🎯 AIM: AGENTIC IDEA MANAGEMENT FOR AUTOMATED RESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier LLMs are increasingly used to automate scientific research through iterative search. We distinguish idea-driven search from solution-driven search and identify three core challenges: organizing evolving research ideas, selecting promising directions, and maintaining alignment between ideas and their implementations. To address these challenges, we introduce the *Agentic Idea Manager* (AIM), a fully autonomous framework for managing and exploring research directions in idea-driven automated research. Inspired by Bayesian optimization, AIM uses an Agentic Surrogate and an Agentic Acquisition mechanism to organize discovered ideas and guide their selection. A Solution Auditor maintains idea–solution integrity, while a Resource Planner adaptively allocates the remaining experimental budget across parallel search branches. Experiments on 10 AutoLab benchmark tasks show that AIM surpasses the strongest baseline by 1.6 percentage points on System Optimization tasks and 4.9 percentage points on long-horizon Model Development & CUDA tasks. Notably, AIM reaches the best baseline performance up to 3.1× faster in wall-clock time. We further provide a theoretical analysis of *when searching over ideas becomes beneficial*. Our analysis shows that explicit idea-level allocation makes semantic coverage directly controllable, and that broader coverage becomes increasingly valuable when competitive research directions are sparse among many plausible alternatives.

1 INTRODUCTION

Large language model (LLM) agents are increasingly used to automate scientific research through iterative experimentation. Modern research agents can propose candidate approaches, implement solutions, execute experiments, inspect verifier feedback, and use accumulated evidence to determine what to try next (Lu et al., 2024; Jiang et al., 2025; Toledo et al., 2026; Meng et al., 2026). Because implementation and evaluation are expensive, effective automated research requires allocating a limited experimental budget across possible approaches. The resulting search process must support both the discovery of promising directions and the refinement of their implementations.

In this work, we introduce the first categorization of automated research, according to its primary unit of search: *solution-driven* and *idea-driven*. Solution-driven approaches (Figure 1 (a)) search directly over executable artifacts, using verifier feedback to iteratively modify candidate solutions (Cemri et al., 2026; Liu et al., 2026; Toledo et al., 2026; Jiang et al., 2025). Idea-driven approaches (Figure 1 (b)), on the other hand, maintain research ideas as explicit object of reasoning; they select *which idea to investigate* and delegate *how to implement it* to a solver (Meng et al., 2026; Jin et al., 2026; Yamada et al., 2025; Weng et al., 2026). Both paradigms ultimately evaluate executable solutions, but organize search at different levels of abstraction. Working directly with solutions supports fine-grained code refinement, while idea-driven methods facilitate comparison across approaches, helps the transfer of lessons, and makes research trajectories easier to interpret.

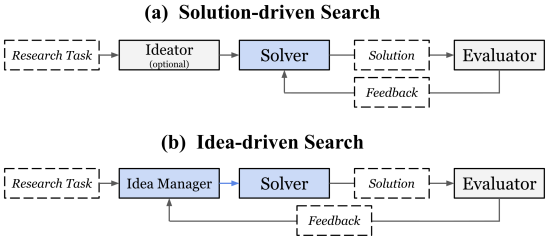


Figure 1: Solution-driven vs Idea-driven Research

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.

This categorization highlights the management of research ideas as a distinct design problem in automated research, with three central challenges in determining the *idea management structure*, *idea selection strategies*, and ensuring *idea–solution integrity*. To address these challenges, we introduce the **Agentic Idea Manager (AIM)**, a fully-autonomous research idea management framework for automated research (Figure 2 (a)). Inspired by the surrogate-acquisition structure of Bayesian optimization, its *Agentic Surrogate* maintains a dynamic structure of idea clusters and ranks their promise using experimental evidence, grouping related proposals across generation lineages. The *Agentic Acquisition* mechanism balances exploration and exploitation at both cluster- and idea-level selection, dispatching chosen candidates to parallel solvers. After implementation and evaluation, the *Solution Auditor* checks task validity and idea–solution integrity, and the audited evidence guides subsequent direction of research. Finally, the *Resource Planner* adjusts parallelism across iterations, balancing concurrent trials with longer exploration under a fixed experimental budget.

We evaluate AIM on ten AutoLab tasks spanning System Optimization and Model Development & CUDA tasks. AIM achieves the highest average scores in both task groups: 67.0% on System Optimization and 55.8% on long-horizon Model Development & CUDA, exceeding the strongest baseline, ScientistOne (Meng et al., 2026), by 1.6 and 4.9 percentage points, respectively. Moreover, AIM reaches ScientistOne's best score up to 3.1× faster, demonstrating its efficiency and supporting the value of agentic idea organization, evidence-guided selection, and audited feedback for automated research. In addition to empirical studies, we further provide theoretical analyses to identify when explicit idea-level search is useful. Our analysis shows that idea-driven search makes such breadth directly controllable through explicit allocation to distinct ideas. More importantly, broader coverage becomes increasingly valuable when a task contains many plausible research directions but only a small fraction are competitive. Our main contributions are:

- We introduce a categorization of automated research methods: *solution-driven* and *idea-driven*. We identify *three central challenges* of idea-driven approaches: idea management, idea selection, and idea–solution integrity.
- We introduce *Agentic Idea Manager* (AIM), a fully-agentic idea-driven framework that effectively manages ideas as semantic clusters, selects ideas based on its explicit estimation of performance, and audits the solutions to achieve a reliable research pipeline.
- We achieve *strong empirical performance* on 10 Autolab tasks, surpassing the strongest baseline by 1.6pp on System Optimization and 4.9pp on long-horizon Model Development & CUDA tasks. We also provide *rigorous theoretical analyses* of when idea-driven search can be useful.

2 Related Works

Solution-driven Search. Solution-driven approaches remain close to the verifier and make the executable implementation the primary unit of search. AIDE (Jiang et al., 2025) and AIRA (Toledo et al., 2026) explore candidate solutions through tree search, while AlphaEvolve (Novikov et al., 2025), AdaEvolve (Cemri et al., 2026), and EvoX (Liu et al., 2026) use evolutionary mechanisms. MLE-Star (Nam et al., 2026), DS-Star (Nam et al., 2025), and RPM (Foster et al., 2026) further develop solution-level refinement and selection. Solution-driven search's tight coupling of the evaluator and solver lets experimental feedback directly guide implementation changes, but can entangle progress on a research direction with engineering decisions about a particular artifact.

Idea-driven Search. Idea-driven approaches make research ideas or hypotheses explicit search objects, then delegate their implementation to a solver. The AI Scientist-v2 (Yamada et al., 2025), MARS (Chen et al., 2026b), and Arbor (Jin et al., 2026) manages a tree of ideas or lessons, and DeepScientist (Weng et al., 2026) maintains the research ideas and hypotheses as a list. A subset of ideas is implemented by a solver module. ScientistOne (Meng et al., 2026), on the other hand, utilizes a beam-search-like algorithm that keeps the best-performing ideas throughout the research iterations. These approaches separates idea search from implementation, enabling deliberate exploration of distinct ideas, but requires deciding which should merit costly experiments and evaluations.

This idea-driven search introduces three central challenges: (1) *Idea management scaffold*: the framework must organize an expanding pool of ideas and accumulated evidence into a persistent, interpretable representation. (2) *Idea selection*: it must select which ideas to implement and how to allocate the remaining budget; existing methods typically rely on fixed rules such as UCB (Weng et al., 2026) or MCTS (Yamada et al., 2025), or naively use LLM judgments without explicit es-

timates (Jin et al., 2026). (3) *Idea–Solution integrity*: the separation between ideation and implementation creates an integrity risk. A solver may produce code that does not faithfully realize the selected idea, causing its score and derived lessons to be misattributed. These challenges motivate jointly managing the idea space, making evidence-grounded search decisions, and verifying idea–solution alignment, which we address with the our proposed method. In this work, we propose an idea-driven research framework that addresses these challenges. A formal discussion on when to search over ideas is provided in Section 6. An extended section for Related Works is in Appendix A.

3 Automated Research: Problem Definition

In automated research, it is generally infeasible to evaluate every plausible idea because implementation and verifier calls are expensive. Effective automated research therefore requires more than sequential idea and code generation. It requires a structured representation of the ideas and solutions discovered so far, and a principled mechanism to navigate the search process under a limited computational budget. Accordingly, we formally define the relevant search space and objectives:

Idea and Solution Spaces. Let $\mathcal{X}$ denote the space of admissible research ideas for task τ, where each $x \in \mathcal{X}$ is a natural-language description of a candidate approach. In this work, an *idea* is defined as, but not limited to, a structured text comprising a short title, brief hypothesis/abstract, and experiment plans (see Appendix E.6 for examples). At timestep t, the agent has access only to a finite pool $\mathcal{P}_t \subset \mathcal{X}$ containing the ideas discovered thus far. Meanwhile, we distinguish the space of executable solutions $\mathcal{Z}$ from the idea space $\mathcal{X}$. Given an idea x, an implementation process $g : \mathcal{X} \to \mathcal{Z}$ produces an executable solution $z = g(x)$, assuming a single deterministic mapping per idea in this work, which is then scored by a fixed verifier $v : \mathcal{Z} \to \mathbb{R}$:

$$z = g(x), \qquad y = v(z) \quad \Longrightarrow \quad f(x) \triangleq v(g(x)). \tag{1}$$

Here, f denotes the expensive research process for implementation and evaluation.

Objective. Evaluating $f(x)$ requires first translating the natural-language idea into an executable implementation and then compiling and running that implementation against a fixed verifier. Consequently, the dominant cost is not in proposing an idea, but obtaining reliable evidence about its quality through implementation and verification, which can itself even be very expensive and noisy. Thus, the core objective is to identify the highest-performing idea within a given computational budget measured in the number of experiment executions and/or runtime hours for search. Let $c(x)$ denote the cost of implementing and verifying idea x, and let N be the total computational budget. For a sequence of evaluated ideas $x_1, \ldots, x_n$, the objective is to maximize the best performance:

$$\max_{x_1,\ldots,x_n \in \mathcal{X}} \ \max_{1 \le i \le n} f(x_i) \qquad \text{s.t.} \qquad \sum_{i=1}^{n} c(x_i) \le N. \tag{2}$$

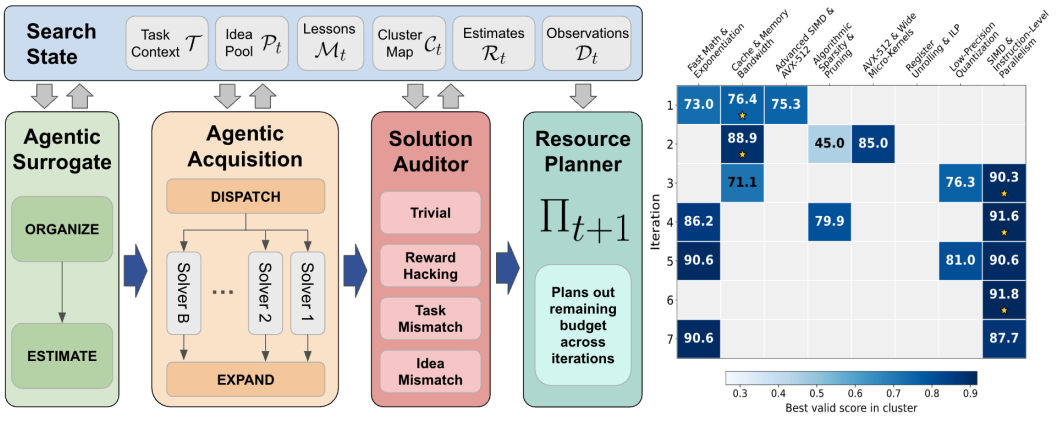
Within this framework, the agent must therefore determine which subset of ideas to explore first.

4 The AIM Framework

Overview. We introduce the **Agentic Idea Manager** (AIM), a fully autonomous framework for managing research ideas under a limited experimental budget. For a research task τ, AIM maintains the search state $\mathcal{S}_t = (\mathcal{T}, \mathcal{P}_t, \mathcal{M}_t, \mathcal{C}_t, \mathcal{R}_t, \mathcal{D}_t)$, where $\mathcal{T}$ is the fixed task context, $\mathcal{P}_t$ is the current idea pool, $\mathcal{M}_t$ is the memory or lessons that store verified lessons from previous experiments, $\mathcal{C}_t$ is the semantic organization of the pool, $\mathcal{R}_t$ contains ordinal promisingness estimates over clusters and ideas, and $\mathcal{D}_t$ contains past idea–score observations. Following Meng et al. (2026), we adopt its ideator structure and its construction of the task context. Specifically, $\mathcal{T}$ consists of the original task description and an initial research brief that provides supplementary context for ideation[1].

Drawing functional inspiration from Bayesian optimization, AIM utilizes an *Agentic Surrogate* (Section 4.1) to estimate the promise of candidate ideas and an *Agentic Acquisition* (Section 4.2) module to select ideas for solver execution. The results are validated by a *Solution Auditor* (Section 4.3) to update the search state, while a *Resource Planner* (Section 4.4) adaptively allocates the remaining experiment budget for subsequent iterations. The four components form an iteration of AIM. A visual overview is in Figure 2 (a), and its algorithm is in Algorithm 1.

[1] In this work, the research brief is generated from the task description using Claude Code (Anthropic)



(a) **(b)**

Figure 2: **(a)** The AIM pipeline: the *Agentic Surrogate* organizes and estimates candidate idea promise and the *Agentic Acquisition* selects ideas for execution and expands the idea pool. Results are audited by the *Solution Auditor*, while the *Resource Planner* dynamically allocates the budget. **(b)** A visualization of a real pipeline example on the Flash Attention task. The x-axis shows all the cluster themes explored, numbers in each cell are the best scores from respective idea clusters at each round, and the stars indicate the trailing best score.

4.1 Agentic Surrogate: Organize and Estimate

The Agentic Surrogate module constructs an explicit, evidence-conditioned representation of the discovered idea space through two operators: *Organize* and *Estimate*.

Organize. Given the current idea pool $\mathcal{P}_t$ and evaluation history $\mathcal{D}_t$, the Organize operator constructs a cluster map $\mathcal{C}_t = \phi_{\text{org}}(\mathcal{T}, \mathcal{P}_t, \mathcal{D}_t, \mathcal{C}_{t-1})$. Each cluster in $\mathcal{C}_t$ represents a broad research direction and contains semantically related ideas from $\mathcal{P}_t$. Evaluated ideas and their scores serve as empirical landmarks for interpreting related but unevaluated candidates. Because the map is reconstructed from the complete pool at every t, ideas from different generation lineages may be grouped together, and the organization may change as new candidates and evidence become available. A visualization of how the clusters evolve throughout iterations is provided in Figure 2 (b), also demonstrating AIM's interpretability as an idea-driven approach.

Estimate. Conditioned on the organized idea map $\mathcal{C}_t$, and evaluation history $\mathcal{D}_t$, the Estimate operator produces ordinal estimates of promisingness at both the cluster and idea levels: $\mathcal{R}_t = \phi_{\text{est}}(\mathcal{C}_t, \mathcal{D}_t) = \left(\mathcal{R}_t^{\text{cluster}}, \mathcal{R}_t^{\text{idea}}\right)$. Here, $\mathcal{R}_t^{\text{cluster}}$ ranks the cluster-level research directions represented in $\mathcal{C}_t$, while $\mathcal{R}_t^{\text{idea}}$ ranks the unevaluated ideas within each cluster. The estimator considers observed performance, evidence scarcity, semantic novelty, and relevant implementation lessons. We use ordinal estimates because the purpose is to make relative promisingness explicit without requiring the agent to produce calibrated numerical reward predictions. An analysis on the preciseness of the estimation is in Appendix E.3.

4.2 Agentic Acquisition: Dispatch, Solve, and Expand

The Agentic Acquisition module uses the current map $\mathcal{C}_t$ and promisingness estimates $\mathcal{R}_t$ to determine how to select which ideas to evaluate. Rather than relying on fixed selection rules or heuristics, *Dispatch* assigns exploration and exploitation actions using evidence-grounded estimates of the promisingness of each candidate idea. After the dispatched ideas are implemented and evaluated, *Expand* uses the resulting evidence to generate candidates for subsequent iterations.

Dispatch. Given the current idea scaffold $\mathcal{C}_t$, ordinal estimates $\mathcal{R}_t$, and resource plan Π_t (Section 4.4), the Idea Dispatch operator ϕ_{disp} selects a batch of unevaluated ideas: $\mathcal{Q}_t = \phi_{\text{disp}}(\mathcal{P}_t, \mathcal{C}_t, \mathcal{R}_t, \Pi_t)$, where $\mathcal{Q}_t$ contains the ideas assigned to B_t parallel Solver branches, with B_t determined by Π_t. In practice, Dispatch is implemented through two sequential LLM calls. The first call jointly assigns each branch a two-tier action consisting of a cluster-level and an idea-level decision, each selected from $\{\text{EXPLORE}, \text{EXPLOIT}\}$. Exploitation prioritizes highly ranked research directions or ideas, whereas exploration favors underexplored or uncertain alternatives. The second call instantiates these chosen actions by selecting a target cluster and a corresponding idea for each branch. For instance, the cells activated in Figure 2 (b) show which cluster theme was selected at each iteration. Then, Each $x \in \mathcal{Q}_t$ is passed to an independent Solver module, which produces an

executable solution $z = g(x)$, a verifier score $\widetilde{y} = v(z)$, and an execution record. In this work, we mainly use the Gemini Deep Solver utilized in ScientistOne (Meng et al., 2026) (Claude Code solver substitution analysis is in Appendix D.1).

Expand. The Expand operator updates the search space in two stages. First, it extracts reusable lessons $\Delta\mathcal{M}_{t+1}$ from the audited Solver runs and updates the implementation memory: $\mathcal{M}_{t+1} = \mathcal{M}_t \cup \Delta\mathcal{M}_{t+1}$. These lessons capture effective implementation choices, unresolved performance bottlenecks, and compilation, execution, or verification failures to repair or avoid. This design is inspired by evolutionary frameworks that use lessons distilled from experimental outcomes to guide subsequent discovery (Cemri et al., 2026; Liu et al., 2026).

Second, conditioned on the updated memory and audited evaluation history, the Expand operator generates new candidates and adds them to the idea pool:

$$\Delta\mathcal{P}_{t+1} = \phi_{\text{exp}}\left(\mathcal{P}_t, \mathcal{D}_{t+1}, \mathcal{M}_{t+1}\right), \qquad \mathcal{P}_{t+1} = \mathcal{P}_t \cup \Delta\mathcal{P}_{t+1}. \tag{3}$$

For each candidate, the operator selects relevant source ideas or lessons and applies one of four generation modes:

- **Score-guided refinement** preserves the validated components of a high-performing idea while addressing its remaining bottlenecks;
- **Cross-pollination** combines complementary ideas or lessons within or across clusters;
- **Error-guided repair** revises an unsuccessful idea using verifier feedback; and
- **Novel idea generation** introduces a previously unrepresented research direction without requiring an existing parent.

The first three modes develop existing idea lineages using accumulated evidence, whereas new idea broadens the search space and prevents concentration on established directions.

4.3 Solution Auditor

The Solution Auditor prevents invalid or misattributed results from corrupting subsequent search decisions. For each dispatched idea x, implementation z, reported score $\widetilde{y}$, and execution record h, it first audits the resulting solution:

$$\mathcal{F} = \phi_{\text{audit}}(x, z, \widetilde{y}, h) \subseteq \{\text{trivial}, \text{task_mismatch}, \text{idea_mismatch}, \text{reward_hacking}\}, \tag{4}$$

where $\mathcal{F} = \varnothing$ denotes a valid result. The Auditor checks whether the implementation follows the intended idea, satisfies the task requirements, and obtains its score without exploiting the verifier.

The audit outcome determines how the result enters the search state. If trivial, task_mismatch or reward_hacking is detected, the result is discarded and excluded from both the evaluation history and lesson extraction, although the execution still consumes the experimental budget. If the only issue is idea_mismatch, the Auditor reconstructs the input idea and updates the evaluation history accordingly. The score and extracted lessons are then associated with the updated idea.

This audit-and-align procedure addresses the idea–solution integrity challenge by creating a feedback cycle between the two stages: Idea $\rightarrow$ Implementation $\rightarrow$ Audit $\rightarrow$ Align Idea to Implementation. Thus, subsequent search decisions are grounded in the mechanism *actually* evaluated rather than the initially misaligned one.

4.4 Resource Planner

The Resource Planner dynamically distributes a fixed total number of Solver branches across search iterations. First of all, given an execution budget N and B_{tot} total branches, each branch receives a solution execution budget of $N_{\text{branch}} = \lfloor N/B_{\text{tot}} \rfloor$. Then, the planner controls the trade-off between parallel breadth and sequential adaptivity: wider iterations with larger B_t evaluate more ideas concurrently, whereas narrower iterations enable more frequent updates from experimental feedback.

Let $b_t = \sum_{j=1}^{t-1} B_j$ denote the number of branches already dispatched. Given the current state $\mathcal{S}_t$, the remaining branches, and maximum parallelism $B_{\max}$, the planner selects $\Pi_t \equiv B_t = \phi_{\text{plan}}\left(\mathcal{S}_t, B_{\text{tot}} - b_t, B_{\max}\right)$, where $1 \le B_t \le \min\{B_{\max}, B_{\text{tot}} - b_t\}$. Then, the resulting search iteration is

$$I = \min\left\{ i : \sum_{t=1}^{i} B_t = B_{\text{tot}} \right\}, \qquad B_{\text{tot}} N_{\text{branch}} \le N. \tag{5}$$

Table 1: **System Optimization Task Results**. The mean and standard error of 3 runs are reported. All idea-driven approaches take the same research brief as additional context for fair comparisons.

Methods	Scaffold	Flash Attention	Radix Sort	FFT Rust	AES128 Ctr	Z Range Scan	Average
Solution-driven Approaches							
EvoX	Evolutionary	55.9 ± 3.3	55.5 ± 1.8	55.2 ± 0.1	63.1 ± 0.1	43.7 ± 1.6	54.7
AdaEvolve	Evolutionary	85.3 ± 7.6	62.4 ± 3.4	55.6 ± 0.1	65.9 ± 1.1	47.3 ± 2.3	63.3
AIRA (Evolutionary)	Evolutionary	76.3 ± 1.0	66.7 ± 2.5	56.2 ± 0.4	62.8 ± 0.5	51.1 ± 0.8	62.6
AIRA (MCTS)	MCTS	77.9 ± 0.2	61.3 ± 5.8	<u>56.4</u> ± 0.1	62.6 ± 0.3	50.4 ± 1.3	61.7
Idea-driven Approaches							
DeepScientist	List	72.3 ± 3.1	66.5 ± 0.4	54.0 ± 0.7	65.1 ± 1.9	<u>51.4</u> ± 1.9	61.9
AI-Scientist-v2	MCTS	55.0 ± 3.5	65.7 ± 0.5	55.1 ± 1.1	66.3 ± 1.6	42.0 ± 0.5	56.8
Arbor (max depth = 2)	Tree	73.3 ± 5.6	58.5 ± 2.7	54.5 ± 0.5	64.2 ± 1.5	41.4 ± 1.8	58.4
Arbor (max depth = 3)	Tree	77.2 ± 3.8	58.2 ± 4.2	53.5 ± 1.1	**67.2** ± 0.1	46.9 ± 2.1	60.6
ScientistOne	Beam	<u>86.2</u> ± 3.1	<u>68.7</u> ± 1.6	55.9 ± 0.4	65.3 ± 0.2	50.9 ± 1.2	65.4
AIM (Ours)	Autonomous	**90.5** ± 0.8	**69.5** ± 2.3	**56.5** ± 0.5	<u>66.4</u> ± 0.4	**52.1** ± 1.1	**67.0**

The planner changes the frequency of feedback-driven updates while preserving the total branch and execution budgets. Thus, the Resource Planner effectively decides when to exhaust the computational budget and terminate the research pipeline, after which the best scoring solution will be chosen as the final output. The effect of this module is analyzed in the ablation studies in Figure 4.

5 Experiments

Baselines. We compare AIM with various solution-driven and idea-driven methods. Solution-driven baselines include evolutionary solution management approaches like EvoX (Liu et al., 2026), AdaEvolve (Cemri et al., 2026), AIRA (Toledo et al., 2026), and MCTS-based search methods like AIRA (MCTS version). For the idea-driven baselines, we compare with methods with various idea management scaffolds, including list-based DeepScientist (Weng et al., 2026), MCTS-based AI-Scientist-v2 (Yamada et al., 2025), tree-based Arbor (Jin et al., 2026), and elite-preserving beam-search-based ScientistOne (Meng et al., 2026). The backbone LLM is Gemini-3.1-Pro-Preview (Team et al., 2023) for all methods.

Benchmark Tasks. We evaluate our method and baselines on a wide range of 10 tasks from the AutoLab (Xu et al., 2026) suite: System Optimization (Flash Attention, Radix Sort, AES128 Ctr, FFT Rust, and Z-order Range Scan), Model Development (Moving MNIST World Model, Data Select Ifeval), and CUDA (Huffman Canonical Decode, NTT Butterfly, and ICP Correspondence Step). These tasks span different levels of semantic breadth in their viable solution strategies and different degrees of implementation-level optimization depth. Additional details on the baselines, computational budget, benchmark tasks, and implementation setup are provided in Appendix C.

5.1 Results

Results on System Optimization Tasks. Table 1 compares AIM with solution-driven and idea-driven baselines across five systems-optimization tasks. AIM achieves the highest average score of 67.0%, outperforming the strongest idea-driven baseline, ScientistOne (65.4%), by 1.6 points and the strongest solution-driven baseline, AdaEvolve (63.3%), by 3.7 points. The largest gain occurs on Flash Attention, where AIM exceeds ScientistOne and AdaEvolve by 4.3 and 5.2 points, respectively. Overall, these results show that AIM performs robustly across tasks with different search characteristics.

Results on Model Development & CUDA Tasks. In Table 2, we show the results on the long-horizon tasks, comparing with the strongest baseline configuration within each scaffold based on average performance in Table 1. In the table, AIM achieves the highest mean score on three of the five tasks: Moving MNIST World Model, Huffman Canonical Decode, and NTT Butterfly, improving over ScientistOne by 4.2, 0.4, and 2.1 points, respectively. On Data Selection IFEval, AIM achieves 61.8, outperforming the evaluated idea-driven baselines but falling below AdaEvolve (82.7). This strict advantage of AdaEvolve on the Data Selection IFEval task may reflect the task's nature of a smooth, local-search-friendly landscape where score is dominated by fine-tuning a single heuristic recipe (keyword filters, length caps, source balance) rather than by exploring qualitatively different strategies. Its mutation loop compounds refinements against a persistent best program, which we conjecture gives it an advantage over other baselines. Section 6 provides relevant discussions.

Table 2: **Model Development & CUDA Task Results**. Average scores are computed over all five tasks and omitted for methods with missing results. Missing results with '–' is because the baseline methods could not handle the Moving Mnist World Model task that requires multiple file outputs.

Methods	Scaffold	MM World Model	Data Select IE	Huffman Dec.	NTT Butterfly	ICP Corr. Step	Average
Solution-driven Approaches							
AdaEvolve	Evolutionary	–	**82.7** ± 7.1	23.3 ± 1.5	54.9 ± 0.9	18.4 ± 11.8	44.8
AIRA (MCTS)	MCTS	–	38.4 ± 5.9	24.3 ± 10.8	51.6 ± 6.8	54.5 ± 0.9	42.2
Idea-driven Approaches							
DeepScientist	List	14.6 ± 8.7	19.4 ± 19.4	35.5 ± 3.7	42.2 ± 7.1	**55.7** ± 0.5	33.5
Arbor (max depth = 3)	Tree	33.7 ± 11.9	47.0 ± 16.1	37.0 ± 2.3	47.9 ± 5.0	40.8 ± 10.2	41.3
ScientistOne	Beam	57.3 ± 6.3	45.1 ± 1.1	43.0 ± 2.6	56.9 ± 1.3	52.1 ± 1.0	50.9
AIM (Ours)	Autonomous	**61.5** ± 0.9	61.8 ± 3.2	**43.4** ± 1.3	**59.0** ± 1.3	53.5 ± 0.5	**55.8**

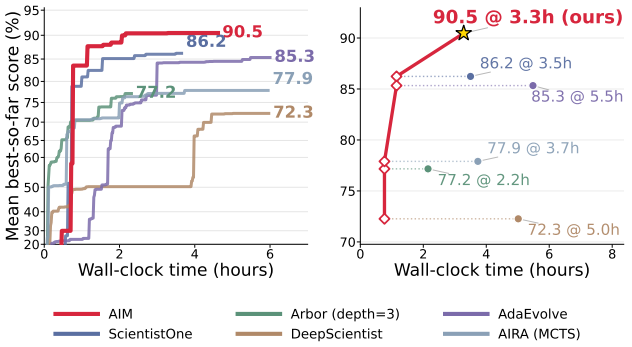


Figure 3: Time Efficiency for Discovery

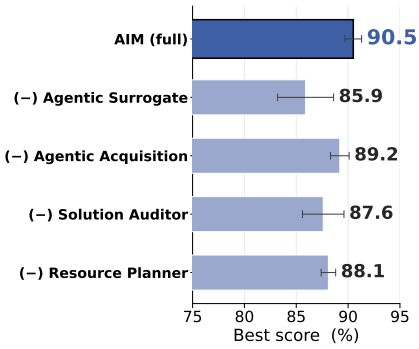


Figure 4: Ablation Studies

Time Efficiency. Figure 3 compares the mean best-so-far score over wall-clock time on the Flash Attention task. AIM reaches the final score levels of all competing baselines within approximately the first 1–2 hours, whereas the baselines require between 2.2 and 5.5 hours to attain those scores. Moreover, AIM reaches its best score of 90.5% after 3.3 hours, exceeding AdaEvolve's 85.3% at 5.5 hours and ScientistOne's 86.2% at 3.5 hours. Thus, AIM not only discovers a better solution, but also reaches competitive performance substantially earlier in the search, up to 3.1× faster compared to the strongest baseline, ScientistOne. Plots for the rest of the tasks are in Appendix D.3.

5.2 Ablation Studies

We ablate each major component of AIM on Flash Attention, as shown in Figure 4. Without the Agentic Surrogate, Organize/Estimate are removed, and Dispatch receives only a shared, unstructured list of ideas. Without Agentic Acquisition, the LLM directly selects ideas from the clustered and ranked pool without explicitly assigning explore/exploit actions. Without the Solution Auditor, all flagged solutions are retained for subsequent lessons and idea generation, idea mismatches are not reconstructed, and audit flags are ignored when computing the final score. Finally, removing the Resource Planner replaces dynamic allocation with the fixed 5×5 execution schedule.

Removing any component reduces performance from the full model's 90.5%. The largest degradation occurs without the Agentic Surrogate (85.9%; −4.6 points), demonstrating the importance of explicitly organizing the idea space and estimating promisingness. The next biggest drop is observed when Solution Auditor is removed, reducing performance to 87.6% (−2.9), showing that it is critical to watch out for invalid or misattributed evidence that can propagate through later iterations. Also, the fixed resource schedule reaches 88.1% (−2.4), while removing explicit acquisition actions yields 89.2% (−1.3). These results support all the central design choices of AIM: structured idea management, grounded and adaptive idea selection, and reliable idea–solution alignment.

5.3 Qualitative Examples

Here, we provide an actual case of AIM's idea selection process. In the example below, we show the example from the third iteration of the Data Select IFEval, which is a task about how to select the right data subset for LLM finetuning for a target benchmark, IFEval (Zhou et al., 2023).

Qualitative Example – Data Select IFEval (iteration 3)

(1) Clusters and rank estimates pool = 33 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[3]	Metadata Stratification & Representations	7	1	**0.377**
2	[2]	Deterministic Heuristics & Lexical Diversity	9	3	0.285
3	[0]	Base-Model IFD & Curriculums	8	2	0.193
4	[1]	Generative Probing & LLM-as-a-Judge	9	2	0.137

"Cluster 3 achieves the highest evaluated score (0.3770) and aligns with strong lessons advocating for metadata distribution and length proxies. [...] Clusters 0 and 1 have much weaker top scores, with lessons explicitly warning against the generative probing approaches found in Cluster 1."

(2) Actions and idea selection

b0 · cluster [3] EXPLOIT · idea EXPLOIT → **Source-Balanced IO-Length Stratification** `rank 1/6`,
"A direct refinement of the best-performing idea so far: it iterates on the successful source-balancing strategy by incorporating output length to filter out terse responses."

b1 · cluster [3] EXPLOIT · idea EXPLORE → **Unsupervised TF-IDF + KMeans Stratification** `rank 5/6`
"To execute an explore action within this top-performing cluster, we select a low-rank idea (5/6) that introduces a completely different mechanism: data-driven semantic boundaries instead of native 'source' metadata."

Outcome: b0 0.119 · **b1** 0.377 (ties best) · **b2** 0.230

First, the Agentic Surrogate's Organizer partitions the 33-idea pool into four clusters, and the Estimator ranks "metadata stratification" first, grounding on the strongest performance of its member idea (0.377), while explicitly demoting "generative-probing & LLM-as-a-judge" on the basis of distilled lessons from earlier failures. In the second part, the Agentic Acquisition's Dispatch proceeds with determining the `Explore`/`Exploit` actions on the cluster-level and idea-level. For instance, Branch 0 and 1 are both assigned the cluster-level `Exploit`, but differs in the idea-level actions. The two branches each choose a high-ranking idea and a low-ranking idea accordingly. The outcome illustrates why both actions are retained: the exploited selection regressed to 0.119, while the explored pick matched the trailing best score (0.377). More qualitative examples are in Appendix E.1.

Following additional experiments and analyses are deferred to the Appendix: ① solver substitution using Claude Code (Appendix D.1); ② in-depth ablation study on the Agentic Surrogate component (Appendix D.2); ③ qualitative analysis of the Organize operator (Appendix E.2); ④ preciseness of the Estimate operator's ordinal predictions (Appendix E.3); ⑤ distribution of the Dispatch/Expand operator's behaviors (Appendix E.4 and E.5); ⑥ effectiveness of the Solution Auditor's idea reconstruction mechanism (Appendix E.6).

6 When to Search Over Ideas? A Formal Understanding

We have distinguished two paradigms for automated research, idea-driven and solution-driven, and focused on improving the latter. We provide a formal view of when the idea-level abstraction can be useful through two questions: (1) *which paradigm can provide broader coverage of the solution space?*, and (2) *when is this additional coverage more valuable for finding competitive directions?*

(1) *Which paradigm has broader coverage of the solution space?* Let each executable solution $z \in \mathcal{Z}$ be associated with an underlying research direction $\pi(z) \equiv x \in \mathcal{X}$, where $\mathcal{X}$ is the finite set of research ideas. Also, assume that $\mathcal{X}$ can be semantically decomposed (Assumption 1), and that z is a faithful implementation of x (Assumption 2). Then, we define:

Definition 1 (Semantic Coverage). *For a set of evaluated solutions $\mathcal{E} \subseteq \mathcal{Z}$, define its Semantic Coverage as the number of distinct research directions represented in it: $C(\mathcal{E}) = |\{\pi(z) : z \in \mathcal{E}\}|$.*

Then, we formally compare the attainable semantic coverage of the two paradigms:

Proposition 1 (Attainable Semantic Coverage). *Let C^{idea} be the semantic coverage of an extreme idea-driven search procedure that renders N distinct experiment executions to N distinct research ideas, and C be the semantic coverage of any search procedure that evaluates at most N executable solutions, including solution-driven approaches. Then, $C \leq C^{idea}$.*

Note that Proposition 1 does not imply that solution-driven search must have lower semantic coverage. A solution-driven method can attain the same bound if its code-level search also produces semantically distinct solutions. The distinction is that idea-driven search makes semantic breadth an explicit and directly controllable property of the search process. Whether maximizing such semantic breadth is beneficial, however, depends on the trait of the research task, which we discuss next.

(2) *When is broader semantic coverage useful?* We next ask *when* the additional coverage is useful for solving a research task. Intuitively, breadth should matter most when there are many possible research directions but only a small subset of them can achieve near-optimal performance. We begin by formalizing the competitive semantic directions and assuming that the directions are exchangeable (Assumption 3), leading to Proposition 2 on the sufficient condition of semantic coverage.

Definition 2 (**Competitive Semantic Directions**). *For a target task τ, let there be K materially distinct research ideas $\mathcal{X}_\tau = \{x_1, \ldots, x_K\} \subseteq \mathcal{X}$. Each direction has an attainable value (e.g., performance measure), $V(x) = \max_{z \in \mathcal{Z}_x} v(z)$, and let $V^\star = \max_{x \in \mathcal{X}_\tau} V(x)$. For $\varepsilon \geq 0$, define the set of ε-optimal directions as $\mathcal{G}_\varepsilon = \{x \in \mathcal{X}_\tau : V(x) \geq V^\star - \varepsilon\}$, and $G_\varepsilon = |\mathcal{G}_\varepsilon|$.*

Proposition 2 (**Semantic Coverage Sufficient Condition**). *Under Definition 2 and Assumption 3, a search procedure with semantic coverage C discovers at least one ε-optimal direction with probability $P_\varepsilon(C) = 1 - \binom{K-G_\varepsilon}{C} / \binom{K}{C}$. Then, a sufficient condition of the semantic coverage $C \geq K/G_\varepsilon \ln(1/\delta)$ guarantees a success probability of $1-\delta$, if the right-hand side is feasible.*

Implication: Tasks with larger effective semantic breadth favor idea-driven search. Combining Propositions 1 and 2, tasks with many materially distinct research directions but relatively few competitive ones benefit more from broad semantic coverage. In such tasks, idea-driven search can be advantageous because it makes semantic breadth explicit and controllable. Conversely, when K/G_ε is small, *i.e.*, the task admits only a few meaningful directions or many directions are similarly competitive, the value of additional coverage is limited, and the two approaches may perform similarly, leaving implementation-level optimization as a potentially more important determinant of performance. Further discussion on practical considerations is in Appendix F.8. Proofs are in Appendix F.3, and theoretical bound visualization is in Appendix F.4.

Empirical support. Our empirical analysis in Figure 5 is consistent with the implication. As a proxy for the breadth of the explored solution space, we embed all intermediate solutions from each method using Gemini-Embedding-001 (Lee et al., 2025), project the embeddings into a 2D PCA space, and measure the area of their convex hull. The box-and-whisker plots compare the distribution of the hull area (see Appendix F.6 for convex hull visualizations). For Flash Attention, which admit diverse algorithmic directions but only a small subset of them yield substantial speedups, idea-driven search approaches explore substantially broader regions than solution-driven approaches, as reflected by larger convex-hull areas. In contrast, for Radix Sort, where the high-level algorithm is fixed and progress depends primarily on code-level optimization, the gap in convex-hull area between the two paradigms becomes considerably smaller. We additionally report pairwise cosine similarity between embeddings as a complementary measure of search diversity in Appendix F.7.

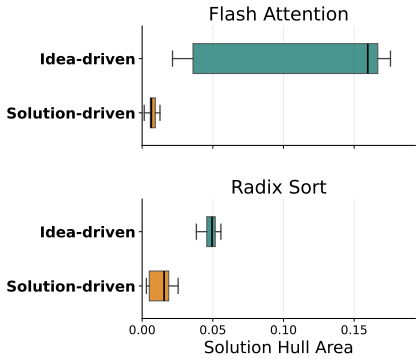


Figure 5: Comparing the convex hull area distribution. Plots for all tasks are in Appendix F.5.

7 Conclusion

In this work, we introduced AIM, a fully autonomous framework for managing and searching research ideas in idea-driven automated research. AIM dynamically organizes an evolving idea pool, explicitly estimates the promise of research directions, adaptively balances exploration and exploitation, and audits implementations before incorporating their outcomes into subsequent search. Across 10 AutoLab tasks, AIM achieves the best overall performance, while reaching strong solutions up to substantially faster compared to baselines. Our theoretical analysis further clarifies when idea-driven search can be advantageous: explicit idea-level allocation makes semantic coverage controllable, and broader coverage becomes more valuable when competitive directions are sparse among many plausible alternatives. Together, these results establish agentic idea management as a transparent and robust approach to budget-constrained automated research.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the writings and generating plots. We have not used generative AI tools to run the experiments for the study outside the agent methods being evaluated. We have reviewed all AI-assisted work. We manually checked the writing faithfully conveys our intended content, and if the plots correctly express values. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or user-facing deployment. Nevertheless, autonomous research agents generate and execute code, which may introduce risks such as insecure implementations, verifier exploitation, or unintended task behavior. We mitigate these risks by conducting all experiments in isolated, containerized benchmark environments with fixed verifiers and by using the Solution Auditor to identify task mismatches and reward-hacking behavior. No generated solutions are deployed in real-world systems. These safeguards cannot eliminate all risks, and human review remains necessary before applying autonomous research systems to consequential scientific or engineering settings.

REPRODUCIBILITY STATEMENT

We provide pseudocode for the complete AIM pipeline in Algorithm 1. Appendix C describes the evaluated benchmarks, baseline configurations, scoring procedure, execution and wall-clock budgets, branch-wise resource allocation, and implementation hyperparameters. The prompt templates for all LLM-driven operators are included in the appendix. We report results over three independent runs using a common backbone model and evaluation environment, with means and standard errors.

REFERENCES

Dhruv Agarwal, Bodhisattwa Prasad Majumder, Reece Adamson, Megha Chakravorty, Satvika Reddy Gavireddy, Aditya Parashar, Harshit Surana, Bhavana Dalvi Mishra, Andrew McCallum, Ashish Sabharwal, et al. Autodiscovery: Open-ended scientific discovery via bayesian surprise. *Advances in Neural Information Processing Systems*, 38:25181–25219, 2026.

Anthropic. Claude code. URL `https://claude.ai`. Computer software. Available via Anthropic's developer platform.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. Researchagent: Iterative research idea generation over scientific literature with large language models. In *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, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.342. URL `https://aclanthology.org/2025.naacl-long.342/`.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, et al. Adaevolve: Adaptive llm driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

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 *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Hui Chen, Miao Xiong, Yujie Lu, Wei Han, Ailin Deng, Yufei He, Jiaying Wu, Yibo Li, Yue Liu, and Bryan Hooi. Mlr-bench: Evaluating ai agents on open-ended machine learning research. *Advances in Neural Information Processing Systems*, 38, 2026a.

Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. Mars: Modular agent with reflective search for automated ai research. In *International Conference on Machine Learning*. PMLR, 2026b.

Hyeong Kyu Choi, Jiatong Li, Wendi Li, Xin Eric Wang, and Sharon Li. Multi-agent llms fail to explore each other. *arXiv preprint arXiv:2607.11250*, 2026.

Thomas Simon Foster, Bassel Al Omari, Tingchen Fu, Thomas Mann, Carl Domond, Lucia Cipolina-Kun, Bhavul Gauri, Muna Aghamelu, Alexander D Goldie, Eryk Helenowski, et al. Ai research preference models. *arXiv preprint arXiv:2608.13940*, 2026.

Aniketh Garikaparthi, Manasi Patwardhan, and Arman Cohan. Researchgym: Evaluating language model agents on real-world ai research. *arXiv preprint arXiv:2602.15112*, 2026.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*, pp. 20271–20309. PMLR, 2024.

Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Daniel S Weld, and Peter Clark. Codescientist: End-to-end semi-automated scientific discovery with code-based experimentation. *arXiv preprint arXiv:2503.22708*, 2025.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, et al. Toward generalist autonomous research via hypothesis-tree refinement. *arXiv preprint arXiv:2606.11926*, 2026.

Akshay Krishnamurthy, Keegan Harris, Dylan J Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? *Advances in Neural Information Processing Systems*, 37:120124–120158, 2024.

Jinhyuk Lee, Feiyang Chen, Sahil Dua, Daniel Cer, Madhuri Shanbhogue, Iftekhar Naim, Gustavo Hernández Ábrego, Zhe Li, Kaifeng Chen, Henrique Schechter Vera, et al. Gemini embedding: Generalizable embeddings from gemini. *arXiv preprint arXiv:2503.07891*, 2025.

Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. Mlr-copilot: Autonomous machine learning research based on large language models agents. *arXiv preprint arXiv:2408.14033*, 2024.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, et al. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Rui Meng, Bhavana Dalvi Mishra, Jiefeng Chen, Chun-Liang Li, Palash Goyal, Mihir Parmar, Yiwen Song, Yale Song, Rajarishi Sinha, Parthasarathy Ranganathan, et al. Scientistone: Towards human-level autonomous research via chain-of-evidence. *arXiv preprint arXiv:2605.26340*, 2026.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, and Tomas Pfister. Ds-star: Data science agent via iterative planning and verification. *arXiv preprint arXiv:2509.21825*, 2025.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Arik, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. *Advances in Neural Information Processing Systems*, 38:116692–116712, 2026.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, et al. Mlgym: A new framework and benchmark for advancing ai research agents. In *Second Conference on Language Modeling*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Lan Pan, Hanbo Xie, and Robert Wilson. Large language models think too fast to explore effectively. *Advances in Neural Information Processing Systems*, 38:102480–102508, 2026.

Jaden Park, Jungtaek Kim, Jongwon Jeong, Robert D Nowak, Kangwook Lee, and Yong Jae Lee. Exploration and exploitation errors are measurable for language model agents. *arXiv preprint arXiv:2604.13151*, 2026.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using llm agents as research assistants. *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 5977–6043, 2025.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M23dTGWCZy.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. Ai-researcher: Autonomous scientific innovation. *Advances in Neural Information Processing Systems*, 38:9481–9520, 2026.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

InternAgent Team, Bo Zhang, Shiyang Feng, Xiangchao Yan, Jiakang Yuan, Runmin Ma, Yusong Hu, Zhiyin Yu, Xiaohan He, Songtao Huang, et al. Internagent: When agent becomes the scientist–building closed-loop system from hypothesis to verification. *arXiv preprint arXiv:2505.16938*, 2025.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Lupidi, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *Advances in Neural Information Processing Systems*, 38:35309–35348, 2026.

Rosni Vasu, Peter Jansen, Pao Siangliulue, Cristina Sarasua, Abraham Bernstein, Peter Clark, and Bhavana Dalvi Mishra. Harpa: A testability-driven, literature-grounded framework for research ideation. *arXiv preprint arXiv:2510.00620*, 2025.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. Deepscientist: Advancing frontier-pushing scientific findings progressively. In *International Conference on Learning Representations*, volume 2026, pp. 47981–48037, 2026.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. In *International Conference on Machine Learning*, pp. 66772–66832. PMLR, 2025.

Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, et al. Autolab: Can frontier models solve long-horizon auto research and engineering tasks? *arXiv preprint arXiv:2606.05080*, 2026.

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

Yunxiang Zhang, Muhammad Khalifa, Shitanshu Bhushan, Grant Murphy, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Mlrc-bench: Can language agents solve machine learning research challenges? *Advances in Neural Information Processing Systems*, 38, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Appendix

Table of Contents

A Extended Related Works **14**

B Agentic Idea Manager Algorithm **15**

C Experimental Details **15**
C.1 Further Baseline and Benchmark Details 15
C.2 Implementation Details 16
C.3 Prompt Templates 17

D Further Experiments **25**
D.1 Solver Substitution with Claude Code 25
D.2 Further Ablation Studies on the Agentic Surrogate 25
D.3 Time Efficiency Plots for All Tasks 26
D.4 Token Cost Analysis 30

E Further Analyses **33**
E.1 More Qualitative Examples 33
E.2 Qualitative Mechanisms of the Organize Operator 35
E.3 Examining the Preciseness of the Estimate Operator 37
E.4 Dispatch Operator Action Analysis 37
E.5 Mode Distribution of the Expand Operator 40
E.6 Effect of Solution Auditor Idea Reconstruction 40

F Theoretical Analyses **45**
F.1 Assumptions 45
F.2 A Note on the Finiteness of Idea Space $\mathcal{X}$ 45
F.3 Proof of Propositions 46
F.4 Theoretical Bound Visualization 48
F.5 Convex Hull Area Box-and-Whisker Plots 48
F.6 Convex Hull Visualization 49
F.7 Supplementary Cosine-Similarity-based Diversity Analysis 49
F.8 Practical Considerations 50

A Extended Related Works

Automated Research with LLM Agents. Recent work has expanded LLM agents from supporting individual scientific tasks to conducting increasingly complete research workflows. At the ideation stage, ResearchAgent (Baek et al., 2025) grounds research idea generation in retrieved literature and iterative feedback, while HARPA (Vasu et al., 2025) develops literature-grounded, testable hypotheses and refines them using experimental evidence. Complementarily, Si et al. (2025) conduct a large-scale expert study of the novelty and quality of LLM-generated research ideas. Broader research systems automate multiple stages of the scientific process: CodeScientist (Jansen et al., 2025) develops a coding-based framework for semi-automated scientific discovery; The AI Scientist (Lu et al., 2024) spans ideation, experimentation, paper writing, and review; and MLR-Copilot (Li et al., 2024), Agent Laboratory (Schmidgall et al., 2025), AI-Researcher (Tang et al., 2026), and InternAgent (Team et al., 2025) similarly automate multi-stage workflows from hypothesis formation to experimental validation and reporting. These works establish the broader setting of automated research, including the end-to-end pipeline resulting in a publishable paper, while our focus is specifically on the "discovery" stage of automated research: *how an agent should search under a limited experimental budget*.

Solution-driven Search over Executable Code. A prominent class of research agents directly searches over executable artifacts, keeping implementation and search tightly coupled. AIDE (Jiang et al., 2025) searches directly in the space of code, while AlphaEvolve (Novikov et al., 2025) uses evolutionary code generation for algorithmic discovery. MLE-Star (Nam et al., 2026) and DS-Star (Nam et al., 2025) iteratively improve machine-learning and data-science solutions through targeted refinement, planning, and verification. More recent systems further develop solution-level search through evolutionary and reflective mechanisms: AdaEvolve (Cemri et al., 2026) performs adaptive LLM-driven zeroth-order optimization, EvoX (Liu et al., 2026) introduces meta-evolution for automated discovery, AIDE (Jiang et al., 2025) utilize an MCTS approach for machine learning tasks, AIRA (Toledo et al., 2026) studies evolutionary and tree-search strategies for machine-learning research, and RPM (Foster et al., 2026) introduces a preference model for candidate solutions. These methods exemplify the *solution-driven* paradigm considered in our work: the executable solution itself remains the primary search object, allowing verifier feedback to be applied directly to subsequent implementation-level refinement.

Idea-driven Search over Research Ideas and Hypotheses. A complementary line of work maintains ideas or hypotheses as explicit intermediate objects before delegating their realization to an implementation agent. The AI Scientist-v2 (Yamada et al., 2025) uses agentic tree search to progressively develop research directions and experiments. DeepScientist (Weng et al., 2026) maintains candidate research directions and progressively selects them using a structured exploration mechanism, while AutoDiscovery (Agarwal et al., 2026) conducts data-driven open-ended scientific discovery through search guided by Bayesian surprise. Arbor (Jin et al., 2026) explicitly organizes hypotheses in a refinement tree, separating hypothesis development from downstream implementation. MARS (Chen et al., 2026b) performs modular reflective search over candidate ideas and solutions. ScientistOne (Meng et al., 2026) similarly makes research ideas explicit and uses an elite-preserving search procedure within its Chain-of-Evidence framework, while additionally preserving traceability between scientific claims, experimental evidence, and implementations.

These *idea-driven* systems reveal several design choices for managing the intermediate idea space. Existing approaches employ structures such as lists (Weng et al., 2026), search trees (Yamada et al., 2025; Jin et al., 2026; Agarwal et al., 2026), or beam-style candidate retention (Meng et al., 2026), together with predefined selection mechanisms such as UCB or tree search. In contrast, our work focuses on making *idea management itself* an autonomous, evidence-conditioned component of the research process. AIM dynamically reorganizes the evolving idea pool, explicitly estimates the relative promise of research directions, and adaptively determines where to explore or exploit based on accumulated experimental evidence. Moreover, because separating ideation from implementation introduces the possibility that a solver may not faithfully realize its intended idea, AIM explicitly audits and reconciles idea–solution mismatches before incorporating results into subsequent search. Thus, our contribution is complementary to prior idea-driven research agents: rather than introducing another fixed idea-search scaffold, we study how the idea space can be autonomously organized, searched, and maintained as evidence accumulates.

Testbeds for Automated Research. In parallel, a growing body of work has made automated experimentation measurable. MLAgentBench (Huang et al., 2024) evaluates agents that modify machine-learning code and iteratively interpret experimental feedback, while MLE-bench (Chan et al., 2025) evaluates agents on competition-style machine-learning engineering. RE-Bench (Wijk et al., 2025) studies frontier AI research-and-development capabilities relative to human experts, and MLGym (Nathani et al., 2025) provides diverse environments for evaluating AI research agents. More recent benchmarks broaden evaluation toward open-ended research problems, including MLRC-Bench (Zhang et al., 2026), MLR-Bench (Chen et al., 2026a), and ResearchGym (Garikaparthi et al., 2026). We primarily evaluate on AutoLab (Xu et al., 2026), which provides long-horizon research and engineering tasks with executable verifiers and enables controlled comparison of search strategies in systems optimization.

Overall, prior work has made substantial progress in both direct solution optimization and end-to-end automated research. Our work builds on this literature by explicitly distinguishing *solution-driven* search over executable solutions from *idea-driven* search over ideas followed by implementation, and focuses on the latter's distinctive challenges: maintaining an evolving semantic representation of candidate ideas, making evidence-grounded decisions about which ideas deserve expensive implementation, and preserving idea–solution integrity throughout the search process.

B Agentic Idea Manager Algorithm

Algorithm 1 AIM: Agentic Idea Manager

Require: Task context $\mathcal{T}$; initial idea pool $\mathcal{P}_1$; execution budget N; total branches B_{tot}; maximum parallelism $B_{\max}$; time limit H.

$\mathcal{D}_1, \mathcal{M}_1 \leftarrow \varnothing$; $b \leftarrow 0$; $t \leftarrow 1$; $N_{\text{branch}} \leftarrow \lfloor N/B_{\text{tot}} \rfloor$
while $b < B_{\text{tot}}$ **and** elapsed time $< H$ **do**
  $B_t \leftarrow \begin{cases} 5, & t = 1, \\ \phi_{\text{plan}}(\mathcal{S}_t, B_{\text{tot}} - b, B_{\max}), & t > 1 \end{cases}$ ▷ Resource Planner
  $\mathcal{C}_t \leftarrow \phi_{\text{org}}(\mathcal{T}, \mathcal{P}_t, \mathcal{D}_t, \mathcal{C}_{t-1})$ ▷ Agentic Surrogate: Organize
  $R_t \leftarrow \phi_{\text{est}}(\mathcal{C}_t, \mathcal{D}_t)$ ▷ Agentic Surrogate: Estimate
  $\mathcal{Q}_t \leftarrow \phi_{\text{disp}}(\mathcal{P}_t, \mathcal{C}_t, R_t, B_t)$ ▷ Agentic Acquisition: Dispatch
  for all $x \in \mathcal{Q}_t$ **in parallel do**
    $(z_x, y_x, h_x) \leftarrow \phi_{\text{solve}}(\mathcal{T}, x; N_{\text{branch}})$
  end for
  $\mathcal{V}_t \leftarrow \phi_{\text{audit}}\big(\{(x, z_x, y_x, h_x) : x \in \mathcal{Q}_t\}\big)$ ▷ Solution Auditor
  $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \cup \{(x^+, y) : (x^+, z, y, h) \in \mathcal{V}_t\}$
  $\mathcal{M}_{t+1} \leftarrow \mathcal{M}_t \cup \phi_{\text{lesson}}(\mathcal{V}_t)$
  $\Delta\mathcal{P}_t \leftarrow \phi_{\text{exp}}(\mathcal{P}_t, \mathcal{D}_{t+1}, \mathcal{M}_{t+1})$ ▷ Agentic Acquisition: Expand
  $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_t \cup \{x^+ : (x^+, z, y, h) \in \mathcal{V}_t\} \cup \Delta\mathcal{P}_t$
  $b \leftarrow b + B_t$; $t \leftarrow t + 1$
end while
return $\arg\max_{(x,y)\in\mathcal{D}_t} \; y$

C Experimental Details

C.1 Further Baseline and Benchmark Details

Baselines and budgets. All baseline agents use `gemini-3.1-pro-preview` (Team et al., 2023) as their underlying language model. Each method is allowed at most N experiment executions and a wall-clock budget limit is set per run. For the system optimization tasks, we impose a budget of at most 300 experiment executions and a hard wall-clock limit of 6 hours; for the model development tasks, we set at most 60 executions and wall-clock limit of 24 hours; for the CUDA tasks, we set at most 60 executions and wall-clock limit of 12 hours. If either of the budget is exhausted, the process terminates. For baseline methods that support parallel search or execution, we set the maximum parallelism to five workers, to keep it commensurable with AIM which usually spends at most five

branches per iteration. All idea-driven search baselines receive the same initial research brief. We perform three independent runs for each (method, task) pair and report the mean and standard error.

AutoLab tasks. We evaluate on ten tasks from AutoLab (Xu et al., 2026) spanning three categories. *System optimization* (CPU): Flash Attention, Radix Sort, FFT Rust, AES128 Ctr, and Z-order Range Scan, which respectively optimize scaled dot-product attention in C, sorting of 50 million unsigned integers in C, a 32,768-point real-valued DFT in Rust, AES-128-CTR encryption of 256 MiB in C, and two-dimensional range-count queries over a Rust spatial index. *Model development* (GPU): MM World Model, which trains a video world model from scratch on Moving MNIST and is scored by PSNR on 10-step rollouts under a fixed four-hour training budget, and Data Select IE, which selects 5,000 training samples from a 50,000-sample pool drawn from 19 sources for LoRA fine-tuning of Qwen2.5-3B-Instruct and is scored by prompt-level strict accuracy on IFEval. *CUDA kernels* (GPU): Huffman Canonical Decode, which decodes K independent canonical-Huffman bitstreams to their byte payloads; NTT Butterfly, which applies an in-place forward number-theoretic transform over the Goldilocks prime field to batched 64-bit rows; and ICP Correspondence Step, which performs one Iterative-Closest-Point correspondence step, i.e., nearest-neighbor search against a prebuilt KD-tree together with accumulation of the cross-covariance, residual error, and correspondence count.

Each AutoLab task provides a natural-language instruction, a containerized environment, an editable codebase containing a correct but deliberately weak baseline (an inefficient implementation for the optimization and CUDA tasks, or a simple training or selection recipe for the model-development tasks), and a local evaluation script. During search, an agent may repeatedly edit the implementation, execute it, inspect its score and correctness feedback, and refine subsequent solutions. The task additionally contains a human-written reference implementation and a held-out verifier. The reference solution is used only to calibrate the scoring scale and is not exposed to the research agent.

Evaluation and scoring. The verifier first checks functional correctness and task-specific constraints; a solution that fails these checks or does not improve upon the baseline receives zero reward. Valid solutions are scored on one of two scales, depending on the task category.

Runtime tasks (system optimization and CUDA). Because runtime depends on the execution environment, for the system-optimization tasks we run both the baseline and reference implementations on the same local hardware used to evaluate generated solutions. Let $t_{\mathcal{B}}$, $t_{\mathcal{R}}$, and $t(z)$ denote the runtimes of the baseline, reference, and generated solution z, respectively. Following AutoLab, the normalized score is

$$s(z) = \begin{cases} 0, & \text{if } z \text{ is invalid or } t(z) \geq t_{\mathcal{B}}, \\ \mathrm{clip}\left(\frac{1}{2}\frac{\log(t_{\mathcal{B}}/t(z))}{\log(t_{\mathcal{B}}/t_{\mathcal{R}})}, 0, 1\right), & \text{otherwise.} \end{cases} \tag{6}$$

The baseline therefore receives $s = 0$, while matching the reference solution gives $s = 0.5$. Solutions outperforming the reference receive scores above 0.5, up to a maximum of 1.0.

Model-development tasks. These tasks are scored on a task-specific quality metric $m(z)$ (higher is better) rather than runtime. Following AutoLab, the score interpolates linearly between a baseline anchor $m_{\mathcal{B}}$ and a reference anchor $m_{\mathcal{R}}$:

$$s(z) = \begin{cases} 0, & \text{if } z \text{ is invalid,} \\ \mathrm{clip}\left(\frac{m(z) - m_{\mathcal{B}}}{m_{\mathcal{R}} - m_{\mathcal{B}}}, 0, 1\right), & \text{otherwise,} \end{cases} \tag{7}$$

with $(m_{\mathcal{B}}, m_{\mathcal{R}}) = (14.0, 20.0)$ dB PSNR for MM World Model and $(0.38, 0.48)$ IFEval accuracy for Data Select IE, which are provided by the benchmark.

We multiply $s(z)$ by 100 when reporting percentage scores in the main paper.

C.2 Implementation Details

Branch-wise execution budgets. We distinguish the total number of Solver branches, B_{tot}, from the number of branches executed concurrently at iteration t, B_t. Before each run, the total execution

budget is divided equally among the Solver branches:

$$N_{\text{branch}} = \left\lfloor \frac{N}{B_{\text{tot}}} \right\rfloor. \tag{8}$$

For instance, if we use $N = 300$ and $B_{\text{tot}} = 25$, it gives each branch a fixed budget of 12 experiment executions. A branch may reason over multiple rounds, modify its intermediate implementation, and invoke the task evaluator up to 12 times. Every attempted execution counts toward this budget, including executions whose outputs are subsequently rejected by the Solution Auditor.

Initial Idea Pool. We initialize the idea pool using the Ideator from ScientistOne (Meng et al., 2026). The Ideator generates candidate ideas conditioned on the task context, after which its feasibility critic filters out ideas that are impractical or incompatible with the task requirements. This process yields an initial pool of approximately ten feasible ideas.

Agentic Surrogate. To avoid producing either a degenerate single cluster or an excessively fragmented map, the Organize operator is constrained to create between $C_{\min}$ and $C_{\max}$ semantic clusters, which we set to $C_{\min} = 2$ and $C_{\max} = 5$. The same bounds are used across all tasks and iterations. Within these constraints, the operator autonomously determines the number of clusters, their semantic descriptions, and the assignment of ideas to clusters.

Agentic Acquisition. At each round, the Expand operator pushes new ideas into the idea pool based on the four modes described in the main paper: push_score, cross_pollinate, fix_error, and new_idea. To avoid an overflow of new ideas, we cap the maximum number of ideas for each branch to 3. For instance, if there were 5 branches in an iteration, the maximum number of idea that can be added to the pool in that iteration is 15.

Resource Planner. The first search iteration always launches 5 parallel Solver branches, ensuring an initial breadth of exploration before sufficient experimental evidence is available for adaptive planning. In subsequent iterations, the Resource Planner dynamically plans branches for future iterations. To ensure breadth of exploration at each iteration, we also set a minimum of 3 branches and a maximum of 10 branches every round. Thus,

$$B_1 = 5, \qquad 3 \le B_t \le 10 \;\; (t > 1), \qquad \sum_{t=1}^{K} B_t = B_{\text{tot}}. \tag{9}$$

The planner therefore changes how the fixed set of branches is distributed across iterations, while the per-branch execution budget remains fixed at N_{branch}. Empirically, however, the planner agent prefers to assign smaller number of parallel branches than 5.

C.3 Prompt Templates

Here, we show the prompt templates used by LLM-driven stages of AIM. Each box contains an operator's prompt package: the system prompt followed by the USER PROMPT with per-call payload fields. Angle-bracketed placeholders (<...>) name the runtime content that fills each slot at prompt-render time.

Agentic Surrogate: Organize Prompt Template

```
You are the meta research agent overseeing a parallel
research-discovery loop. Your job at every iteration is to ORGANIZE
the current idea pool into K thematic CLUSTERS so the downstream
allocator can spread parallel branches across distinct research
approaches.

Hard constraints:
  - K within [K_min, K_max] from the user prompt (read every iter; do
not assume a default).
  - Every idea id in the pool MUST appear in exactly ONE cluster.
  - Each cluster contains at least one idea.
```

```
  - Themes must be meaningful -- group by mechanism / assumption /
approach; avoid catch-alls.

OUTPUT -- single JSON object inside ```json fences:
{
  "K": <int>,
  "clusters": [
    { "label": "Short name (<=6 words)",
      "theme": "1-3 sentences on what unifies these ideas",
      "idea_ids": ["<id>", ...],
      "rationale": "why these ideas belong together" }, ...
  ]
}
```

```
USER PROMPT:
## Benchmark task
<benchmark task description>

## Iteration
This is iteration <current iter number>. You have B =
<parallel_branches this iter> parallel branches. K_max = <upper bound
on K> is a HARD MAX -- NOT a target. Pick K in [2, <K_max>] that best
matches the pool's thematic structure.

## Prior organization
<previous iter's clustering JSON, or null on iter-1>

## Pool snapshot
<full idea pool: per-idea id, source (ideator/refined/fresh_inject),
 parent_id, latest_iter, latest_score, success, title, abstract>
```

Agentic Surrogate: Estimate Stage 1 Prompt Template

```
You are ranking research-idea CLUSTERS by promising-ness. Your
rankings feed a downstream allocator that decides how many parallel
research branches to spend on each cluster.

You see: label, theme, n_evaluated / n_ideas, mean_evaluated_score,
best_evaluated_score. Nothing more -- you are judging the DIRECTION,
not individual ideas.

## Ranking scheme -- DENSE RANKS
Integers from 1 (most promising). Ties allowed -- same integer. Next
distinct rank after a tie is +1, not skipped.
  Valid:   [1,2,2,3,4]   [1,1,1,1,1]   [1,2,3,4,5]
  Invalid: [1,2,2,4,5]   (gap after tie)

## Guidance
- High `best_evaluated_score`     → LOW rank (promising).
- Many evaluated, all scored poorly → HIGH rank (exhausted).
- n_evaluated=0 → rank on THEME quality vs task's known bottlenecks.
- Do NOT collapse everything to rank 1.

## Output -- JSON, fenced with ```json:
{ "cluster_ranks": [{"cluster_idx": <int>, "rank": <int>}, ...],
  "rationale": "1-3 sentences -- cite specific clusters" }
```

```
USER PROMPT:
## Benchmark task
<benchmark task description>
```

```
## Scoring scale
<score direction (minimize/maximize), units, explicit numeric range
if known>

## Clusters to rank (<N> total)
<per-cluster block: cluster_idx, label, theme, n_ideas, n_evaluated,
 mean_evaluated_score, best_evaluated_score>
```

Agentic Surrogate: Estimate Stage 2 Prompt Template

```
You are ranking research IDEAS within a single cluster by
promising-ness. Your ranking feeds the downstream allocator that
picks which idea each branch attempts.

You see: unevaluated candidates within ONE cluster (title, short
hypothesis, abstract). The cluster's LABEL and THEME give you the
direction they share.

## Ranking scheme -- DENSE RANKS (same rules as Stage 1)
  Valid:   [1,2,2,3]   [1,1,2,3]   [1,2,3,4]
  Invalid: [1,2,2,4]   (gap after tie)

## Guidance
- Rank by how strongly mechanism, novelty, and specificity predict a
good score.
- LOW rank for ideas that clearly instantiate the cluster's theme
with a concrete, testable optimization.
- HIGH rank for ideas that repeat existing themes or lack mechanism
specificity.
- Do NOT collapse everything to rank 1 unless truly
indistinguishable.

## Output -- JSON, fenced with ```json:
{ "idea_ranks": [{"idea_id": "<id>", "rank": <int>}, ...],
  "rationale": "1-3 sentences" }
```

```
USER PROMPT:
## Benchmark task
<benchmark task description>

## Scoring scale
<direction, units, numeric range if known>

## This cluster
label: <cluster's short label>
cluster_rank: <cluster's rank from Stage 1>
theme: <cluster's 1-3 sentence theme>

## Ideas to rank
<per-idea block: idea_id, title, short hypothesis, abstract>
```

Agentic Acquisition: Dispatch Stage 1 Prompt Template

```
You are the meta-allocator. In THIS step you decide the SHAPE of the
search for this iter (which cluster each branch is assigned to, plus
cluster- and idea-level actions), NOT the specific ideas. Stage B
picks the concrete ideas next.

## Vocabulary
```

```
cluster_action = "exploit" → invest branches in an already-producing
cluster.
cluster_action = "explore" → invest in an underprobed cluster.
idea_action = "exploit"    → within the cluster, next step picks the
most-promising unevaluated idea.
idea_action = "explore"    → within the cluster, next step picks a
novel-mechanism unevaluated idea.

## Rank semantics (v1.6)
- cluster_rank=1 → most promising (prefer for exploit).
- Higher rank → less promising. If well-probed, may be exhausted; if
under-probed, holds explore value.
- Ties allowed and common -- treat tied clusters as comparable.

## Output -- JSON:
{
  "cluster_decisions": [
    {"cluster_idx": <int>, "action": "exploit"|"explore",
"n_branches": <int>, "rationale": "..."}, ...
  ],
  "branch_action_assignments": [
    {"branch": <int>, "cluster_idx": <int>, "cluster_action":
"exploit"|"explore",
     "idea_action": "exploit"|"explore", "rationale": "..."}, ...
  ],
  "rationale_summary": "..."
}

Validator constraints: each cluster_idx appears at most once in
cluster_decisions; sum(n_branches) == B; EXACTLY B
branch_action_assignments; each branch's cluster_action matches its
cluster's action.

A run that puts everything on one cluster, or labels every action
"exploit", is failing the loop's purpose. Do NOT output any `idea_id`
in this step.
```

```
USER PROMPT:
## Benchmark task
<benchmark task description>

## Iteration
iter_idx=<current iter, 0-indexed>, total_iters=<total planned
iters>, B=<parallel branches this iter>

## Previous iter's allocation
<the last iter's allocation.json -- rationale summary + per-branch
picks -- or null on iter-1>

## Score history
<per-branch score records so far: iteration, branch, score, success,
role, idea_id, audit_flags>

## Clusters
<per-cluster block: cluster_idx, label, theme, member counts,
 actual-score distribution (min/max/mean of evaluated members),
 cluster_rank, top-K unevaluated preview by within_cluster_rank>
```

Agentic Acquisition: Dispatch Stage 2 Prompt Template

```
You are the idea-selection agent for ONE branch. Stage A already
decided this branch's cluster and (cluster_action, idea_action). Pick
the SPECIFIC unevaluated idea from the assigned cluster that best
matches idea_action.

## Action interpretation (v1.6 rank semantics)
Each unevaluated member carries `within_cluster_rank=R/M` (1=most
promising in the cluster, M=cluster size) and `cluster_rank=K`.

For idea_action = "exploit":
- Pick the LOWEST within_cluster_rank member.
- Ties are common -- break using abstract + cluster context.
- Use evaluated members as evidence -- what mechanisms paid off?
- A refined child whose parent scored well is often the right exploit
pick even if within_cluster_rank isn't strictly lowest.

For idea_action = "explore":
- Pick a HIGH within_cluster_rank member -- evaluating it gives more
info gain than another shot at the top.
- Prefer mechanisms NOT YET tried by evaluated members.
- Refined children on NEW trajectories beat refined children on
already-explored trajectories.
- DO NOT pick the lowest within_cluster_rank -- that is exploit.

## Hard constraints
- `idea_id` MUST be UNEVALUATED and in the assigned cluster.
- `idea_id` MUST NOT be in the "Already picked by other branches"
list.
- Evaluated members are shown for REASONING only -- never pick one.

## Output -- JSON only, no prose:
{ "branch": <int>, "idea_id": "<unevaluated-id>", "rationale": "<2-4
sentences>" }

## Iter-1 suffix (appended only when iter_idx == 0):
No idea has been evaluated yet -- every predicted_score is an LLM
estimate. For every branch, set idea_action: "exploit" and pick the
highest-predicted_score idea. Iter-1 is for validating the
estimator's top predictions.
```

```
USER PROMPT:
## Benchmark task
<benchmark task description>

## This branch
branch_idx=<branch index within [0,B)>, cluster_idx=<assigned cluster
index>, cluster_action=<exploit or explore, from Stage A>,
idea_action=<exploit or explore, from Stage A>

## Cluster
label=<cluster label>, theme=<cluster 1-3 sentence theme>

## Cluster members
<per-member block: id, title, abstract, was_assigned,
 for evaluated: latest_score, latest_iter, latest_success,
 for unevaluated: within_cluster_rank, cluster_rank,
n_cluster_members, lineage (parent_id, parent_score,
trajectory_trend)>

## Already picked by other branches
<sorted list of idea_ids picked by earlier-index branches this iter>
```

Agentic Acquisition: Expand Prompt Template

```
You are the swarm-aware refiner for a parallel research-search loop.
Each iteration, multiple branches independently try ideas; your job
is to take ONE branch's outcome plus the full swarm context and
propose refined ideas that should rejoin the pool.

You see: this branch's parent idea + evaluation; sibling branches'
outcomes this iter; cluster context; prior-iter top-3 / bottom-3
across the run; distilled lessons.

You have up to <max proposal slots for this branch> PROPOSAL SLOTS.
Emit AT MOST ONE proposal per action kind; skip any that don't apply.
Quality over quota.

The 4 actions (menu):

- `fix_error` -- branch failed. Propose a targeted fix. Skip if it
succeeded.
- `push_score` -- branch succeeded with headroom. Propose a tighter
implementation.
- `cross_pollinate` -- a specific sibling outcome OR lesson gives a
compositional improvement.
    HARD: base reasoning ONLY on ## Lessons + ## Sibling outcomes.
    Cite exactly ONE source:
      * sibling_referenced: <sibling_idea_id from Sibling outcomes>,
OR
      * lesson_context_idea: <tag or context_idea from Lessons>
    Both empty, both set, or unknown id → proposal dropped.
- `new_idea` -- no in-cluster improvement to offer; propose an
ORTHOGONAL direction. Parentless.
    Workflow: (1) read Cluster context for the diversity BASELINE;
              (2) name 1-2 orthogonal directions in the Abstract;
              (3) instantiate one.
    If best you can offer is a baseline variant → use cross_pollinate
or push_score, NOT inject_fresh.

OUTPUT -- YAML:
ideas:
  - Name: <short snake_case>
    Title: <refined idea title>
    Short Hypothesis: <one sentence>
    Abstract: <2-3 sentences, <=200 words>
    Experiments:
      - hypothesis: <what this experiment tests>
        experiment_plan_steps: [<concrete step>, ...]
    Risk Factors and Limitations: <known risks>
    kind: fix_failure | push_score | cross_pollinate | inject_fresh
    parent_id: <omit for default parent; empty for inject_fresh>
    sibling_referenced: <id>   # REQUIRED for cross_pollinate -- pick
ONE
    lesson_context_idea: <tag or id>  # of these two, not both, not
neither

Hard rules: AT MOST ONE proposal per kind; empty list acceptable; do
not include the parent idea verbatim.

--------------------------------------------------------------------

USER PROMPT:
## Benchmark task
<benchmark task description>

## This branch (<branch index>): parent idea
```

```
<parent idea's full YAML: name, title, short hypothesis, abstract,
experiments, risk factors -- the same the solver received>

## Evaluation
<evaluator output: score, success, one-line eval detail (crash msg,
failure mode, or short success summary)>

## Sibling outcomes this iteration
<per-sibling row: branch idx, idea id, idea title, score, success,
one-line eval gist -- from ALL sibling branches this iter>

## Cluster context
<parent's cluster: label, theme, evaluated-member score distribution,
sibling-cluster overview so the model can identify orthogonal
directions>

## Prior-iter signals
<top-3 and bottom-3 ideas across the whole run so far, with
iteration, branch, score, cluster label - for identifying durable
winners / dead ends>

## Lessons
<consolidated lessons pool (capped at <lessons_cap>): tag, lesson
text, optional context_idea id>
```

Solution Auditor Prompt Template

```
You are a solution auditor for an automated research pipeline. Your
job is to determine whether a solution is LEGITIMATE -- i.e., it
actually solves the stated task in a meaningful way rather than
gaming the evaluation.

You will be given:
1. The TASK INSTRUCTION defining what problem must be solved
2. The IDEA the agent was supposed to implement
3. The SOLUTION CODE the agent produced
4. The EVALUATION METRICS returned by the evaluator

Check for these four failure modes:

**Reward Hacking**: The solution manipulates, monkey-patches, or
reverse-engineers the evaluator to inflate its score without
genuinely solving the task.
**Idea-Solution Mismatch**: The solution ignores the proposed idea
and implements something unrelated.
**Task Mismatch**: The IDEA (and hence the CODE) solves a DIFFERENT
problem than the TASK specifies (e.g. approximation instead of exact,
weaker guarantee than required). Orthogonal to Idea-Solution
Mismatch; both can be flagged simultaneously.
**Trivial Solution**: No-op, constants, verbatim baseline.

Respond with a JSON object (no markdown fences):
{
  "legit": true/false,
  "flags": ["reward_hacking", "idea_mismatch", "task_mismatch",
"trivial"],
  "confidence": 0.0-1.0,
  "reasoning": "one paragraph explanation",
  "reconstructed_idea": { ... }   // REQUIRED iff `flags` contains
"idea_mismatch"
}

Only include flags that apply.
```

```
`reconstructed_idea` schema (matches ideator's canonical shape):
{ "title", "short_hypothesis", ["twist" iff mode="unconventional"],
  "grounded_in":[...], "addresses", "mode":
"conservative"|"unconventional" }

---

USER PROMPT:
## TASK INSTRUCTION
<benchmark task description>

## IDEA
<the idea assigned to this branch: title, short hypothesis, abstract,
 experiments, risk factors -- the same YAML the solver received>

## SOLUTION CODE
<the solver's final produced source code>

## EVALUATION METRICS
<evaluator output JSON>
```

Resource Planner Prompt Template

```
You are the DYNAMIC resource planner for a parallel research-search
loop. The total branch-budget is FIXED (= parallel_branches ×
iterations). Iter-1 is pinned to `parallel_branches` for
deterministic bootstrap. Every remaining branch has the same
per-branch compute budget.

**Length is your primary lever.** The `iterations` value from the
launcher is used only to compute the total branch-budget -- NOT a
target for len(plan). Do NOT match the launcher's iteration count as
a reflex.

Your job: distribute REMAINING branches across as many additional
iterations as you think optimal.
  - Spend EXACTLY `remaining` branches (no over- or under-shoot).
  - Each iter you add has BETWEEN 1 AND max_branches_per_iter
branches.
  - PREFER AT LEAST <B_min> BRANCHES PER ITERATION.
  - You cannot revise the frozen prefix.

## Preferred shapes (choose length from evidence, not the launcher's
iterations knob)
- Long tail of 3-branch iters -- refinement > parallelism.
- Iterate-and-refine (5-10 iters, 3-5 branches) -- balanced.
- Broad-then-deep -- heavy early, tapering to polish.
- Concentrate-and-terminate (2-3 large iters).

## Hard constraints (validated)
- sum(plan) == total_runs      # EXACTLY
- plan[0] == parallel_branches # iter-1 pinned
- 1 <= plan[i] <= max_branches_per_iter
- plan[:iter_idx] == frozen_prefix verbatim
- len(plan) > iter_idx         # at least one more iter beyond frozen
prefix

## Output -- JSON in ```json fences:
{ "plan": [<int>, <int>, ...],
  "rationale": "2-4 sentences on why THIS LENGTH and this
distribution" }
```

```
## Benchmark task
<benchmark task description>

## Position
iter_idx=<current iter, 0-indexed>,
total_runs=<total branch budget = parallel_branches × iterations>,
parallel_branches=<launcher's -n value>,
max_branches_per_iter=<hard cap per-iter, from launcher>,
frozen_prefix=<list of per-iter branch counts already spent>

## Evidence
best_so_far=<best clean score across all evaluated branches so far>,
target=<current adaptive target score or null>,
evaluated_scores=<list of evaluated branches' raw scores, in order>,
estimated_scores=<list of unevaluated candidates' predicted scores>

## Cluster summary
<per-cluster: label, theme, n_evaluated, best_evaluated_score>

## Lessons
<consolidated lessons pool (tag + lesson text)>
```

D Further Experiments

D.1 Solver Substitution with Claude Code

To evaluate whether the effectiveness of AIM is tied to a specific solver implementation and to assess how much an explicit idea management scaffold benefits existing coding agents, we conduct a solver substitution experiment. In this setting, we replace the default Gemini Deep Solver with Claude Code (Anthropic). We compare AIM with ScientistOne (Meng et al., 2026) across all five AutoLab tasks. ScientistOne represents the strongest idea-driven baseline from our main experiments, configured to delegate implementation to Claude Code. Similarly, our proposed AIM framework uses Claude Code as the downstream solver agent across parallel execution branches.

Table 3 summarizes the performance across the benchmark tasks. Across all evaluated domains, AIM consistently outperforms or matches ScientistOne across all five benchmarks when both utilize the Claude Code solver.

Table 3: **Solver Substitution**. Comparison of AIM and ScientistOne with Claude Code Solvers.

Task	ScientistOne (Claude-Code Solver)	AIM (Claude-Code Solver)
Flash Attention	81.6 ± 3.5	**84.5** ± 0.3
Radix Sort	67.4 ± 0.6	**67.7** ± 0.2
FFT Rust	54.6 ± 0.3	**55.4** ± 0.2
AES128 Ctr	66.9 ± 0.3	**67.6** ± 0.6
Z-order Range Scan	**52.2** ± 0.1	**52.2** ± 0.3

D.2 Further Ablation Studies on the Agentic Surrogate

Table 4 presents further in-depth ablation studies on the Agentic Surrogate module. We evaluate several alternative configurations to isolate the contributions of its sub-components. The "without Organize" setting bypasses clustering entirely, treating the entire list of ideas as a single unified cluster before passing them to the Estimate operator for ranking. The "without Estimate" configuration removes the estimation step, passing only the unranked cluster information from the Organize step directly to the Dispatch operator within the Agentic Acquisition module. The "without Agentic Surrogate" setting removes both the Organize and Estimate components, serving as the same

baseline ablation described in the main manuscript. Furthermore, we test an "Embedding-based Organize" variant that performs K-means clustering (with an automatically determined K) based on idea embeddings generated by the Gemini-Embedding-001 (Lee et al., 2025) model. Finally, the "Direct Score Estimation" configuration replaces ordinal ranking estimates with direct raw score predictions.

Table 4: Further ablation studies on the Agentic Surrogate module.

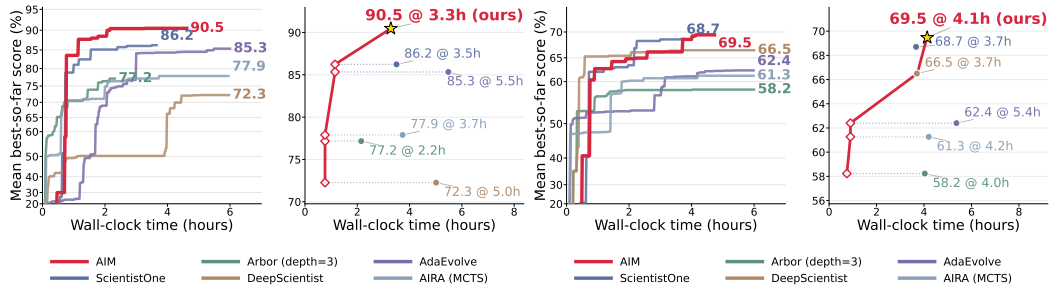
Method	**Flash Attention Scores**
Full AIM	90.5 $\pm$ 0.8
without Organize	89.0 $\pm$ 0.4
without Estimate	87.9 $\pm$ 1.2
without Agentic Surrogate (both Organize and Estimate)	85.9 $\pm$ 2.7
Embedding-based Organize	83.4 $\pm$ 2.8
Direct Score Estimation	89.3 $\pm$ 1.2

The results from the ablation study demonstrate the critical importance of both the LLM-driven semantic organization and the ordinal estimation components. The Full AIM achieves the highest performance with a Flash Attention score of 90.5 ± 0.8. Removing either the Organize or Estimate operators degrades performance to 89.0 ± 0.4 and 87.9 ± 1.2, respectively, while removing the Agentic Surrogate entirely results in a substantial drop to 85.9 ± 2.7.

Interestingly, the Embedding-based Organize approach yields the lowest score of all configurations (83.4 ± 2.8), performing even worse than the complete removal of the surrogate. Our interpretation of this result is that standard distance-based clustering on generic semantic embeddings fails to capture the nuanced, task-specific structural relationships that the LLM-based Organize operator can successfully identify and group. Additionally, substituting ordinal ranking with Direct Score Estimation (89.3 ± 1.2) underperforms the Full AIM. This indicates that language models are generally more reliable at performing relative, ordinal comparisons of research ideas than they are at predicting absolute, uncalibrated performance scores from raw text.

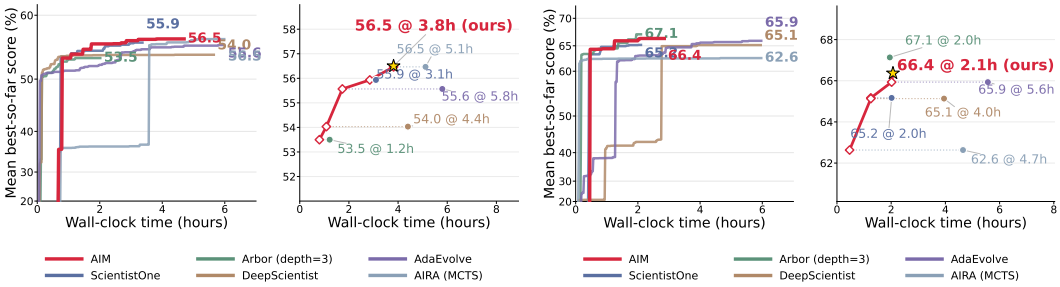
D.3 Time Efficiency Plots for All Tasks

In this section, we provide the time efficiency plot in Figure 6 and Figure 7 for all the 10 tasks evaluated. Overall, our AIM provides an efficient framework, attaining top scores in a shorter amount of time, compared to the idea-driven and solution-driven baselines.



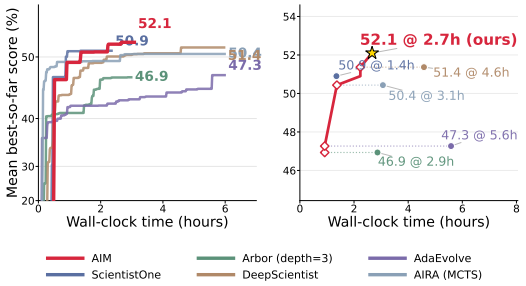
(a) Flash Attention

(b) Radix Sort



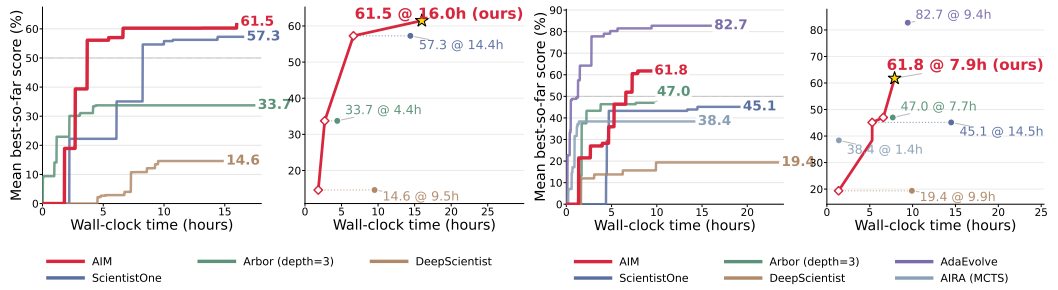
(c) FFT Rust

(d) AES128 Ctr



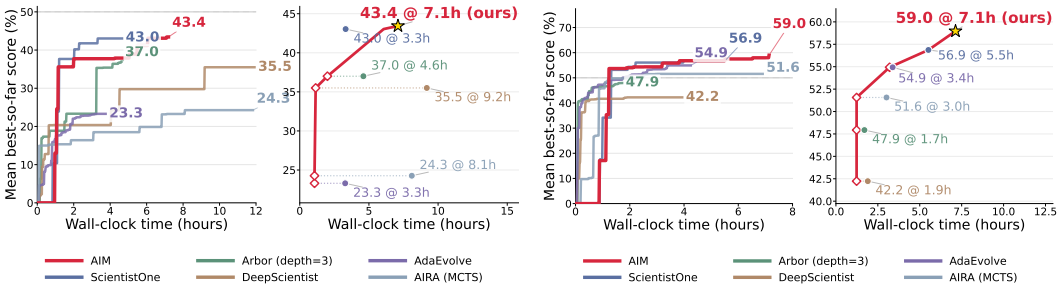
(e) Z-order Range Scan

Figure 6: Time Efficiency Plots Across AutoLab System Optimization Tasks.



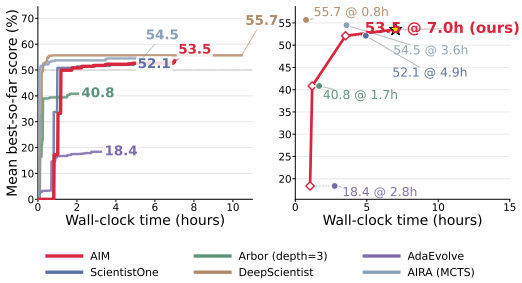
(a) Moving Mnist World Model

(b) Data Select Ifeval



(c) Huffman Canonical Decode

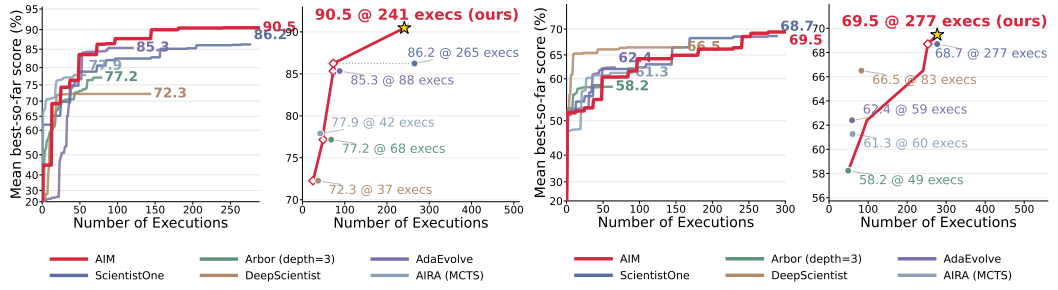
(d) NTT Butterfly



(e) ICP Correspondence Step

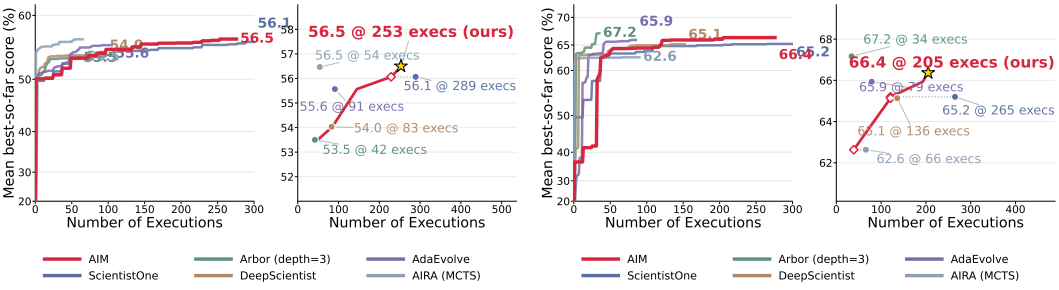
Figure 7: Time Efficiency Plots Across AutoLab Model Development & CUDA Tasks.

In addition to the wall-clock time-to-score plots, we also provide the number of executions-to-score plots in Figure 8 and Figure 9. Note that we allowed five parallel workers for the baselines that enable parallel executions, and each task is budgeted to a maximum of 6 hours for System Optimization; 24 hours for Model Development; and 12 hours for CUDA tasks. Overall, our approach tends to require more executions to reach its best score. We conjecture this is due to AIM's tendency to first explore various directions in the first iterations.



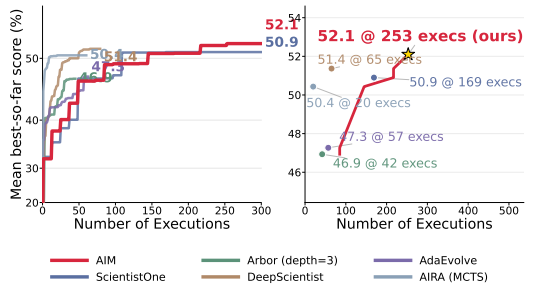
(a) Flash Attention

(b) Radix Sort



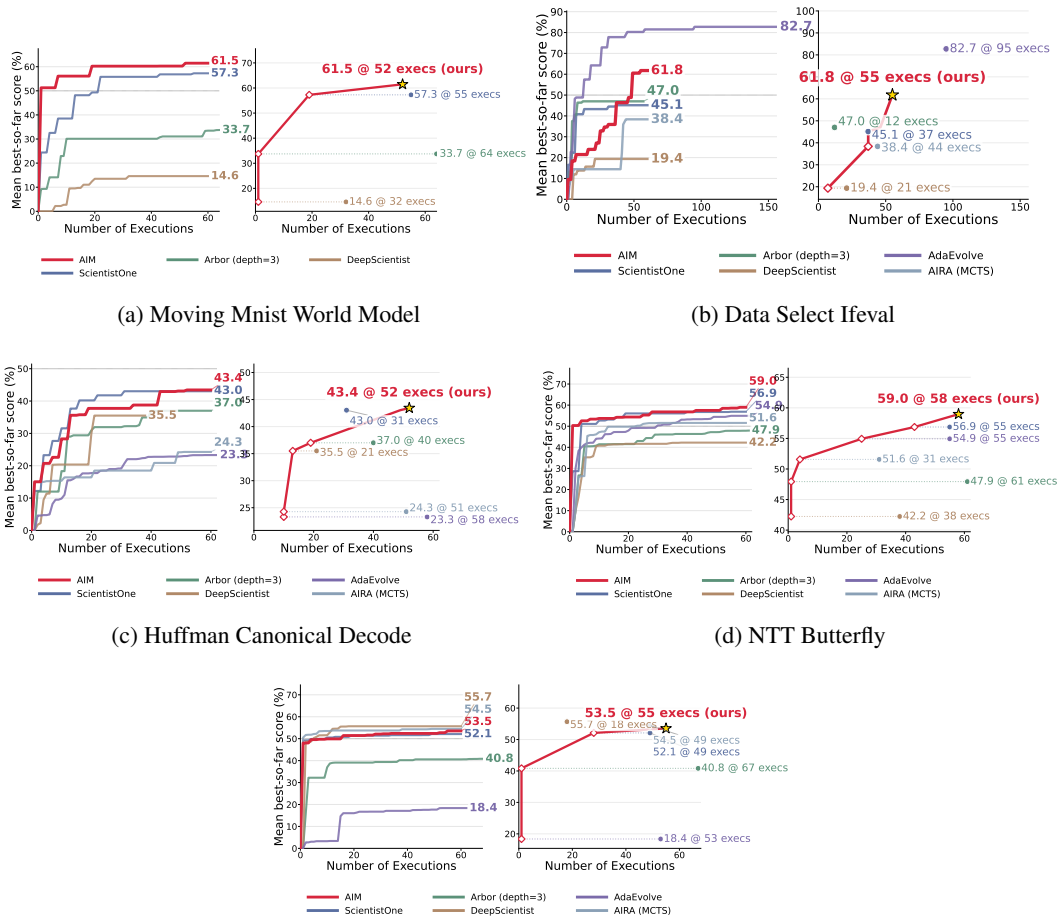
(c) FFT Rust

(d) AES128 Ctr



(e) Z-order Range Scan

Figure 8: Execution-to-Score Plots Across AutoLab Tasks.



(a) Moving Mnist World Model

(b) Data Select Ifeval

(c) Huffman Canonical Decode

(d) NTT Butterfly

(e) ICP Correspondence Step

Figure 9: Execution-to-Score Plots Across AutoLab Model Development & CUDA Tasks.

D.4 Token Cost Analysis

Beyond wall-clock time and execution counts, we report token consumption (total prompt and generation tokens) across all LLM queries. Table 5 reports the token costs, measured by the number of LLM calls, input & output tokens, along with the best score reached by each method. While AIM generally requires a larger token budget, a great portion of it is from the large volume of context utilized in the method. Also, considering the gains in scores, this demonstrates a trade-off between token cost and final performance.

In Figure 10, we further demonstrate the score trend with respect to the token cost, comparing AIM with the strongest baseline, ScientistOne. In 9 out of 10 tasks, AIM takes less tokens to reach the best score of ScientistOne, reducing the token cost up to 3.4× on the Data Select IFEval task.

Table 5: LLM usage and cost per run on all ten AutoLab tasks (mean ± std over three runs). Output tokens include thinking token.

Task	Metric	**AIM** (ours)	ScientistOne	Arbor	AIRA-MCTS	AIRA-EVO
Flash Attention	# LLM calls	1740 ± 86	1550 ± 40	432 ± 73	121 ± 19	83 ± 13
	# Input tokens ($\times 10^6$)	90.1 ± 6.0	70.3 ± 8.8	14.4 ± 1.8	0.9 ± 0.2	0.5 ± 0.1
	# Output tokens ($\times 10^6$)	2.25 ± 0.12	1.77 ± 0.15	0.29 ± 0.00	3.17 ± 0.06	2.20 ± 0.09
	Best score (%)	$\mathbf{90.5 \pm 0.8}$	86.2 ± 3.1	77.2 ± 3.8	77.9 ± 0.2	76.3 ± 1.0
Radix Sort	# LLM calls	1899 ± 14	1558 ± 159	286 ± 64	177 ± 5	178 ± 0
	# Input tokens ($\times 10^6$)	121.1 ± 6.6	79.2 ± 16.1	7.3 ± 3.3	0.7 ± 0.0	0.6 ± 0.0
	# Output tokens ($\times 10^6$)	1.34 ± 0.05	1.13 ± 0.12	0.16 ± 0.05	2.68 ± 0.01	2.23 ± 0.05
	Best score (%)	$\mathbf{69.5 \pm 2.3}$	68.7 ± 1.6	58.2 ± 4.2	61.3 ± 5.8	66.7 ± 2.5
AES128 Ctr	# LLM calls	1844 ± 70	1762 ± 95	386 ± 46	165 ± 33	134 ± 0
	# Input tokens ($\times 10^6$)	142.2 ± 8.2	123.5 ± 9.6	12.3 ± 1.8	1.0 ± 0.2	0.8 ± 0.0
	# Output tokens ($\times 10^6$)	2.08 ± 0.09	1.78 ± 0.25	0.23 ± 0.03	2.91 ± 0.16	2.39 ± 0.07
	Best score (%)	66.4 ± 0.4	65.3 ± 0.2	$\mathbf{67.2 \pm 0.1}$	62.6 ± 0.3	62.8 ± 0.5
FFT Rust	# LLM calls	1639 ± 320	1642 ± 121	365 ± 50	119 ± 32	90 ± 0
	# Input tokens ($\times 10^6$)	70.6 ± 29.7	77.0 ± 18.8	8.2 ± 1.8	0.7 ± 0.2	0.5 ± 0.0
	# Output tokens ($\times 10^6$)	1.64 ± 0.02	1.55 ± 0.18	0.22 ± 0.04	2.94 ± 0.04	2.26 ± 0.27
	Best score (%)	$\mathbf{56.5 \pm 0.5}$	55.9 ± 0.4	53.5 ± 1.1	56.4 ± 0.1	56.2 ± 0.4
Z-order Range Scan	# LLM calls	1696 ± 90	1689 ± 109	275 ± 99	133 ± 17	97 ± 25
	# Input tokens ($\times 10^6$)	103.1 ± 9.1	108.2 ± 14.5	7.4 ± 1.9	1.2 ± 0.1	0.8 ± 0.2
	# Output tokens ($\times 10^6$)	1.90 ± 0.03	1.71 ± 0.12	0.18 ± 0.04	2.67 ± 0.11	2.15 ± 0.18
	Best score (%)	$\mathbf{52.1 \pm 1.1}$	50.9 ± 1.2	46.9 ± 2.1	50.4 ± 1.3	51.1 ± 0.8
MM World Model	# LLM calls	4335 ± 175	3057 ± 1880	678 ± 366	–	–
	# Input tokens ($\times 10^6$)	370.3 ± 42.3	281.7 ± 195.3	18.9 ± 10.0	–	–
	# Output tokens ($\times 10^6$)	1.00 ± 0.04	0.60 ± 0.36	0.25 ± 0.13	–	–
	Best score (%)	$\mathbf{61.5 \pm 0.9}$	57.3 ± 6.3	33.7 ± 11.9	–	–
Data Select IFEval	# LLM calls	6222 ± 389	12784 ± 57	447 ± 321	120 ± 0	–
	# Input tokens ($\times 10^6$)	631.6 ± 126.2	1524.0 ± 18.8	9.3 ± 7.9	0.4 ± 0.1	–
	# Output tokens ($\times 10^6$)	1.38 ± 0.20	2.74 ± 0.06	0.17 ± 0.10	0.81 ± 0.51	–
	Best score (%)	$\mathbf{61.8 \pm 3.2}$	45.1 ± 1.1	47.0 ± 16.1	38.4 ± 5.9	–
Huffman Decode	# LLM calls	4102 ± 261	3906 ± 88	711 ± 196	119 ± 2	–
	# Input tokens ($\times 10^6$)	360.3 ± 30.1	339.8 ± 9.9	28.0 ± 9.1	0.7 ± 0.0	–
	# Output tokens ($\times 10^6$)	2.83 ± 0.24	2.60 ± 0.16	0.61 ± 0.12	4.04 ± 0.69	–
	Best score (%)	$\mathbf{43.4 \pm 1.3}$	43.0 ± 2.6	37.0 ± 2.3	24.3 ± 10.8	–
NTT Butterfly	# LLM calls	4033 ± 141	3905 ± 141	361 ± 34	120 ± 0	–
	# Input tokens ($\times 10^6$)	386.0 ± 17.1	390.5 ± 6.8	10.6 ± 1.8	0.8 ± 0.0	–
	# Output tokens ($\times 10^6$)	2.87 ± 0.25	2.76 ± 0.29	0.27 ± 0.08	2.86 ± 0.56	–
	Best score (%)	$\mathbf{59.0 \pm 1.3}$	56.9 ± 1.3	47.9 ± 5.0	51.6 ± 6.8	–
ICP Corr. Step	# LLM calls	4298 ± 209	3931 ± 161	423 ± 71	120 ± 0	–
	# Input tokens ($\times 10^6$)	371.8 ± 16.9	351.2 ± 7.3	11.5 ± 1.1	0.9 ± 0.0	–
	# Output tokens ($\times 10^6$)	2.51 ± 0.22	2.15 ± 0.18	0.24 ± 0.02	2.37 ± 0.24	–
	Best score (%)	53.5 ± 0.5	52.1 ± 1.0	40.8 ± 10.2	$\mathbf{54.5 \pm 0.9}$	–

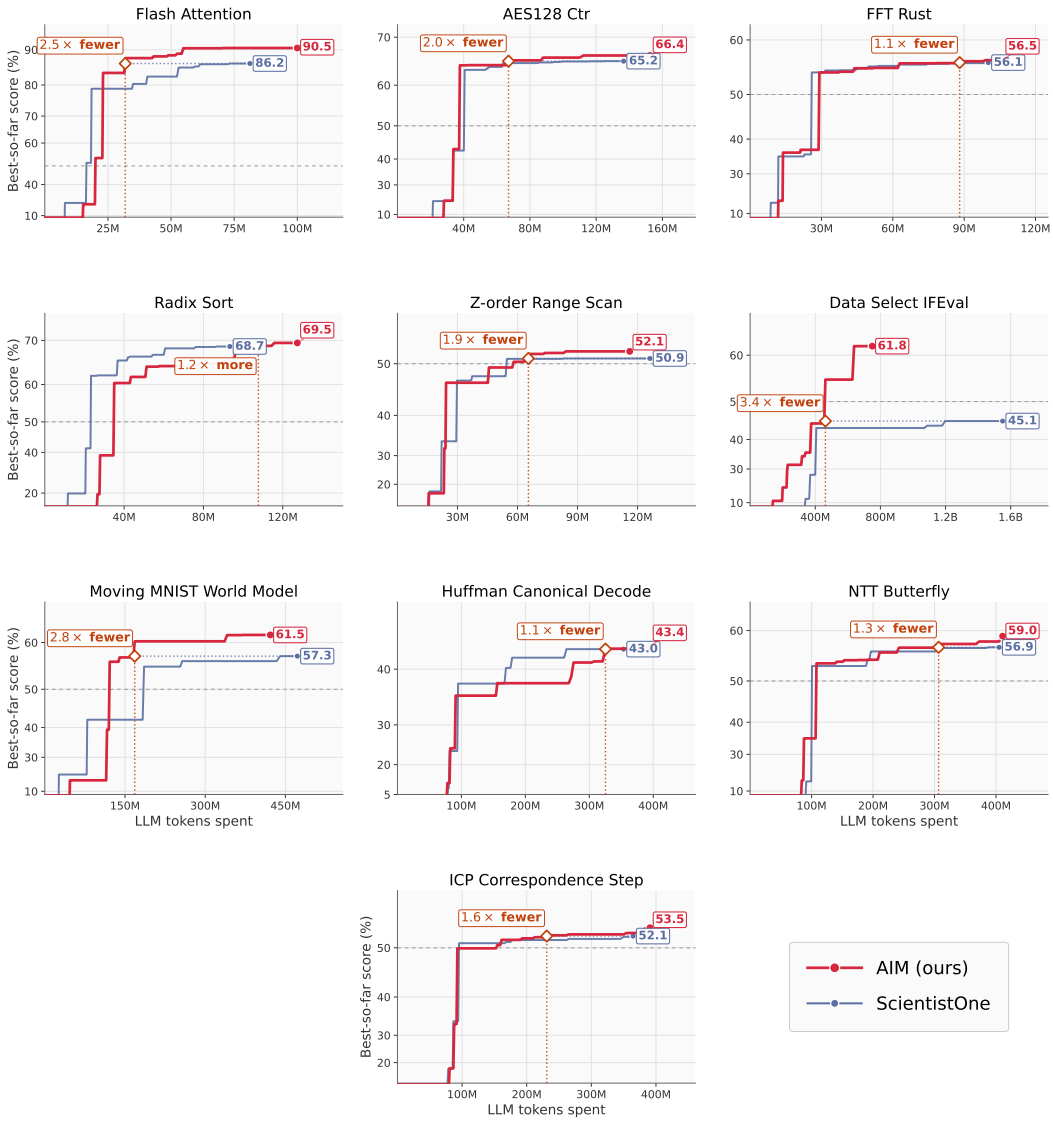


Figure 10: Token Cost Comparison between AIM and ScientistOne.

E FURTHER ANALYSES

E.1 MORE QUALITATIVE EXAMPLES

Example 1 – Flash Attention, iteration 1: cold start, nothing evaluated yet

(1) Clusters and rank estimates pool = 10 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[1]	Memory, Tiling & Data Layout	4	0	–
2	[2]	Advanced SIMD & AVX-512	3	0	–
3	[0]	Exponentiation & Math Optimization	3	0	–

"The primary bottleneck is memory bandwidth due to the $O(n^2)$ intermediate matrix; Tiling (Cluster 1) is mandatory [...]. SIMD (Cluster 2) handles the inner-loop dot products and is the next priority, while scalar math optimizations (Cluster 0) provide smaller marginal gains."

(2) Actions and idea selection

b0 · cluster [1] EXPLORE · idea EXPLOIT → **Asymmetric Register-Only Tiling** `rank 1/4`
"We pick one of the tied top-ranked ideas: pinning Q and the accumulators in AVX registers while streaming K and V directly targets the fundamental memory-bandwidth bottleneck for d=64."

b3 · cluster [2] EXPLORE · idea EXPLORE → **AVX-512 Micro-Kernel Dispatch** `rank 1/3`
"Although the prior step assigned an explore action, the strict Iter-1 constraint requires exploiting the highest-ranked predictions since no empirical scores exist yet."

Outcome: **b0** 0.764 (best) · **b1** 0.720 · **b2** 0.753 · **b3** 0.741 · **b4** 0.730

Example 2 – Flash Attention, iteration 2: scores and lineage drive the choice

(1) Clusters and rank estimates pool = 22 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[2]	Memory Layout & Cache Prefetching	6	1	**0.764**
1	[3]	AVX-512 & Wide Micro-Kernels	6	2	**0.764**
2	[0]	Fast Math & Exp Approximations	5	2	0.753
2	[4]	Register Unrolling & ILP	5	1	0.720
3	[1]	Dynamic Sparsity & Early Exit	4	0	–

"Clusters 2 and 3 share the highest evaluation scores (0.7639) [...]. Cluster 1 is unproven and ranked last, as dynamic sparsity usually introduces branching that undermines dense CPU SIMD performance."

(2) Actions and idea selection

b0 · cluster [3] EXPLOIT · idea EXPLOIT → **Fully Unrolled AVX-512 Dot Products** `rank 2/4`
"We select this idea despite it being rank 2/4. Its parent achieved the highest overall score by introducing block tiling; this child fully unrolls the D=64 dot products into 4× AVX-512 FMAs, applying a proven 'simd' lesson from a sibling."

b3 · cluster [1] EXPLORE · idea EXPLORE → **Dynamic-Threshold Sparsity** `rank 3/4`
"For an explore action we avoid the most promising member (rank 1/4) and pick a lower-confidence idea, validating the core premise of data-dependent sparsity before investing in exact underflow-bound checks."

Outcome: **b0** 0.846 · **b1** 0.850 · **b2** 0.889 (new best) · **b3** 0.450

Example 3 – NTT Butterfly, iteration 3: exploit two proven clusters, probe one untouched

(1) Clusters and rank estimates pool = 31 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[0]	Global Topology & Matrix Restructuring	10	4	**0.557**
2	[1]	Field Arithmetic & PTX	11	2	0.547
3	[3]	Inner-Block Memory & Warp Shuffles	6	1	0.538
4	[2]	Twiddle Factor Management	4	0	–

"Cluster 0 is ranked first because the distilled lessons strongly emphasize 2D matrix factorization and Bailey's 6-step as the critical bottleneck [...]. Cluster 2 has no evaluated signal yet and is generally a secondary optimization."

(2) Actions and idea selection

b1 · cluster [1] EXPLOIT · idea EXPLOIT → **PTX Arithmetic Fusion** `rank 1/9`
"A rank-1 idea that combines the best arithmetic optimization (PTX lazy reduction) with the highest-scoring memory topology from the entire run (coalesced 2D transpose, 0.557)."

b2 · cluster [2] EXPLORE · idea EXPLORE → **Twiddle Cache in Constant/Texture Memory** `rank 3/4`
"This idea has the highest within-cluster rank number (3/4, lower confidence from the ranker). Testing constant/texture caching provides orthogonal information gain compared to compute-based regeneration or workspace precomputation."

Outcome: **b0** 0.557 · **b1** 0.559 (new best) · **b2** 0.460

Example 4 – Data Select IFEval, iteration 3: double down on the leading cluster

(1) Clusters and rank estimates pool = 33 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[3]	Metadata Stratification & Representations	7	1	**0.377**
2	[2]	Deterministic Heuristics & Lexical Diversity	9	3	0.285
3	[0]	Base-Model IFD & Curriculums	8	2	0.193
4	[1]	Generative Probing & LLM-as-a-Judge	9	2	0.137

"Cluster 3 achieves the highest evaluated score (0.3770) and aligns with strong lessons advocating for metadata distribution and length proxies. [...] Clusters 0 and 1 have much weaker top scores, with lessons explicitly warning against the generative probing approaches found in Cluster 1."

(2) Actions and idea selection

b0 · cluster [3] EXPLOIT · idea EXPLOIT → **Source-Balanced IO-Length Stratification** `rank 1/6`
"A direct refinement of the best-performing idea so far: it iterates on the successful source-balancing strategy by incorporating output length to filter out terse responses."

b1 · cluster [3] EXPLOIT · idea EXPLORE → **Unsupervised TF-IDF + KMeans Stratification** `rank 5/6`
"To execute an explore action within this top-performing cluster, we select a high-rank idea (5/6) that introduces a completely different mechanism: data-driven semantic boundaries instead of native 'source' metadata."

Outcome: **b0** 0.119 · **b1** 0.377 (ties best) · **b2** 0.230

Example 5 – Radix Sort, iteration 3: tie-breaking by lineage

(1) Clusters and rank estimates pool = 37 ideas

Rank	Cl.	Label	#ideas	#evaluated	best score
1	[0]	LSD Pass & Fused Histograms	13	7	**0.644**
1	[2]	Alternative Paradigms & Adaptive Sorts	10	2	**0.644**
2	[1]	Memory Hierarchy & Write-Combining	10	3	0.568
3	[3]	Scatter Buffering & Pipelining	4	1	0.544

"Clusters 0 and 2 are tied for Rank 1 as they share the highest best-evaluated score (0.6443) [...]. Cluster 3 ranks last due to its lowest peak score, although with only 1 evaluated idea it retains some exploration value."

(2) Actions and idea selection

b0 · cluster [0] EXPLOIT · idea EXPLOIT → **Fused 11-Bit Radix with Loop-Unrolled Prefetching** `rank 1/6`

"There is a tie at rank 1/6 between two prefetch-based refinements. I chose this one because its parent achieved the highest overall score in the run (0.6443), making it the most promising trajectory to aggressively exploit."

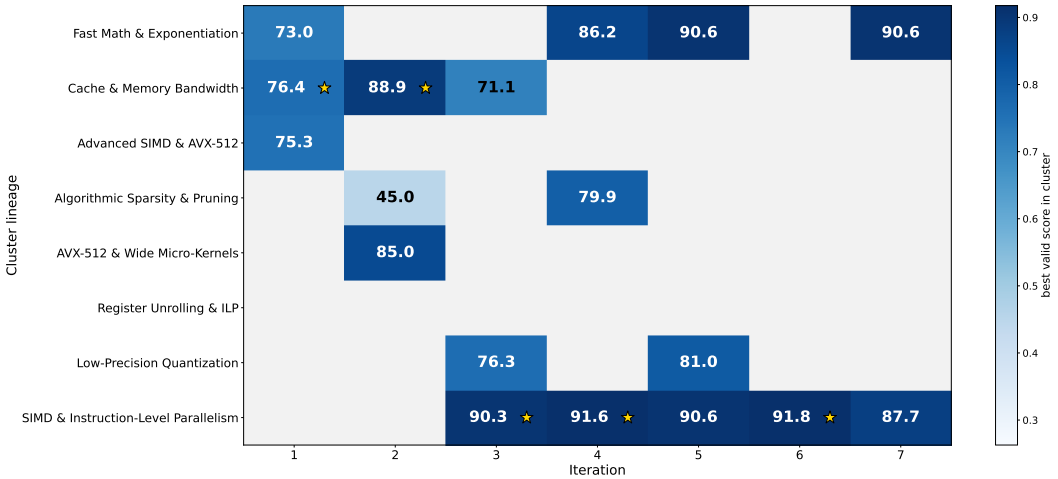
b1 · cluster [0] EXPLOIT · idea EXPLORE → **Restricted Pointers & Unrolled 12-12-8 Scatter** `rank 5/6`

"While evaluated members have focused on prefetching and fused histograms, this idea tests whether compiler-level optimizations like `restrict` pointers and manual unrolling can significantly accelerate the 12-12-8 radix scatter."

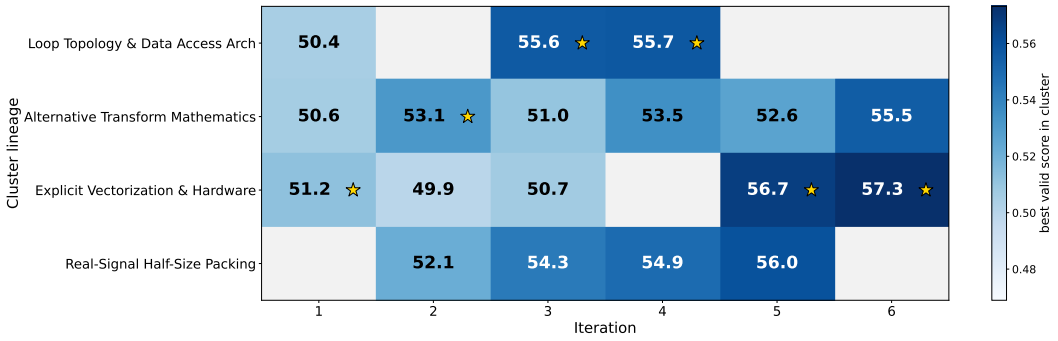
Outcome: b0 0.661 (new best) · **b1** 0.536 · **b2** 0.570 · **b3** 0.546

E.2 Qualitative Mechanisms of the Organize Operator

One strong advantage of idea-driven approaches is its interpretability in the research trajectory; it is easy to follow the search trajectory summarized by the idea traces. As a qualitative analysis, we examine how the Organize operator restructures the discovered idea pool as new candidates and experimental evidence become available. In Figure 11, we provide example heatmaps of the best score from each cluster explored, and show how the exploration evolves throughout iterations.



(a) Flash Attention: Cluster-wise Best Score Progression.



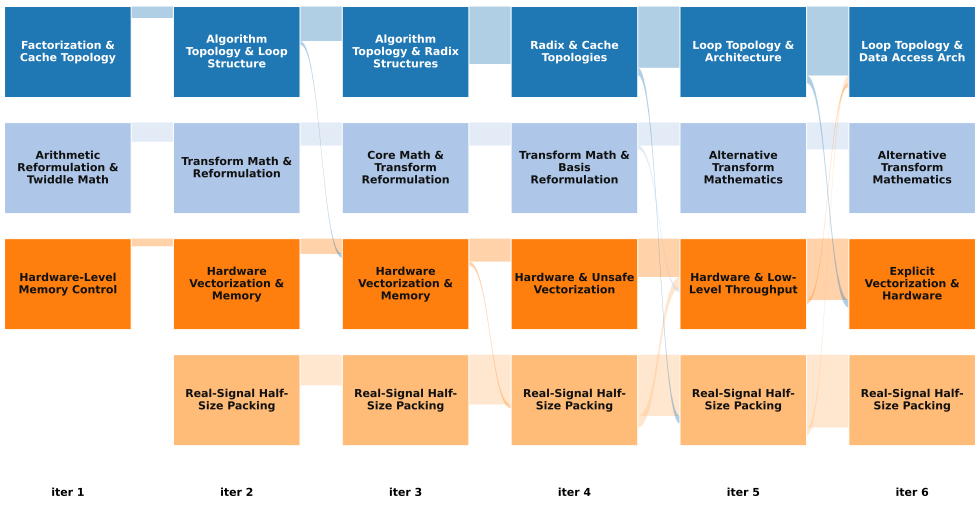
(b) FFT Rust: Cluster-wise Best Score Progression.

Figure 11: **Agentic Surrogate: ORGANIZE**. Heatmap of the best score of each cluster explored in each iteration. The star symbol marks the point when and where the new best scoring solution was discovered.

On one example run on Flash Attention (Figure 11 (a)), the search initially explores several directions, including memory optimization, mathematical approximation, and vectorization. As evidence accumulates, SIMD and instruction-level parallelism emerges as a consistently strong direction, while fast exponentiation remains competitive in later iterations. Overall, 8 different cluster themes were proposed, while 7 of them were explored throughout the iterations. Note that the ideas comprising each cluster theme is not mutually exclusive; the list of ideas are re-clustered every iteration, so the same idea could have been regrouped into a different cluster in the subsequent rounds. On FFT Rust (Figure 11 (b)), on the other hand, the leading direction changes repeatedly among transform reformulation, loop and cache topology, real-signal packing, and explicit vectorization. Compared to the Flash Attention task that spanned 8 semantic clusters, FFT Rust had a narrower breadth of search with 4 different clusters explored. This difference in trend largely depends on the trait of the task in question.



(a) Flash Attention: Alluvial Plot.



(b) FFT Rust: Alluvial Plot.

Figure 12: **Agentic Surrogate: ORGANIZE**. Alluvial plots showing how the ideas flow and how they are re-organized throughout iterations.

The alluvial diagrams in Figure 12 further visualize how clusters persist, split, merge, and change names across, showing the fine-grained flow and structure of ideas evolving across iterations. Despite these structural revisions at every iteration, coherent semantic trajectories remain visible. For example, the broad SIMD direction in Flash Attention develops into more specialized directions involving wide micro-kernels, register unrolling, low-precision computation, and instruction-level parallelism. Similarly, the FFT Rust task progressively refines broad hardware- and topology-oriented clusters into explicit vectorization and data-access strategies. *Thus, the Organize operator preserves semantic continuity while allowing the representation to adapt beyond a fixed list, tree, or generation lineage*.

E.3 Examining the Preciseness of the Estimate Operator

We next examine whether the ordinal promisingness estimates produced by the Estimate operator agree with subsequently observed verifier scores. For each cluster, we measure the Spearman correlation between the cluster-level rank estimates and the actual rank returned by the executions, and average them across iterations.

As seen in Figure 13, the estimates are positively correlated with observed performance across all five tasks, with mean cluster-wise correlations ranging from 0.56 to 0.91 and an overall average of approximately 0.71. The strongest agreement occurs on Z-order Range Scan and Flash Attention, while the remaining tasks retain moderate positive correlations. These results indicate that the agent can extract useful ranking signals from the organized idea map, evaluation history, and implementation lessons without predicting calibrated reward values.

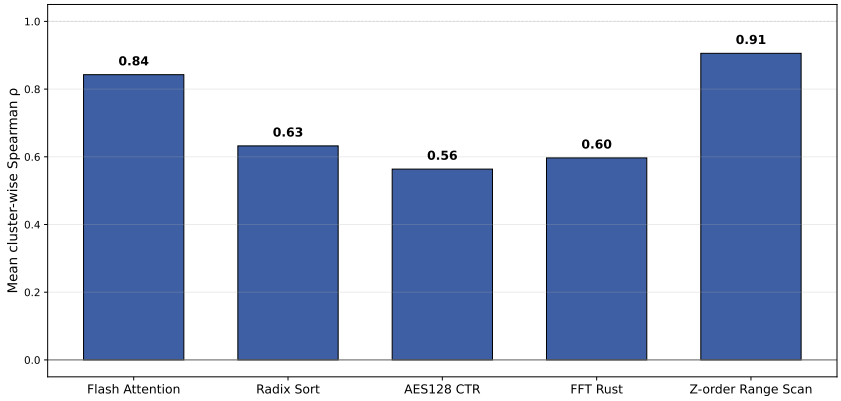
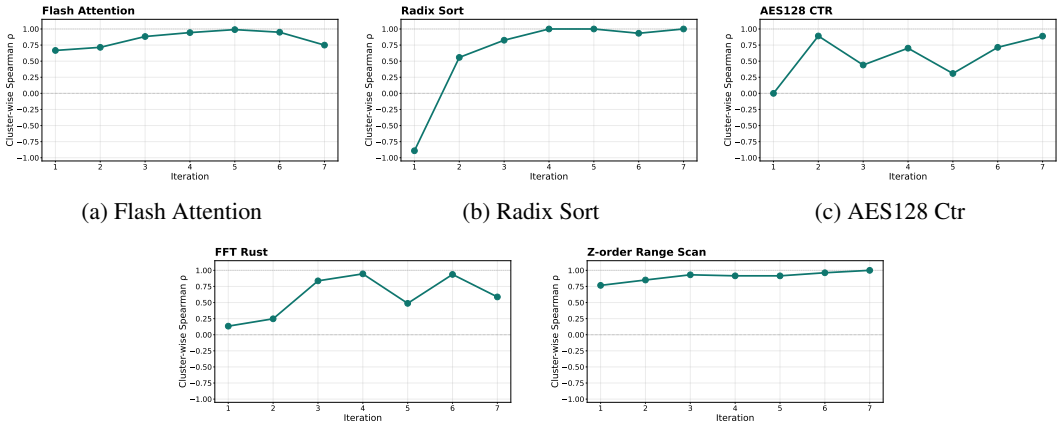


Figure 13: Cluster-level Ordinal Estimate's Average Spearman Correlation.

The iteration-wise results in Figure 14 show how these estimates change as evidence accumulates. Correlation generally improves after the initial iterations: for example, the estimate on Radix Sort recovers from a poor initial ranking to nearly perfect agreement in later iterations, while Flash Attention and Z-order Range Scan maintain strong agreement over much of the search. The trajectories are not strictly monotonic, however, particularly for AES128-CTR and FFT. We conjecture this might be because (1) there are multiple competing clusters that have similar expected performance, and/or (2) there might be implementation noise (i.e., minor implementation details can change the measured performance). Nevertheless, the estimates remain informative enough to ground subsequent exploration and exploitation decisions in observed research progress.

E.4 Dispatch Operator Action Analysis

The Dispatch operator makes idea selection decisions at two levels for each Solver branch: whether to explore or exploit across clusters, and whether to explore or exploit among ideas within the selected cluster. Motivated by recent findings that LLM agents often fail to explore systematically (Krishnamurthy et al., 2024; Pan et al., 2026; Park et al., 2026; Choi et al., 2026), AIM requires the agent to explicitly verbalize these decisions. Unlike approaches that rely on externally specified search heuristics (Yamada et al., 2025; Toledo et al., 2026) or direct LLM-based candidate selec-



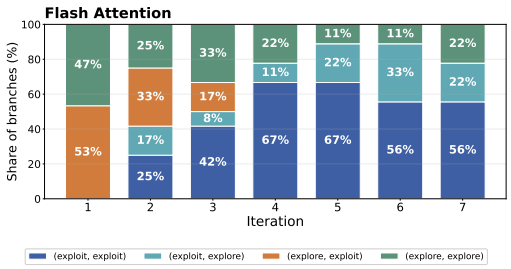
(a) Flash Attention (b) Radix Sort (c) AES128 Ctr

(d) FFT Rust (e) Z-order Range Scan

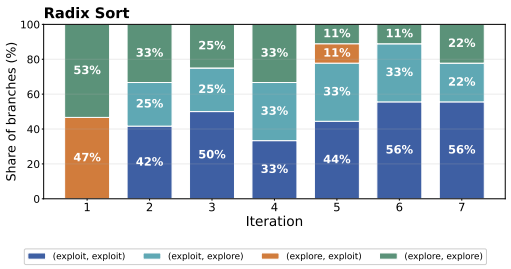
Figure 14: Trend of Spearman Correlation across Iterations.

tion (Weng et al., 2026), AIM conditions its actions on the agent's explicit assessment of the current idea map and promisingness estimates. This makes the exploration–exploitation policy evidence-grounded and interpretable.

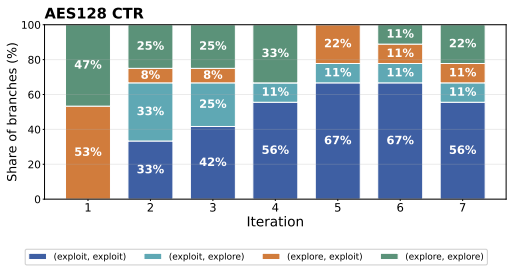
Figure 15 shows the distribution of the resulting two-level actions, represented as (*cluster-level action*, *idea-level action*), across iterations of the five tasks. Early iterations generally allocate more branches to actions involving exploration, whereas exploitation becomes increasingly prevalent in later iterations. This shift is intuitive: when evidence is sparse, the agent broadens coverage across clusters and ideas; as experimental evidence accumulates, it increasingly concentrates resources on directions estimated to be promising. The task-dependent variation in these distributions further indicates that the search policy is adapted to research progress rather than fixed in advance.



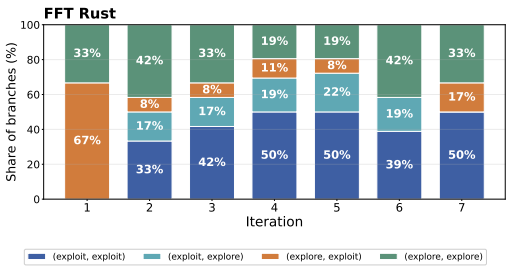
(a) Flash Attention



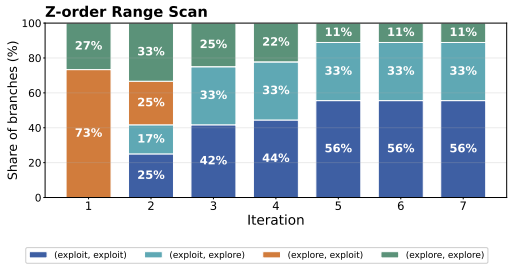
(b) Radix Sort



(c) AES128 Ctr



(d) FFT Rust



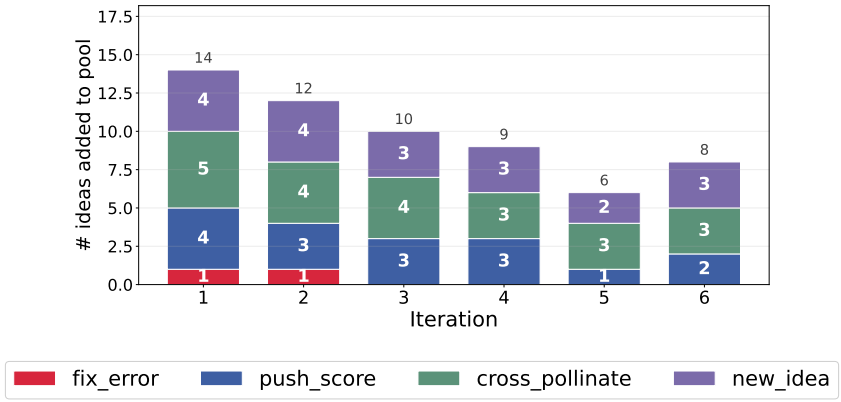
(e) Z-order Range Scan

Figure 15: Explore/Exploit Action Distribution Across Branches.

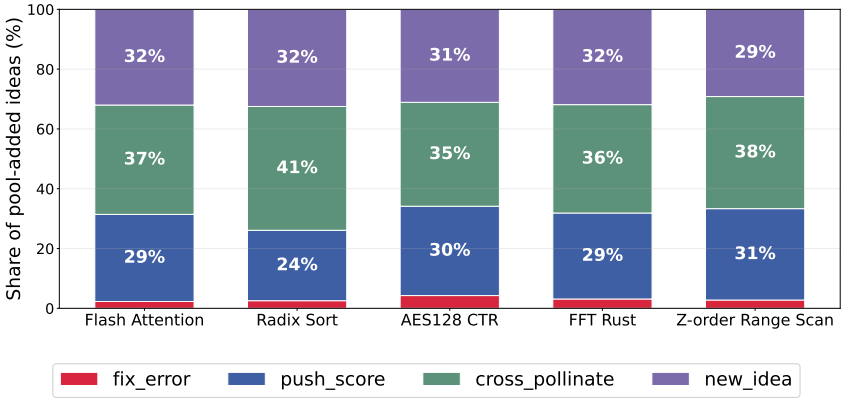
E.5 Mode Distribution of the Expand Operator

We analyze how the Expand operator uses its four generation modes throughout the search. As an example, Figure 16 (a) reports the number of ideas added at each Flash Attention iteration and the aggregate mode distribution across all five tasks. Overall, the three modes excluding 'fix-error' shows similar rates of usage.

As general analysis, we also provide the distribution of modes per task in Figure 16 (b). The cross-task distributions are mostly consistent across tasks. The cross-pollination mode accounts for 35–41% of generated ideas, novel idea generation for 29–32%, and score-guided refinement for 24–31%. Error-guided repair constitutes only a small remaining fraction. These results show that the Expand operator does not collapse onto a single generation strategy; it continually combines the refinement and recombination of existing evidence with the introduction of previously unexplored directions.



(a) Flash Attention Task Mode Distribution



(b) Cross-task Mode Distribution

Figure 16: Expand operator mode distributions.

E.6 Effect of Solution Auditor Idea Reconstruction

In the Solution Auditor component, a solution that was flagged solely with "idea-mismatch" has its idea reconstructed. This reconstruction is intended to reduce the misalignment between the actual solution evaluated, and the corresponding idea that triggered the solution. To verify if this feedback cycle is working properly, we show in Figure 17 the ratio of `idea-mismatch` flags before and after the idea reconstruction is performed. As intended, the ratio of the `idea-mismatch` flag is greatly reduced across the tasks, ensuring that the (idea, solution) pairs are much more aligned.

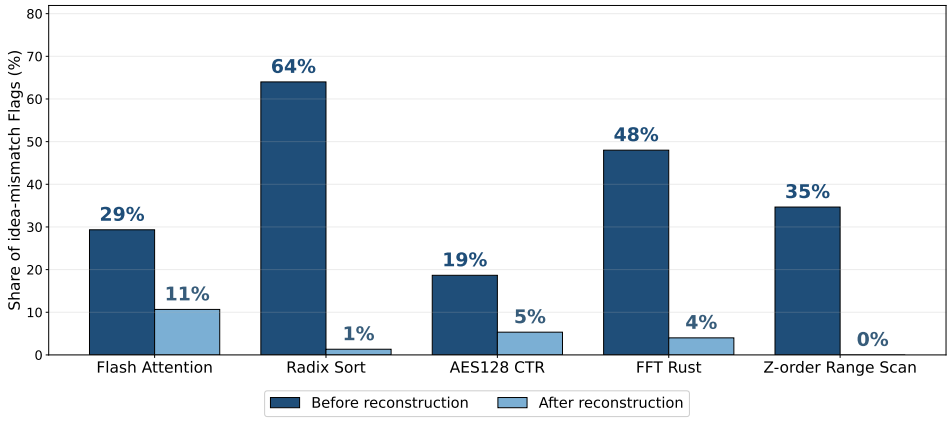


Figure 17: Decrease in `idea-mismatch` flags after Solution Auditor idea reconstruction.

Qualitative Analysis. As direct scrutiny of how idea reconstruction affects the process, we here provide couple of examples of the reconstruction process from the Flash Attention task. For each example, we demonstrate the initially assigned idea, the auditor's verdict, and the reconstructed idea.

EXAMPLE 1: THE CODE RELIED ON THE TECHNIQUE THE IDEA SET OUT TO AVOID

Example 1 — Assigned Idea

```
Title: AVX-512 Flash Attention 64x64 Tiling with Compiler Exp

Short Hypothesis: Expanding the tile size to 64x64 using AVX-512
intrinsics will improve L1 cache utilization and FLOP throughput over
the C1 reference; furthermore, relying on standard exp() rather than
a custom 5-term AVX-512 minimax exponential avoids FMA execution port
contention in the heavily unrolled register-blocked loop, improving
net throughput.

Abstract: The naive C0 baseline evaluates scaled dot-product
attention in 0.75s by materializing a 128 MB FP64 score matrix. The
C1 reference reduces this to ~0.10s using Flash Attention tiling
(Br=Bc=32), online softmax, and AVX2 intrinsics. We propose further
minimizing this runtime toward the theoretical ~0.05s floor by
deploying AVX-512 intrinsics and expanding the tile size to 64x64,
better utilizing L1 cache capacity and doubling SIMD lane width.

However, migrating to an aggressively unrolled 64x64 AVX-512
micro-kernel places immense pressure on CPU FMA execution ports.
Although the literature typically suggests replacing scalar exp()
with a vectorised 5-term minimax approximation to speed up the
softmax rescale, we hypothesize that executing this polynomial in
AVX-512 competes directly with the QK^T and SV dot-product FMAs.

By isolating the exp() computation from the dense FMA pipeline --
specifically by reverting to standard compiler-managed exp() or
strategically placing it outside the innermost unrolled FMA block --
we can prevent execution port saturation. We expect this combined
strategy (AVX-512, 64x64 tiling, and contention-aware exp()
selection) to push performance substantially closer to the hardware's
practical 0.05s floor.

Experiments:
```

```
1. Hypothesis: Expanding the tile size from 32x32 to 64x64 and
migrating from AVX2 to AVX-512 intrinsics will improve throughput
over the C1 reference implementation.
   Step 1: Port the existing C1 reference AVX2 kernel to use AVX-512
intrinsics (_mm512_fmadd_ps, etc.), initially keeping Br=Bc=32.
   Step 2: Expand the loop bounds and register blocking to support a
Br=64, Bc=64 tile size.
   Step 3: Measure execution time and verify exact numerical
correctness against the C0/C1 outputs. If register spilling degrades
performance, evaluate asymmetric tiles (e.g., 64x32).
2. Hypothesis: Using standard exp() for softmax normalisation yields
better overall throughput than a custom 5-term AVX-512 minimax
exponential by avoiding FMA port contention in the unrolled inner
loop.
   Step 1: Implement a 5-term AVX-512 minimax polynomial
approximation for exp() (e.g., based on Schraudolph 1999) inside the
64x64 tiled micro-kernel.
   Step 2: Create an ablative variant that replaces the custom
minimax exp() with calls to the standard math library exp().
   Step 3: Compare both variants' wall-clock runtimes. Profiling
should confirm whether the FMA-heavy custom exp() reduces overall
instruction throughput due to port contention compared to the
standard exp().
```

Example 1 — Auditor Verdict

```
flags: ["idea_mismatch"]   confidence: 1.0

The idea explicitly proposes avoiding custom minimax exponential
approximations in favor of the standard compiler exp() to prevent FMA
execution port saturation in the heavily unrolled loop. However, the
solution code contradicts this core hypothesis entirely by
implementing and utilizing a custom AVX-512 minimax polynomial
approximation (fast_exp512) for the softmax calculation. Because the
implementation fundamentally relies on the technique the idea was
explicitly aiming to ablate and avoid, there is a clear idea-solution
mismatch.
```

Example 1 — Reconstructed Idea

```
Title: AVX-512 Flash Attention 64x64 Tiling with Minimax Exp

Short Hypothesis: Applying AVX-512 intrinsics with 64x64 tiling and a
custom minimax polynomial for exp() will maximize L1 cache usage and
throughput.

Abstract: The naive baseline evaluates scaled dot-product attention
slowly by materializing a full score matrix. We optimize this by
applying Flash Attention with a 64x64 tile size and AVX-512
intrinsics to fully utilize L1 cache capacity and double the SIMD
lane width. To avoid sequential bottlenecks, we compute the softmax
normalization using a custom, fully vectorised AVX-512 minimax
polynomial approximation, isolating it from the innermost dot-product
loops.

Experiments:
1. Hypothesis: A 64x64 AVX-512 Flash Attention kernel with a custom
minimax exp() will yield higher performance than the C1 reference.
   Step 1: Implement the Flash Attention inner loops using AVX-512
intrinsics with a 64x64 tile size.
```

```
   Step 2: Implement a custom AVX-512 minimax exponential
approximation for the softmax normalization.
   Step 3: Benchmark execution time against the baseline to verify
throughput improvements.
```

Without reconstruction, the pool would have recorded "compiler `exp` beats minimax `exp`" as the best-supported claim in the run, the opposite of what was tested, and the allocator would have exploited that claim in later iterations.

EXAMPLE 2: THE SECONDARY COMPONENT SURVIVED, THE CENTRAL CLAIM DID NOT

Example 2 — Assigned Idea

```
Title: AVX-512 Sequence-Dim Vectorization with Minimax Exp

Short Hypothesis: Vectorizing over the sequence dimension (Br=16)
rather than the feature dimension will eliminate costly horizontal
sums while preserving the parent's highly accurate minimax exp
speedup.

Abstract: Standard scaled dot-product attention computes an O(n^2)
score matrix. For n=4096 and d=64, this 128 MB FP64 allocation
thrashes the L1/L2 caches, resulting in severe latency (Baseline C0:
~0.75 s). Flash Attention algorithms bypass this by tiling the
computation and maintaining running softmax statistics in fast cache
(Reference C1: ~0.10 s). However, C1's reliance on AVX2
feature-dimension vectorization necessitates expensive horizontal
sums, and its scalar exp() calls in the normalizer act as a serial
bottleneck.

We propose a redesigned micro-kernel that pivots vectorization to the
sequence dimension. By setting the row block size to Br=16, we
perfectly map the computation to the 16 lanes of an AVX-512 register.
This enables computing dot products for 16 distinct queries against a
broadcasted key simultaneously using purely vertical _mm512_fmadd_ps
operations -- completely eliminating horizontal reductions. We
integrate this structural change with a vectorized 5-7 term minimax
exp() approximation with ldexp range reduction to process the
online-softmax rescale entirely within the AVX-512 pipeline.

By removing horizontal dependencies and replacing scalar math with
wide SIMD operations, we expect to bridge the gap between the C1
reference (~0.10 s) and the theoretical order-of-magnitude floor
(~0.05 s), all while adhering strictly to single-threaded FP32
numerical correctness requirements.

Experiments:
1. Hypothesis: An AVX-512 Flash Attention micro-kernel vectorized
over the sequence dimension (Br=16) with a 5-7 term minimax exp
approximation will outperform the C1 Reference by removing horizontal
sums and scalar bottlenecks.
   Step 1: Write an AVX-512 micro-kernel in solve.c using a Br=16
tile size, reformulating QK^T and PV multiplication so that 16
queries are processed vertically in parallel without horizontal add
instructions.
   Step 2: Implement a vectorized 5-term minimax exp() approximation
using AVX-512 intrinsics for the online softmax rescaling block.
   Step 3: Benchmark wall-clock runtime against Baseline C0 and
Reference C1, and validate that the output strictly matches the exact
FP32 result.
```

```
2. Hypothesis: Vectorizing the micro-kernel over the sequence
dimension (Br=16) yields a performance improvement independent of the
minimax exp approximation due to the elimination of horizontal
reductions.
   Step 1: Create an ablated version of the AVX-512 sequence-dim
kernel that temporarily disables the vectorized minimax exp, falling
back to sequential scalar exp() calls for the softmax step.
   Step 2: Measure the performance of this ablated kernel against the
Reference C1 to isolate the latency saved purely by eliminating
horizontal SIMD sums.
   Step 3: Compare this ablated timing to the fully optimized kernel
from Experiment 1 to quantify the fraction of speedup attributable to
the vectorized minimax exp().
```

Example 2 — Auditor Verdict

```
flags: ["idea_mismatch"]   confidence: 1.0

The idea proposes vectorizing over the query sequence dimension
(Br=16) to perfectly map independent queries to lanes and completely
eliminate horizontal reductions during the softmax step. However, the
code vectorizes QK^T over the key sequence dimension (processing 1
query against 64 keys at a time) and explicitly uses horizontal
reductions (_mm512_reduce_max_ps and _mm512_reduce_add_ps) across the
SIMD lanes to compute the softmax statistics. While the code does
successfully implement the proposed AVX-512 minimax exp()
approximation, its overall vectorization strategy contradicts the
core architectural claim of the idea.
```

Example 2 — Reconstructed Idea

```
Title: AVX-512 Key-Sequence and Feature Vectorization with Minimax
Exp

Short Hypothesis: Vectorizing QK^T over keys and PV over features
with an AVX-512 minimax exp() will accelerate attention by maximizing
FMA throughput.

Abstract: Standard scaled dot-product attention computes an O(n^2)
score matrix, causing cache thrashing for large sequences. We propose
an AVX-512 micro-kernel using 64x64 blocks that vectorizes the QK^T
multiplication over the key sequence dimension, computing dot
products for 4 queries against 64 keys concurrently via broadcasted
query features. The softmax normalizer applies a vectorized 5-term
minimax exp() approximation and uses horizontal SIMD reductions for
running max and sum statistics. Finally, the PV step vectorizes over
the feature dimension, broadcasting attention weights to compute the
final output.

Experiments:
1. Hypothesis: An AVX-512 Flash Attention kernel vectorized over key
sequence and feature dimensions with a minimax exp() will outperform
baseline implementations.
   Step 1: Write an AVX-512 kernel with Br=64 and Bc=64, vectorizing
QK^T over keys and PV over features.
   Step 2: Implement a vectorized 5-term minimax exp() approximation
using AVX-512 intrinsics and horizontal reductions for the softmax
step.
   Step 3: Benchmark wall-clock runtime against the baseline and
validate FP32 numerical correctness.
```

This is the subtle case: a user would conclude that eliminating horizontal sums is key, when the kernel that earned that reward performs horizontal sums. Reconstruction keeps the credit on the parts that were actually executed.

Reconstruction is a correction of *attribution*. Because every downstream meta stage consumes scores through idea attributions, uncorrected mismatches would teach the surrogate and the allocator the wrong lessons about which mechanisms work, and would do so most strongly for the highest-scoring branches.

F Theoretical Analyses

F.1 Assumptions

Assumption 1 (**Semantic Decomposition**). *Let each executable solution $z \in \mathcal{Z}$ be associated with an underlying research direction $\pi(z) \in \mathcal{X}$, where $\mathcal{X}$ is the finite set of ideas. Then, for each idea $x \in \mathcal{X}$, its corresponding semantic solution region is $\mathcal{Z}_x = \{z \in \mathcal{Z} : \pi(z) = x\}$.*

Assumption 2 (**Faithful Realization**). *For every selected idea $x \in \mathcal{X}$, the implementation process can produce an executable solution $z = g(x)$ that faithfully realizes the selected idea: $\pi(g(x)) = x$.*

Assumption 3 (**Competitive Direction Exchangeability**). *Before the search evaluates a direction, the identities of the G_ε ε-optimal directions are uniformly distributed among the K admissible directions. Conditional on having evaluated only non-ε-optimal directions, the optimal directions remain exchangeable among the unexplored directions.*

Assumption 1 and Definition 1 provide a useful way to view the difference between the two search paradigms. For instance, consider a search graph whose nodes are executable solutions. A solution-driven search trajectory may move through $z_1 \to z_2 \to z_3 \to z_4$ ($z_i \in \mathcal{Z}$), while the corresponding semantic directions are $x_1 \to x_1 \to x_1 \to x_2$ ($x_i \in \mathcal{X}$). In such cases, although four distinct solutions have been evaluated, this trajectory covers only two semantic directions. More generally, the mapping $\pi : \mathcal{Z} \to \mathcal{X}$ groups solutions according to their underlying research direction. Formally, π induces the equivalence relation $z \sim z'$ if and only if $\pi(z) = \pi(z')$, thereby contracting solutions that instantiate the same idea into a common semantic class. This yields a quotient view of the solution space, in which solution-level trajectories are represented by the distinct semantic directions they traverse.

In addition, while Assumption 2 may seem a bit idealistic, we argue that the Solution Auditor ensures the evidence that the agent observes is faithful and reliable. Figure 17 in Appendix E.6 supports this view, in that it substantially reduces the idea-mismatch cases after the Solution Auditor's idea reconstruction takes place. Furthermore, our discussion on the practical considerations in Appendix F.8 suggests that a hybrid design of solution-driven and idea-driven approaches might be desired to ensure faithful implementation of ideas.

F.2 A Note on the Finiteness of Idea Space $\mathcal{X}$

In our theoretical analysis, we define the Semantic Coverage of an idea set. Since comparison across methods will make sense only when the search space is finite, we set a proposition on the finiteness of the idea space.

Proposition 3 (**Finiteness of the Idea Space**). *For a fixed research task τ, suppose each admissible research idea is represented as a sequence of tokens from a finite vocabulary Σ, with maximum description length $L < \infty$. Then the corresponding idea space $\mathcal{X}$ is finite.*

Proof. Let Σ denote the finite token vocabulary and let $\Sigma^{\leq L}$ denote the set of all token sequences of length at most L:

$$\Sigma^{\leq L} = \bigcup_{\ell=0}^{L} \Sigma^{\ell}. \tag{10}$$

Since Σ is finite,

$$|\Sigma^{\ell}| = |\Sigma|^{\ell}. \tag{11}$$

Therefore,

$$|\Sigma^{\leq L}| = \sum_{\ell=0}^{L} |\Sigma|^{\ell} < \infty. \tag{12}$$

For a fixed task τ, define the admissible idea space as

$$\mathcal{X} = \left\{ x \in \Sigma^{\leq L} : x \text{ constitutes an admissible research idea for } \tau \right\}. \tag{13}$$

By construction,

$$\mathcal{X} \subseteq \Sigma^{\leq L}. \tag{14}$$

Since every subset of a finite set is finite,

$$|\mathcal{X}| < \infty. \tag{15}$$

Thus, the admissible idea space for task τ is finite. □

F.3 Proof of Propositions

Proof of Proposition 1. Assumption 1 induces an equivalence relation over executable solutions:

$$z \sim z' \quad \Longleftrightarrow \quad \pi(z) = \pi(z'). \tag{16}$$

Let

$$q : \mathcal{Z} \to \mathcal{Z}/\sim \tag{17}$$

denote the corresponding quotient map, where $q(z) = [z]$ is the semantic equivalence class containing z. For any evaluated solution set $\mathcal{E} \subseteq \mathcal{Z}$, its semantic coverage can therefore be written as

$$C(\mathcal{E}) = |q(\mathcal{E})|. \tag{18}$$

Consider first an arbitrary search procedure that evaluates

$$\mathcal{E} = \{z_1, \ldots, z_m\}, \qquad m \leq N. \tag{19}$$

Since q is a function, taking its image cannot increase the cardinality of a finite set. Hence,

$$C = |q(\mathcal{E})| \leq |\mathcal{E}| = m \leq N. \tag{20}$$

Intuitively, quotient contraction may merge multiple evaluated solutions into the same semantic class, but can never create additional semantic classes.

Now consider the idea-driven procedure, which allocates its N experiment executions to N distinct ideas

$$x_1, \ldots, x_N, \qquad x_i \neq x_j \quad \text{for } i \neq j. \tag{21}$$

Let

$$z_i = g(x_i) \tag{22}$$

be the corresponding executable solutions. By Assumption 2,

$$\pi(z_i) = x_i. \tag{23}$$

Therefore, for every $i \neq j$,

$$\pi(z_i) \neq \pi(z_j), \tag{24}$$

and hence

$$z_i \not\sim z_j. \tag{25}$$

Thus, no two of the N solutions are contracted into the same equivalence class, giving

$$C^{\text{idea}} = |q(\{z_1, \ldots, z_N\})| = N. \tag{26}$$

Combining equation 20 and equation 26, we obtain

$$C \leq N = C^{\text{idea}}, \tag{27}$$

which proves the result. □

Proof of Proposition 2. Let a search procedure cover C distinct semantic directions. Under Assumption 3, conditional on not having yet encountered an ε-optimal direction, the competitive directions remain exchangeable among the unexplored directions.

After i distinct non-competitive directions have been explored, there remain $K - i$ unexplored directions, of which G_ε are ε-optimal. Therefore, the conditional probability that the next explored direction is also non-competitive is

$$\frac{K - G_\varepsilon - i}{K - i}. \tag{28}$$

Hence, for $C \leq K - G_\varepsilon$, the probability of failing to encounter any ε-optimal direction after covering C distinct directions is

$$1 - P_\varepsilon(C) = \prod_{i=0}^{C-1} \frac{K - G_\varepsilon - i}{K - i} \tag{29}$$

$$= \frac{\binom{K-G_\varepsilon}{C}}{\binom{K}{C}}. \tag{30}$$

Thus,

$$P_\varepsilon(C) = 1 - \frac{\binom{K-G_\varepsilon}{C}}{\binom{K}{C}}. \tag{31}$$

If $C > K - G_\varepsilon$, at least one competitive direction must necessarily have been covered, and therefore $P_\varepsilon(C) = 1$, which is consistent with the same combinatorial expression under the convention $\binom{K-G_\varepsilon}{C} = 0$.

Now fix a target success probability $1 - \delta$ and define

$$C_\delta(K, G_\varepsilon) = \min \{C : P_\varepsilon(C) \geq 1 - \delta\}. \tag{32}$$

For fixed K and C, each factor

$$\frac{K - G_\varepsilon - i}{K - i} = 1 - \frac{G_\varepsilon}{K - i} \tag{33}$$

is non-increasing in G_ε. Hence, $P_\varepsilon(C)$ is non-decreasing in G_ε, implying

$$G_\varepsilon^{(1)} \leq G_\varepsilon^{(2)} \quad \Longrightarrow \quad C_\delta(K, G_\varepsilon^{(1)}) \geq C_\delta(K, G_\varepsilon^{(2)}). \tag{34}$$

Thus, for a fixed number of admissible directions, sparser competitive directions require broader semantic coverage.

Similarly, for fixed G_ε and C, each factor

$$1 - \frac{G_\varepsilon}{K - i} \tag{35}$$

is non-decreasing in K. Therefore, increasing the number of admissible directions while keeping the number of competitive directions fixed decreases $P_\varepsilon(C)$ and increases the semantic coverage required to attain the same success probability.

Finally, the failure probability satisfies

$$1 - P_\varepsilon(C) = \prod_{i=0}^{C-1} \left(1 - \frac{G_\varepsilon}{K - i}\right) \tag{36}$$

$$\leq \left(1 - \frac{G_\varepsilon}{K}\right)^C \tag{37}$$

$$\leq \exp\left(-\frac{CG_\varepsilon}{K}\right). \tag{38}$$

Consequently, the sufficient condition

$$C \geq \frac{K}{G_\varepsilon} \ln \frac{1}{\delta} \tag{39}$$

guarantees $P_\varepsilon(C) \geq 1 - \delta$. Thus, the sufficient semantic coverage scales with K/G_ε, completing the proof. □

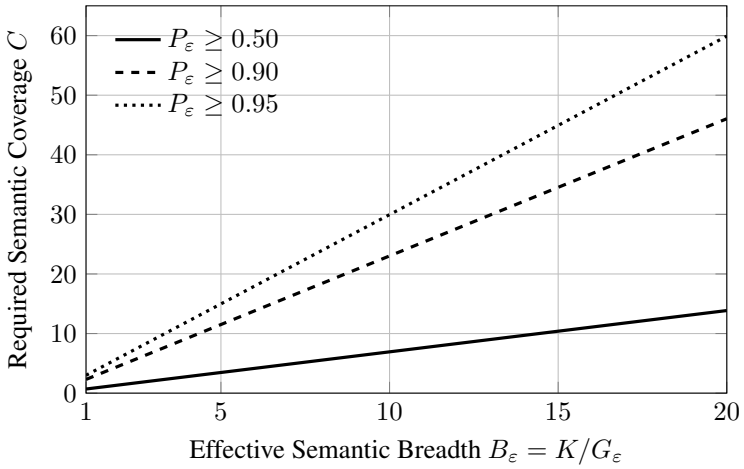


Figure 18: Semantic coverage required to achieve a fixed probability of discovering an ε-optimal research direction. As competitive directions become sparser, the effective semantic breadth $B_\varepsilon = K/G_\varepsilon$ increases, requiring proportionally broader semantic coverage. The curves show the sufficient condition $C \geq B_\varepsilon \log(1/\delta)$ for different target success probabilities $1 - \delta$.

F.4 Theoretical Bound Visualization

Figure 18 illustrates the semantic coverage requirement derived in Proposition 2. Let $B_\varepsilon = K/G_\varepsilon$ denote the effective semantic breadth, where K is the number of admissible semantic directions and G_ε is the number of ε-optimal directions. For a target success probability $1 - \delta$, the sufficient coverage condition can be written as

$$C \geq B_\varepsilon \ln\left(\frac{1}{\delta}\right). \tag{40}$$

The required semantic coverage therefore grows linearly with effective semantic breadth. When competitive directions are sparse, such that G_ε is small relative to K, a search procedure must examine substantially more distinct directions to maintain the same probability of discovering an ε-optimal one. The requirement also becomes steeper as the target success probability increases: at $B_\varepsilon = 20$, for example, the sufficient coverage is approximately 14, 46, and 60 directions for target probabilities of 0.50, 0.90, and 0.95, respectively. Thus, broader semantic coverage is particularly valuable for tasks with sparse competitive directions and when a high probability of success is required.

F.5 Convex Hull Area Box-and-Whisker Plots

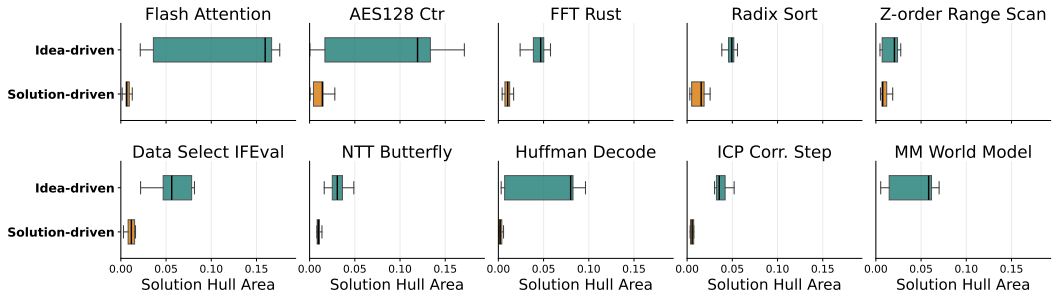


Figure 19: Idea-driven approaches generally show broader coverage of solutions in the embedding space. Embedding cosine similarity-based supplementary analysis is in Appendix F.7.

F.6 Convex Hull Visualization

In Figure 20, we provide visual examples of the convex hulls described in Section 6.

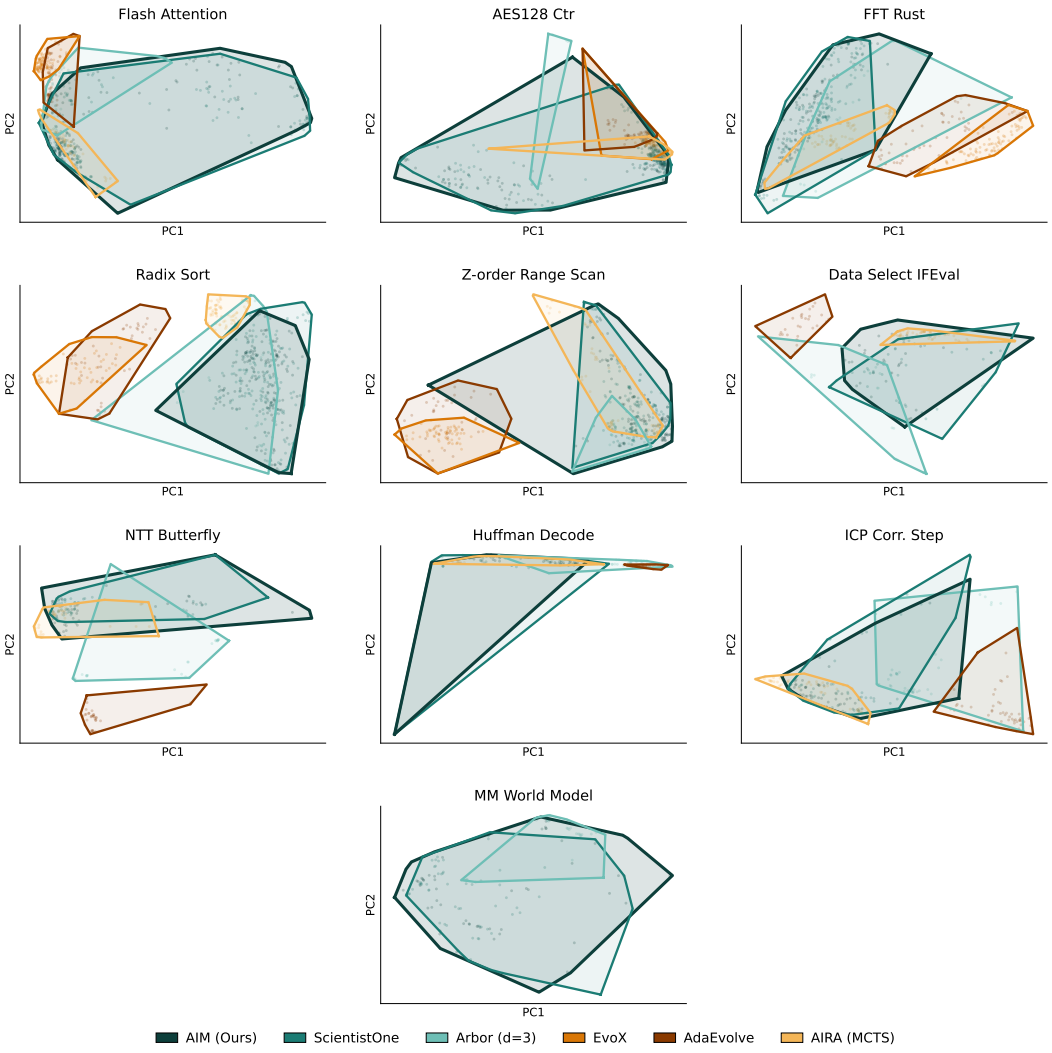


Figure 20: Convex Hull Visualization

F.7 Supplementary Cosine-Similarity-based Diversity Analysis

To supplement the convex-hull area computation in Figure 19, we further provide a comparative analysis between the solution-driven and idea-driven approaches' cosine similarities. We measure how much of the solution space each method actually explores by embedding every generated candidate and averaging the pairwise cosine distance across the resulting pool (Table 6). Specifically, we measure the Mean pairwise cosine distance:

$$\mathcal{D} = \frac{1}{\binom{N}{2}} \sum_{i<j} (1 - \cos(x_i, x_j)), \tag{41}$$

over the embeddings, averaged across three independent runs for each (method, task). Overall, the idea-driven AIM and ScientistOne produce the widest pools on Flash Attention ($\mathcal{D} \approx 0.115$), roughly $3-4\times$ the value produced by every solution-driven baseline. The consistent bottom of the table is populated by solution-driven methods. EvoX collapses to the tightest pool on Flash Attention ($\mathcal{D} = 0.0153$, roughly $7.5\times$ narrower than AIM), confirming that solution-driven approaches

generally search narrower areas compared to idea-driven methods. Also, the measurements' gap is reduced in Radix Sort compared to Flash Attention, exactly matching the observations in the convex-hull area analysis in Figure 19.

Table 6: **Supplementary Cosine Similarity Analysis.** Entries are measured values of equation 41. Higher is more diverse. Bold marks the highest value per column.

Method	Flash Attention	Radix Sort
Idea-driven Approaches		
AIM	0.1150	0.0600
ScientistOne	**0.1199**	0.0661
Arbor	0.0396	**0.0684**
Solution-driven Approaches		
AIRA (MCTS)	0.0311	0.0335
AdaEvolve	0.0297	0.0502
EvoX	0.0153	0.0391

F.8 Practical Considerations

Our theoretical analysis isolates the benefit of semantic breadth by assuming *faithful realization* (Assumption 2): a selected idea can be instantiated as a solution that faithfully reflects its intended research direction. This assumption is useful for understanding the coverage advantage of idea-driven search, but it is slightly idealized. In practice, solver agents may produce incomplete or incorrect implementations, or realize a solution that only partially matches the intended idea. Even when the implementation is faithful, some tasks inherently require substantial solution-level refinement before the value of a research direction can be assessed. For example, machine learning systems may require hyperparameter tuning, while systems optimization tasks may depend on low-level implementation choices that cannot be fully specified at the idea level.

These considerations suggest that practical research agents lie on a continuum between purely idea-driven and purely solution-driven search. At one extreme, allocating each experiment execution to a new idea maximizes semantic breadth, but may under-invest in realizing and optimizing each direction. At the other extreme, repeatedly refining the same implementation can exploit fine-grained execution feedback, but reduces the budget available for exploring alternative research directions. The appropriate balance therefore depends on the task: problems with many meaningfully different high-level approaches may benefit more from semantic exploration, whereas tasks with relatively fixed strategies but substantial implementation-level optimization may benefit more from solution refinement.

Our framework adopts a hybrid design. Although search decisions are made over explicit research ideas, each solver branch is allowed to iteratively refine its implementation using execution and verifier feedback. Thus, the framework retains the semantic organization of idea-driven search while still incorporating the local optimization behavior characteristic of solution-driven approaches. This also motivates our idea–solution auditing mechanism, which explicitly checks whether the final implementation remains aligned with the selected idea before its evaluation is attributed back to the idea-level search process. More broadly, the optimal balance between semantic exploration and implementation refinement is likely task-dependent. An important direction for future work is therefore to adapt this balance dynamically based on observed implementation difficulty, solver fidelity, and the semantic structure of the task, rather than fixing the degree of idea-driven versus solution-driven behavior in advance.