arguments against the task and the
candidates yourself, then commit to the single best candidate.

------------------------------------------------------------
### Expert 1
Vote: {vote_1}
Reasoning:
{reasoning_trace_1}

------------------------------------------------------------
...
------------------------------------------------------------
### Expert N
Vote: {vote_N}
Reasoning:
{reasoning_trace_N}
============================
Now make the final decision among Candidate A or Candidate B.

Output format (STRICT):
- Briefly explain which expert arguments you found decisive and why.
- Give a single final answer: one letter either A or B.
- Provide your answer inside a \boxed{}, e.g. \boxed{A}.
```

Figure 8: **LLM ensembling arbiter prompt.** `individual_ranking_prompt`, `vote_i` and `reasoning_trace_i` are placeholders.

Model	Accuracy (%)
GPT-5	64.66
Claude Opus 4.8	67.44
Gemini 3.1 Pro	67.40
Ensembling (majority vote)	68.04
Ensembling (arbiter, Claude 4.8)	**69.35**

Table 1: **Selection accuracy of the Inference-only RPM across frontier LLM backbones and ensembling configurations.**

A.2.3 Results

Table 1 summarizes the offline ranking results. Individually, Claude Opus 4.8 and Gemini 3.1 Pro perform nearly identically (around 67.4%), while GPT-5 trails slightly at 64.66%. Ensembling helps smooth out unique model errors. A standard majority vote lifts accuracy to 68.04%, but the LLM-Arbiter achieves the top score of 69.35%, showing that weighing the quality of arguments outperforms a simple vote count.

A.3 Reasoning Analysis

To better understand the decision-making process of inference-only RPMs, we analyze the generated reasoning traces across pairwise candidate rollouts. Specifically, we categorize the justifications used in the model's rationale and evaluate how referencing empirical context impacts selection accuracy.

Evaluation Setup We evaluate reasoning traces generated from 8,715 pairwise rollouts across three LLM backbones: GPT-5, Claude Opus 4.8 and Gemini 3.1 Pro, all produced as part of the evaluations in Section A.2.1.

Justification Categorization Using automated keyword matching, we categorize each reasoning trace into multi-label justification types based on five core categories:

- **Correctness:** Identifying code bugs, syntax errors, or execution flaws.
- **Pretrained Backbone:** Evaluating choices of pretrained model backbones.
- **Prior-Evidence:** Citing measured validation/test metrics from previously explored context nodes.
- **Hyperparameters:** Analyzing learning rates, epoch counts, or optimizer settings.
- **Architecture:** Evaluating structural model modifications (e.g., fusion layers, attention blocks).

As shown in Fig. 9, grounding selections in *Correctness* (66.9% vs. 63.9%), *Pretrained Backbone* (67.0% vs. 63.9%), or *Prior-Evidence* (68.3% vs. 62.4%) yields higher selection accuracy compared

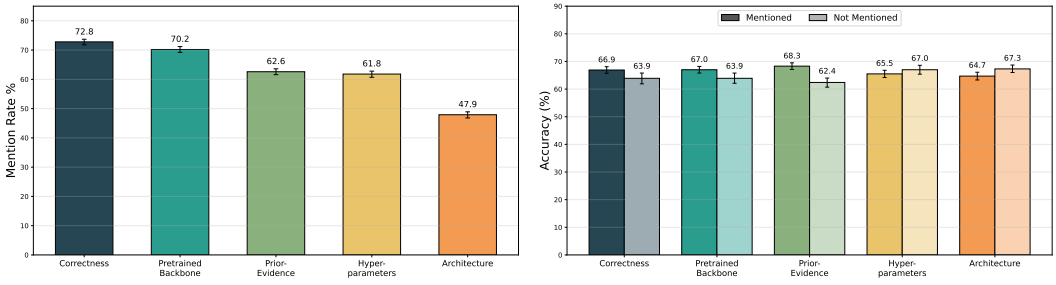

Figure 9: **Reasoning category frequency and selection accuracy impact.** Mention frequency of reasoning categories across reasoning traces (left) and resulting selection accuracy when mentioned versus omitted (right). Citing correctness, pretrained backbones, or prior evidence improves accuracy, whereas citing architecture alone degrades performance. Error bars represent 95% confidence intervals.

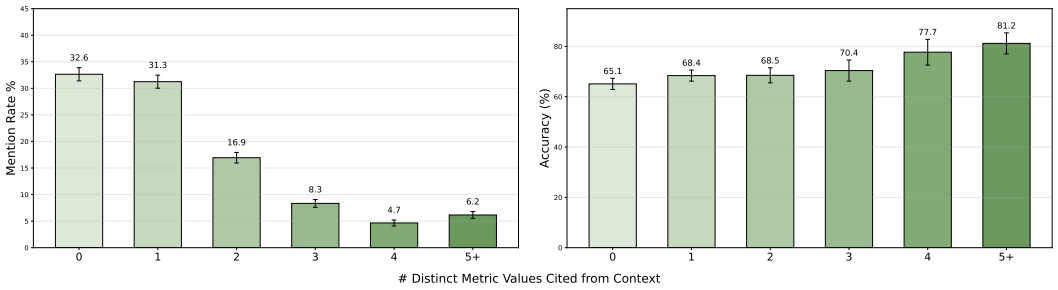

Figure 10: **Context citation frequency and selection accuracy impact.** Mention frequency of distinct context metrics across reasoning traces (left) and resulting selection accuracy across citation counts (right). Selection accuracy increases monotonically as more context metrics are cited. Error bars represent 95% confidence intervals.

to when these justifications are omitted. Conversely, relying on abstract *Architecture* justifications decreases selection accuracy from 67.3% to 64.7%.

Context Citation Analysis We measure how actively the ranker utilizes historical search trajectories by counting the number of distinct context-node metric values cited in each reasoning trace. Specifically, we parse rollouts using regular expressions to extract, deduplicate, and ground explicit node identifiers and validation metrics cited from the prompt history. As shown in Fig. 10, selection accuracy scales monotonically with the volume of cited context metrics: rollouts citing zero metric values achieve 65.1% accuracy, whereas rollouts citing five or more distinct metric values reach 81.2%.

B AGENTIC RPM

B.1 ALGORITHM DESIGN AND HYPERPARAMETERS

In this section we provide more implementation details for agentic RPM. As introduced in Section 3.2, the core of agentic RPM is the pilot experiments. During our study, we find that the agent can be too conservative in scheduling the time budget, leaving a large portion of the time budget unused at the first call of `submit_solution`. Even though we could prompt the agent to run more pilot experiments after the initial submission, the follow-up experiments are often limited to hyperparameter tuning of previous ones. Over conservative time budget scheduling and repetitive follow-up experiments jointly lead to less informative pilot experiments.

To deal with this problem, we use two mechanisms to elicit more informative follow-up experiments. First, we *overstate* the remaining time budget in the prompt (reporting it as several times larger than it truly is), discouraging the agent from stopping prematurely. Second, after each `submit_solution` call, a separate model reviews the findings so far and gives a *feedback* that either proposes the single most informative next experiment or signals that the evidence is already sufficient, in which case we end the loop; its proposal and the remaining budget are then returned to the agent to guide the follow-up experiment. The overall workflow of agentic RPM with the overstating and feedback mechanism is presented in Algorithm 1.

Regarding the hyperparameters, we use the same backbone for the pilot experiment model π_p and the feedback and final prediction model π_f without fine-tuning. The real time budget is $B = 300s$ even though we overstate that the time budget is $B' = 2700s$, which is 9 times larger than the actual value. Every time a pilot experiment is finished and its empirical finding is summarized, we will terminate the loop if 1) the feedback and prediction model π_f does not suggest the next pilot experiment; or 2) the real remaining time budget is less than a threshold $\theta = 60s$; or 3) The maximum number of pilot experiments $N = 30$ is reached. Notably, the time budget only take effect after all the tool calls in the current turn is executed. Therefore, in practice it is possible that the agentic RPM uses more time than the budget B if the a tool call is extremely time-consuming.

Algorithm 1 The workflow of the proposed agentic RPM.

Require: Input query q containing multiple candidate solutions to rank, pilot-experiment model π_p, feedback and final prediction model π_f, maximum number of pilot experiments N, time budget B, stop threshold θ.

Ensure: Final decision on which candidate to choose y_{out}.

Initialize used time $b = 0$; pilot experiment count $n = 0$; pilot experiment context $x = [q]$; pilot experiment findings s = []; last submission flag $F_l = \texttt{False}$; pending experiment flag $F_p = \texttt{False}$
while True **do**
  Generate pilot-experiment model response $y_{\texttt{asst}} \sim \pi_p(\cdot \mid x)$;
  Parse possible tool calling $[(\texttt{tool}_i, \texttt{para}_i)]_{i=1}^{T}$ from response y, where T is the number of function call
  if Find any $\texttt{tool}_i = \texttt{python}, i = 1, 2, \ldots, T$ **then**
    Set pending experiment flag $F_p = \texttt{False}$
  if No tools are called and $T = 0$ **then**
    Update context $x = x+$ "Please proceed to the next step using your best judgment..... "
    continue
  Execute the function $y^i_{\texttt{tool}} = \texttt{tool}_i(\texttt{para}_i),\ i = 1, 2, 3, \ldots, T$ and update used time b.
  Update context $x = x + y_{\texttt{asst}} + [y^i_{\texttt{tool}}]_{i=1}^{T}$
  if Used time surpasses time budget $b > B$ **then**
    Find $i = \text{argmax}_i(\texttt{tool}_i = \texttt{python})$ and $j = \text{argmax}_j(\texttt{tool}_j = \texttt{submit_solution})$
    if $i > j$ **then**
      Set last submission flag $F_l = \texttt{True}$
      Update context $x = x+$ "TIMEOUT. You have run out of time. Please immediately submit ..."
      continue
    break
  if $\texttt{tool}_T = \texttt{submit_solution}$ **then**
    if Pending experiment flag F_p is True **then**
      Update x=x+ "INVALID submit_solution call...You MUST use python tool to run experiment"
      continue
    $s = s + \texttt{para}_T$
    Obtain feedback $f \sim \pi_f(\cdot \mid s)$
    if The feedback f does not contain next experiment **then**
      break
    if The remaining time budget is less than the stop threshold θ **then**
      break
    Set the pending experiment flag F_p to be True.
    Update context $x = x + f$
    if F_l is True **then**
      break
    if The maximum number of pilot experiments is reached $\texttt{len}(s) \geq N$ **then**
      break
  if The last submission flag F_l is True **then**
    break
Make the final decision $y_{\text{out}} \sim \pi_f(\cdot \mid s)$
Return: Final decision y_{out}

B.2 Design Ablations

Both end-to-end (Section 5.1) and offline evaluations (Section 5.2.2) demonstrate the efficacy of the Agentic RPM to accelerate and enhance AIRA. To have a better understanding of how different components contribute, in this section we conduct an ablation study on the proposed Agentic RPM. Specifically, we experiment with the following variants:

- **w/o overstating**: The agent is informed with the real remaining time budget without any exaggeration;
- **w/o feedback**: The feedback mechanism is removed and the it receives no planning about the next valuable experiment to run.
- **w/o GPU access**: The agent no longer has access to the H200 GPU and thus can only run pilot experiments with a CPU.

We follow the same offline evaluation setup presented in Section 4.2, and provide the Agentic RPM a 5-minute compute time budget. As presented in Fig. 11, the inclusion of the feedback mechanims and overstating the provides time budget provides an improvement in selection accuracy. We also observe a significant drop in performance when removing GPU access for the pilot experiments. To further understand the behavior difference between the original agent and its variants, we analyze the average tool call count and the distribution of the pilot-experiments' running time. The results are presented in Fig. 11 and Fig. 12 respectively. From the figures we can observe that: (1) removing time budget overstating leads to less `python` call and less time-consuming experiments; (2) removing the GPU access increases the time cost of each experiment since the pilot experiments can only be run on CPU in this case; (3) removing feedback mechanism leads to more and briefer experiment, suggesting that the pilot experiments it runs can be trivial and less informative.

B.3 Prompt Templates

In this section, we detail the complete set of prompts guiding the agentic RPM workflow. Specifically, we present the prompt templates to initialize the workflow (Fig. 13), to provide feedback (Fig. 14), and to give a final selection (Fig. 15).

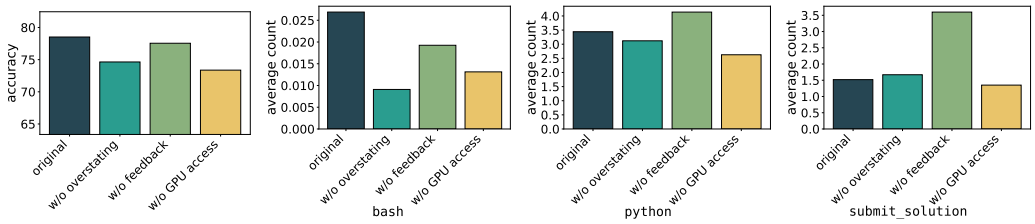

Figure 11: **Accuracy and tool call count across Agentic RPM variants.** Offline selection accuracy using GPT-5 (leftmost) and average tool call counts for `bash` (middle left), `python` (middle right), and `submit_solution` (rightmost). `bash` is seldom used by any variant. The *w/o feedback* variant calls `python` and `submit_solution` more frequently than other variants.

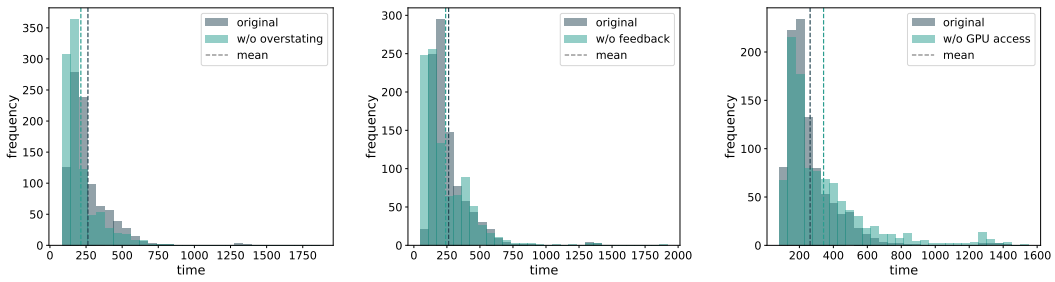

Figure 12: **Pilot experiments' execution time distributions across Agentic RPM variants.** Compared with *w/o overstating* variant (left) and *w/o feedback* variant (middle), the original Agentic RPM produces more time-consuming pilot experiments. Without GPU access (right), the average execution time is further increased.

INTRODUCTION:
You are a Kaggle Grandmaster and Lead Data Scientist acting as a **Strategic Evaluator**. Your task is to analyze two candidate machine learning solutions and **predict which one will achieve a superior test score without actually evaluating on the test set**.
You are given two candidate solutions for a machine learning competition on Kaggle. We do not have access to the test set. Instead, we estimate test performance by running 5-fold cross-validation (CV) on the public training set. However, running the full 5-fold CV on the entire training set is prohibitively time-consuming and beyond our computational budget. Therefore, you must predict which solution would achieve a better test score if it were fully executed, using fast, high-leverage experiments (Pilot Experiments) on the public training set.
Remember: your goal is NOT to build the final solution, but to generate code that helps **predict the relative ranking of the two solutions**.

TASK DESCRIPTION

```
{task_desc}
```

You can access the data using standard Python libraries (pandas, datasets, etc.).
The data is already mounted and ready to use — no need to download or prepare it.
Important: Pay attention to whether higher or lower values indicate better performance (e.g., accuracy is higher-is-better, and RMSE is lower-is-better).
Common Pitfalls: Do NOT use `from transformers import AdamW` (it has been removed). Use `from torch.optim import AdamW` instead.
Your Resources (as Strategy Evaluator): {device}
Time Limit: {time_limit}

CANDIDATE SOLUTIONS:
Option A:

```python
{candidate_a}
```

Option B:

```python
{candidate_b}
```

.....

INSTRUCTIONS FOR PILOT EXPERIMENTS:
You must write a **fast, efficient Python script** (with a target runtime under {time_limit}). When executed, this script should produce evidence that helps determine which candidate solution would achieve a better test score.

You have full freedom in choosing your analysis strategy. For example, you may:
- Run a simplified or partial cross-validation (e.g., fewer folds, subsampled data).
- Compare key architectural or hyperparameter differences between the two solutions and run targeted ablation experiments.
- Evaluate both solutions on a single train/validation split.
- Perform any other analysis you believe is informative.

REQUIREMENT:
Step 1: Use the above information and the provided tools to first understand the task and the candidate solutions, and then write a Python script to generate evidence.
Step 2: Use the provided tools to fix any bugs in your script and run it, adding any additional experiments needed to strengthen the evidence.
Step 3: You **MUST** summarize your experiment design rationale, experimental results and findings in plain text and submit them with the tool `submit_solution()`.

Figure 13: **Agentic RPM's Initialization Prompt.** The `task_desc`, `device`, `time_limit`, `candidate_a` and `candidate_b` are placeholders.

INTRODUCTION:
You are a Kaggle Grandmaster and Lead Data Scientist acting as an **Experiment Planner**. Your task is to decide whether more pilot experimentation is needed, and if so, propose the **single most valuable next experiment** for distinguishing which of the candidate machine learning solutions is most likely to achieve the best test score.
You are given a set of candidate solutions for a machine learning competition on Kaggle. You only have access to the public training set but do not have access to the test set. A natural proxy for test performance would be 5-fold cross-validation (CV) score on the public training set. However, running the full 5-fold CV on the entire training set is time-consuming. Therefore, your goal is to design diverse pilot experiments that can approximate relative test performance **with significantly less computation** than standard 5-fold CV.

Important: Your job is **only** to decide whether to stop or to design the **next best experiment**. A separate coding agent will execute the experiment and collect results, and a separate judging agent will make the final prediction based on the experimental evidence. You should NOT write any code yourself, and you should NOT make the final prediction yourself.

TASK DESCRIPTION
```
{task_desc}
```
ENVIRONMENT & CONSTRAINTS (for the coding agent to follow):
Data Access: You can access the data using standard Python libraries (pandas, datasets, etc.). The data is already mounted and ready to use — no need to download or prepare it.
Metric: Pay attention to whether higher or lower values indicate better performance (e.g., accuracy is higher-is-better, RMSE is lower-is-better).
Common Pitfalls: Do NOT use `from transformers import AdamW` (it has been removed). Use `from torch.optim import AdamW` instead.
Computation Resources: {device}
Time Limit: {time_limit}. If you propose a next experiment, it must fit within the remaining time budget with room left for execution reporting and a possible final decision.
CANDIDATE SOLUTIONS:
Option A:
```python
{candidate_a}
```
Option B:
```python
{candidate_b}
```
.....

{prev_findings}

INSTRUCTIONS:
You must first decide whether the existing evidence is already sufficient. If yes, recommend stopping. If not, propose exactly **one** next experiment that is expected to provide the highest decision value.
The next experiment should be detailed and specific enough that a coding agent can implement and execute it by strictly following your instructions.
Prefer experiments that reduce the most important unresolved uncertainty about the relative ranking of the candidate options (Option A, Option B, Option C, ...).
Do NOT propose a trivial repetition of an earlier experiment unless you clearly justify why a stability check is necessary.
Do NOT propose cosmetic variations of previous experiments (for example, the same proxy, same split logic, and same reasoning with only a tiny tweak) unless that variation is specifically needed to resolve an important uncertainty.
Prefer experiments whose outcome could realistically change the current provisional ranking.
If the existing evidence is already strong enough and another experiment is unlikely to add meaningful value, set ‘stop: true‘.

You have full freedom in designing your strategy. For example, you may:
- Run a simplified or partial cross-validation (e.g., fewer folds, subsampled data).
- Compare key architectural or hyperparameter differences among the candidate solutions and run targeted ablation experiments.
- Evaluate the candidate solutions on a single train/validation split.
- Perform any other analysis you believe is informative.

If you propose a next experiment, it must include:

- **title** (str): A short descriptive name for the experiment.
- **goal** (str): What this experiment is trying to find out.
- **steps** (str): A detailed, step-by-step description of what the coding agent should implement and run.
- **expected_runtime** (float): A rough estimate of how long this experiment will take in minutes given the computation resources.

OUTPUT REQUIREMENT: Output a JSON object with the following schema:

```json
{{
  "stop": <true_or_false>,
  "reason": "<short explanation>",
  "next_experiment": {{
    "title": "<string>",
    "goal": "<string>",
    "steps": "<string>",
    "expected_runtime": <float>
  }}
}}
```

If "stop": true, then set:

```json
"next_experiment": null
```

Important:
- Output strictly valid JSON only.
- Do not wrap the JSON in Markdown fences.
- All string values must be valid JSON strings with double quotes.
- "steps" should be a single string. You may use ‘
n‘ inside the string for line breaks.

Example Output:

```
{{
  "stop": false,
  "reason": "Existing micro-split results are mixed and high-
  variance. A decisive equal-compute comparison
   is still needed.",
  "next_experiment": {{
    "title": "Matched-step class-balanced holdout",
    "goal": "Run a matched-step comparison between the candidate
    options under the same compute budget.",
    "steps": "1. Construct a class-balanced holdout split.
            2. Match training steps and optimizer settings.
            3. Run each candidate option under identical budget.
            4. Compare validation accuracy.",
    "expected_runtime": 6.0
  }}
}}
"""
```

Figure 14: **Agentic RPM's feedback prompt.** The `task_desc`, `device`, `time_limit`, `candidate_a`, `candidate_b` and `prev_findings` are placeholders.

INTRODUCTION:
You are a Kaggle Grandmaster and Lead Data Scientist acting as a **Final Prediction Analyst**. Your task is to analyze multiple machine learning solutions and their pilot experiment results to predict which solution will have superior test cross-validation performance.
You have already conducted fast pilot experiments to probe the potential of these multiple solutions. Now you must analyze the empirical evidence from these experiments and make a final prediction.
TASK DESCRIPTION

```
{task_desc}
```

CANDIDATE SOLUTIONS:
Option A:

```python
{candidate_a}
```

Option B:

```python
{candidate_b}
```

```
{execution_output}
```

INSTRUCTIONS:
Based on the pilot experiment results above, you must predict which candidate solution will achieve the best test cross-validation score when fully implemented.

Consider the following in your analysis:
- **Empirical Evidence:** What do the execution outputs tell you about the performance of each solution?
- **Reliability:** Are the results consistent and reliable, or are there signs of instability?
- **Potential:** Which solution shows more promise for achieving higher validation metrics?
- **Implementation Quality:** Does the output suggest successful execution or potential issues?

OUTPUT REQUIREMENT:
Think step by step and provide your reasoning before giving a final answer. Each candidate solution above is labeled with a letter (Option A, Option B, Option C, ...). Your final answer must be the single letter of the best option, enclosed in `\\boxed{{}}` — for example `\\boxed{{A}}`, `\\boxed{{B}}`, or `\\boxed{{C}}`. Output exactly one letter, and only choose from the options listed above.

Example response format:
Analysis: [Your detailed reasoning here]

Final Prediction: `\\boxed{{A}}`

Figure 15: **Agentic RPM's Final Prediction Prompt.** The `task_desc`, `candidate_a`, `candidate_b` and `execution_output` are placeholders.

C End-to-End Evaluations

C.1 Oracle Selection

To establish empirical performance upper bounds for greedy child candidate selection in E2E evaluations, we compare against two evaluation settings: a **Validation Oracle** and a **Test Oracle**. For both oracles, all 15 generated candidate solutions are fully executed at each step, and the candidate with the highest validation or test score respectively is selected and added back to the tree. To isolate the impact of oracle selection while maintaining comparability with standard agent runs, we ignore the overhead of executing all 15 candidates and count only the selected candidate's runtime toward the 24-hour budget. We conduct the oracle-selection evaluations with Qwen3.6-27B as the LLM backbone.

Neither oracle is viable for online deployment, where the Test Oracle relies on privileged test-set metrics, and the Validation Oracle incurs prohibitive compute overhead by executing all 15 candidates. Nevertheless, they define clear performance ceilings for greedy child candidate selection.

As shown in Fig. 16, both oracles provide immediate and steady gains over all other methods and baselines. The test oracle outperforms than the validation oracle with a final average normalized score of 0.759 and 0.748 respectively. The Agentic RPM most closely approaches the Validation Oracle ceiling, however we suspect that other RPMs could further close this gap and even outperform the immediate selection baselines.

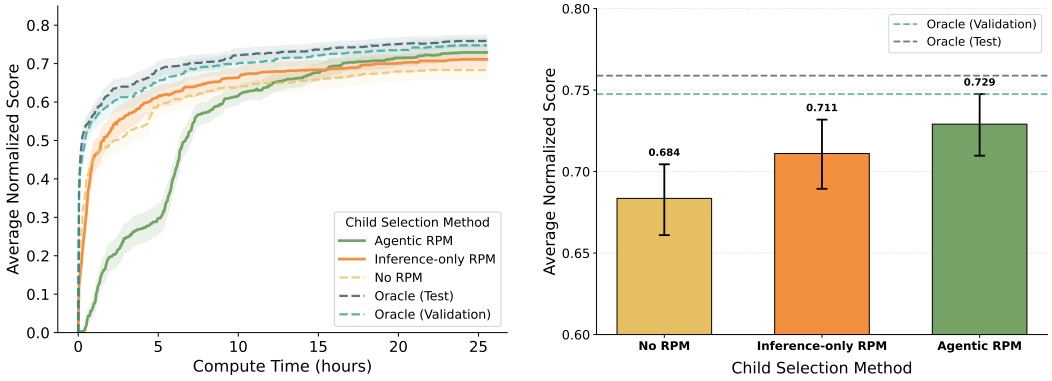

Figure 16: **Comparison between RPM-based and oracle selection.** Average normalized scores over time (left) and final performance (right) for AIRA-dojo with RPM-augmented child selection and Qwen3.6-27B LLM backbone on AIRS-Bench, with error bars indicating the 95% confidence intervals. Dashed lines indicate the No-RPM baseline and immediate validation or test oracles.

C.2 Impact of Selection Quality

To determine whether local decision quality directly drives downstream agent performance, we retrospectively analyze the relationship between step-wise candidate selection quality and final E2E search scores.

Evaluation Setup Each data point in Fig. 17 represents an individual seed run, where the final score is the normalized metric averaged across all AIRS-Bench tasks. To quantify decision quality, we retrospectively evaluate all N candidate solutions generated at each step against ground-truth benchmarks. We define *Selection Advantage* as $S_{\text{chosen}} - \frac{1}{N}\sum_{i=1}^{N} S_i$, measuring the difference between the chosen candidate's ground-truth score S_{chosen} and the mean score of all N candidates in the candidate pool. For each seed run, we report the selection advantage averaged across all selection steps and across all tasks. Analysis was conducted on the E2E evaluations of Qwen3.6-27B.

Findings We find a statistically significant positive correlation between the selection advantage and the final end-to-end performance (Pearson $r = 0.55, p = 0.0007$; Spearman $\rho = 0.56, p = 0.0004$). As expected, random selection yields an average selection advantage near zero. Both preference models consistently improve selection quality over random selection, with the Agentic RPM achieving the highest overall selection advantage and final score. These results confirm that higher per-step candidate selection quality translates directly to better end-to-end AIRA search performance.

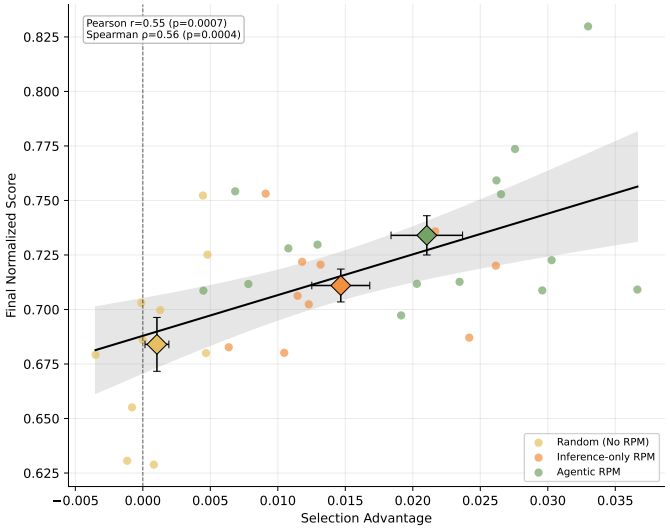

Figure 17: **Selection Advantage vs. Final Normalized Score.** Final normalized score plotted against step-wise selection advantage across individual seed runs. Evaluations use AIRA-dojo with a Qwen3.6-27B backbone on AIRS-Bench tasks. Large diamonds denote mean values ($\pm 95\%$ CI). Selection advantage strongly correlates with end-to-end performance (Pearson $r = 0.55, p = 0.0007$; Spearman $\rho = 0.56, p = 0.0004$), with Agentic RPM achieving both higher selection advantage and final score than Inference-only RPM and Random selection.

D RPM-AUGMENTED FINAL SOLUTION SELECTION

As presented in Sections 2.2 and 3.3, an RPM could intervene at three points within an evolutionary search based AIRA's search: parent selection (PS), child creation (CC) and final-node selection (FNS).

A key distinction separates CC from PS and FNS: CC operates on unexecuted candidates at selection time, while PS and FNS candidates have already been executed and carry validation scores. Thus, a strong selection heuristic (such as greedy selection) can be trivially constructed for PS and FNS. As such, we focus in this manuscript on augmenting CC within an AIRA, but present exploratory initial results on augmenting FNS.

D.1 EVALUATION SETUP

To evaluate the capability of RPMs to perform FNS, we utilize the search trees produced by the Inference-only RPM End-to-End evaluations across the public text-and-tabular AIRS-BENCH tasks (Sections 4.1 and 5.1). Each datapoint in the evaluation represents a complete AIRA-dojo search tree on a specific task, and the RPM is tasked with selecting the candidate solution with the highest test score from this tree. We use Qwen3.6-27B (Qwen Team, 2026) as the LLM backbone for FNS to maintain model consistency between the original AIRA-dojo operators and selection. We also evaluate GPT-5 as an alternative backbone to validate how stronger selection quality impacts FNS. To contextualize overall performance, we compare against two global reference baselines: a global *Test Oracle* (the maximum test score in the entire search tree) and a global *Validation Oracle* (the test score of the candidate with the overall highest validation score; the default FNS mechanism of AIRA-dojo). Performance is measured using the average normalized test score of the selected candidate.

D.2 METHOD

Given an AIRA-dojo search tree, we filter out buggy nodes and retain all valid candidate solutions, with validation and test metrics normalized following Section 2.1. Because an AIRA-dojo search tree can contain hundreds of nodes, we restrict the FNS candidate pool to the N nodes with the highest validation scores, sweeping across pool sizes $N \in \{2, 4, 6, 8, 10\}$. To contextualize performance within each candidate pool, we evaluate against two local baselines: a local *Test Oracle (N shown)* representing the best test score available within the N-candidate pool, and a *Random (N shown)* baseline representing the expected score of a uniform random pick from that pool.

Selection from the candidate pool is conducted via a round-robin tournament, where across 30 repeated matches, candidate subsets of $\min(N, 5)$ are presented to the Inference-only RPM to select between. The candidate accumulating the highest point total across all matches is selected as the tournament winner, with ties broken uniformly at random. The complete prompt template used for the Inference-only RPM during tournament matches is detailed in Fig. 18.

```
You are a strict judge selecting the BEST among 5 candidate solutions
    to the SAME machine learning task.

Your goal is to choose the candidate most likely to achieve a better
    test score.

Task description:
```markdown
{{task_description}}
```

Context from various solutions to the same machine learning task.
    These are NOT the candidates you are judging.
{{context_nodes_description}}

Candidate A - normalized_validation_metric: {{candidate_a_score}}
Candidate A - Code:
```python
{{candidate_a_code}}
```
```

```
```

Candidate B - normalized_validation_metric: {{candidate_b_score}}
Candidate B - Code:
```python
{{candidate_b_code}}
```

Candidate C - normalized_validation_metric: {{candidate_c_score}}
Candidate C - Code:
```python
{{candidate_c_code}}
```

Candidate D - normalized_validation_metric: {{candidate_d_score}}
Candidate D - Code:
```python
{{candidate_d_code}}
```

Candidate E - normalized_validation_metric: {{candidate_e_score}}
Candidate E - Code:
```python
{{candidate_e_code}}
```

Decision rules:
- Prefer the candidate most likely to produce a better test score for
    the given task.
- Prefer correctness, robustness, and task-fit over style or
    verbosity.
- Use context nodes only as supporting evidence (e.g., what has
    already been tried, what validation score looked like).
- Do not assume the context nodes are optimal; the new candidates may
    be better.

Output format (STRICT):
- Think step by step and provide your reasoning before giving a final
    answer.
- Give a final answer of A, B, C, D or E.
- Provide your answer inside a \boxed{}, ie \boxed{A}, \boxed{B},
    \boxed{C}, \boxed{D}, \boxed{E}.
```

Figure 18: **Inference-only RPM's prompt for Final Node Selection.** For pool sizes $N$ where $N < 5$, the match size is capped at $N$ and the boxed-letter menu shrinks accordingly. The `task_description`, `context_nodes_description`, `candidate_i_score` and `candidate_i_code` are placeholders.

### D.3 RESULTS

As shown in Fig. 19, both RPM variants consistently outperform the local random baseline across all pool sizes $N$. Leveraging GPT-5 yields consistently higher normalized test scores than Qwen3.6-27B, confirming that stronger base reasoning directly improves selection quality. However, the performance of both RPMs degrades as $N$ increases, sliding further below the global *Validation Oracle* (AIRA-dojo's default selection mechanism) as the judge attempts to separate between an increasingly weaker pool of candidates.

These results illustrate the fundamental challenge of augmenting FNS: once candidates are executed, validation-based selection enables strong heuristics that are difficult to improve upon purely through code inspection. This is further limited given the HCE protocol's strong test-validation generalization, the gap between the *Validation Oracle* and the ultimate *Test Oracle* is extremely narrow. While static inference-only judgment struggles to outperform this greedy baseline on executed nodes, future work
```

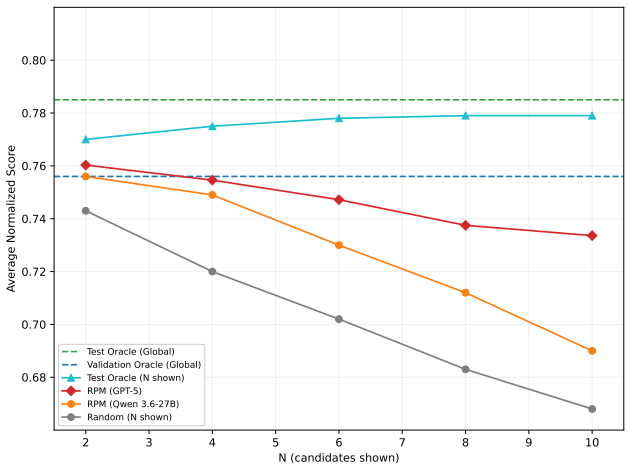

Figure 19: **Final Node Selection performance across candidate pool sizes.** Average normalized test score as a function of the top-N validation candidate pool size. Inference-only RPM judges (GPT-5 and Qwen3.6-27B) beat random selection but degrade as N increases, becoming progressively worse than the Validation Oracle (AIRA-dojo's default selection strategy). The Inference-only RPM with the stronger GPT-5 LLM backbone outperforms that with the Qwen backbone.

could deploy more active selection paradigms, such as the Agentic RPM, to run targeted experiments and bridge the remaining gap to the Test Oracle.