SEDIMA: Cross-Run Hierarchical Insight Memory for Evolutionary Search Agents

Amirhossein Abaskohi[1*], **Mahdi Mostajabdaveh**[2], **Zirui Zhou**[2]

[1]University of British Columbia, [2]Huawei Technologies Canada

Abstract

Large language model (LLM)-driven evolutionary search is a powerful paradigm for automated program and algorithm discovery, yet existing systems are largely *memoryless*: each run explores from scratch, so agents repeatedly rediscover the same improvements and re-encounter the same dead ends. We introduce SEDIMA, a persistent hierarchical insight memory for evolutionary search agents. SEDIMA distills raw traces into natural-language insights, clusters them by semantic similarity using attention-weighted centroids, and retrieves relevant guidance to condition future mutations, accumulating transferable knowledge *across* runs and problems rather than within a single trajectory. As a drop-in module that leaves the search operators unmodified, SEDIMA improves average final performance by **5.5% on AlgoTune** and **6.6% on ALE-Bench LITE** under a fixed budget of 100 evaluated candidates. Under OpenEvolve, SEDIMA reaches baseline-best performance with **32.3% fewer iterations** on average across the five evaluated backbones[1].

1 Introduction

Recent advances have turned LLMs from one-shot code generators into iterative search operators. Rather than producing a single solution, these systems repeatedly select promising candidates, use an LLM to propose semantically informed modifications, evaluate the resulting programs, and retain the strongest variants for further refinement (Romera-Paredes et al., 2023; Novikov et al., 2025). This formulation has enabled progress in mathematical discovery, algorithm design, and heuristic synthesis, while open-source frameworks such as OpenEvolve, ShinkaEvolve, and LLM4AD have made the approach increasingly accessible (Sharma, 2025; Lange et al., 2025; Liu et al., 2026).

Despite this success, these systems are fundamentally *memoryless*: each run begins from scratch and discards the reasoning that produced its successes and failures the moment the run terminates. This is wasteful in two ways. Across runs, an agent applied to a new task re-derives modifications it has already discovered and re-explores directions that were previously neutral or useless, because it retains no record of *why* a past edit helped or hurt. Within a single run, the search trajectory is unstable: solution quality fluctuates sharply across iterations as the agent proposes, forgets, and re-proposes locally useful edits, with no persistent signal consolidating what is working. Recent work begins to close this gap by conditioning mutation on reflective signals rather than raw code (Ye et al., 2024; Wan et al., 2025; Jiang et al., 2026), but in each case memory remains within a single evolutionary trajectory and resets when a new run begins.

We introduce SEDIMA, a **persistent hierarchical insight memory for evolutionary search agents**. SEDIMA distills raw evolution traces into compact natural-language insights capturing why an edit succeeded or failed, then organizes them into a **three-level hierarchy**: raw traces, distilled insights, and semantically clustered insights whose cluster representatives are formed by **attention-weighted centroids**, so that prototypical, high-value lessons dominate retrieval. To condition a mutation, SEDIMA retrieves relevant guidance by semantic similarity and injects it into the prompt.

Two properties distinguish how this memory is used. First, it **persists across runs and problems**, so each run is warm-started with distilled experience rather than starting cold. Second, because insights are **retrieved by semantic content rather than genealogical lineage**, guidance learned on one problem can surface for another that is only loosely related, letting **experience transfer be-**

*Corresponding author: aabaskoh@cs.ubc.ca.

[1]Code is available on GitHub.

The Proceedings of the Second Workshop for REsearch on Agent Language Models (REALM 2026), pages 903–921
October 29, 2026 ©2026 Association for Computational Linguistics

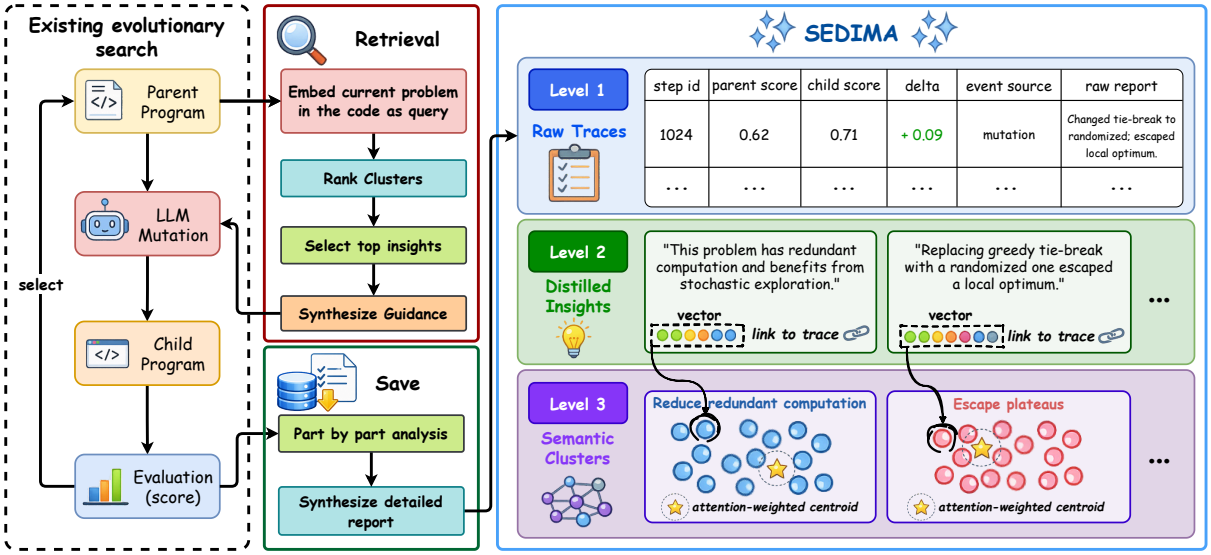

Figure 1: **Overview of SEDIMA .** A persistent three-level memory (right) augments an evolutionary search loop (left, dashed): evaluated children are distilled into hierarchical insights on *save*, and relevant insights are injected into the mutation prompt on *retrieval*.

yond a single task family. Crucially, SEDIMA interfaces with the underlying search at exactly two points, a write on evaluation and a read into the mutation prompt, leaving selection, population management, and mutation operators unmodified. This makes it a **drop-in module** rather than a redesign: we integrate the same memory, unchanged, into two independent open-source harnesses, OpenEvolve (Sharma, 2025) and ShinkaEvolve (Lange et al., 2025), with no modification to either codebase's search logic.

2 Related Work

Prior work has begun to give evolutionary search agents a memory, but it stays confined to a single run or requires rearchitecting the search. ReEvo (Ye et al., 2024) reflects over performance differences, LoongFlow (Wan et al., 2025) summarizes each generation into insights retrieved along the genealogical lineage, and DeltaEvolve (Jiang et al., 2026) stores pairwise semantic deltas; in all three, memory is scoped to one trajectory and discarded when the run ends, whereas SEDIMA persists insight across runs and problems and retrieves it by semantic similarity rather than lineage. CORAL (Qu et al., 2026) and PRISM (Mishra, 2026) add persistent memory but only by turning the loop into long-running multi-agent systems, whereas SEDIMA is a single-agent, drop-in module that leaves the search operators unmodified. Beyond evolutionary search, general-purpose agents accumulate reusable knowledge across episodes as distilled insights (Zhao et al., 2024; Ouyang et al., 2026), skills (Wang et al., 2024), or verified experience libraries (Kong et al., 2025), and others meta-evolve the memory architecture (Zhang et al., 2026) or train a memory policy with dense rewards (Li et al., 2026); SEDIMA instead brings this distill-embed-retrieve recipe to population-based evolution with a non-parametric, hierarchically clustered store whose attention-weighted centroids (Gonçalves et al., 2023) let high-value insights dominate retrieval. To our knowledge, SEDIMA is the first hierarchical memory for single-agent evolutionary search that transfers across runs and problems without changing search operators.

3 SEDIMA

Figure 1 shows how SEDIMA connects to evolutionary search through two hooks: a write after each evaluation and a read before each mutation. It leaves the search logic unchanged while maintaining a persistent three-level memory hierarchy that carries experience across runs and problems.

Level 1: raw traces. After each evaluation, SEDIMA stores a trace containing the step index, parent and child fitness, fitness difference, event source, and raw evaluation report. Level 1 performs no interpretation and serves as evidence for higher-level memories. Retaining these traces allows retrieved recommendations to remain linked to the evaluation events from which they were derived.

Level 2: distilled insights. An LLM converts each trace into a compact natural-language insight and links it back to its Level 1 evidence. For the initial evaluation of a problem, it generates a *problem-level* insight describing the main algorithmic or architectural bottleneck. For each subsequent parent–child evaluation, it generates a *change-level* insight describing the type of modification and its observed performance effect using the parent fitness, child fitness, fitness delta, change summary, and evaluation report. These abstractions remove implementation-specific details while preserving reusable lessons such as beneficial caching, pruning, vectorization, data-structure changes, or reductions in redundant computation. Each insight is embedded for semantic organization and retrieval.

Level 3: semantic clusters. Level 2 insights are grouped by embedding similarity: a new insight joins the nearest cluster if its cosine similarity exceeds a threshold and otherwise starts a new one. Cluster representatives are refreshed as members are added, and an optional LLM summary captures the recurring optimization pattern. Rather than a plain mean, SEDIMA uses a non-parametric attention-weighted centroid that assigns larger weights to semantically central, prototypical insights; lower temperatures sharpen this emphasis, while higher temperatures approach mean pooling. Appendix A provides the formal definitions.

Hierarchical retrieval. Before mutation, SEDIMA embeds a query describing the current program and its observed issues, retrieves the K_c most similar clusters above a threshold, and then selects the K_i most relevant insights from each. Retrieved items retain cluster- and insight-level similarity scores and may include up to two linked Level 1 traces. An LLM converts this evidence into 3 concrete recommendations inserted into the diff or full-rewrite prompt without changing the harness logic. Because retrieval is semantic rather than genealogical, lessons can transfer across problems with similar failure modes or optimization patterns.

Budgeting and fallback. Retrieval is bounded by fixed limits on clusters, insights, and prompt length, so context cost remains constant as memory grows. If no cluster passes the threshold, SEDIMA falls back to the original mutation prompt. Each write adds one distillation call and one insight embedding, while each memory-conditioned mutation adds one query embedding and one recommendation-synthesis call; optional cluster summaries require only periodic write-time calls. Appendix A provides the full procedure.

4 Experiments

4.1 Experimental Setup

Benchmarks and evaluation protocol. We evaluate SEDIMA on AlgoTune (Press et al., 2025), reporting the harmonic mean speedup over numerical programming tasks, and the ten-problem ALE-Bench LITE subset (Imajuku et al., 2025), reporting its problem-agnostic average performance score and separate short- and long-format results. Search uses only public local-evaluation scores as fitness, while all reported results come from private evaluation. We integrate SEDIMA into OpenEvolve (Sharma, 2025) and ShinkaEvolve (Lange et al., 2025) without changing their selection, population, evaluation, or mutation logic. Each run is limited to 100 evolution steps with one evaluated child per step. Comparisons are therefore matched by the number of evaluated candidates rather than total LLM calls or tokens; SEDIMA incurs additional memory-construction and recommendation-synthesis calls as described in Section 3. Maintable cells use one run, while the per-step, ablation, and transfer experiments use three seeds. Memory is populated using disjoint problems, ensuring that no evaluation problem contributes to the memory used to solve it.

Models and implementation. We evaluate GPT-5.4 (OpenAI, 2026), DeepSeek V4 Pro (DeepSeek-AI, 2026), Gemini 3 Pro (Google DeepMind, 2025), Qwen 3.7 Max (Qwen Team, 2026), and Qwen 3 Coder (Cao et al., 2026). All LLM calls within a run use the same backbone, preventing stronger models from being used for memory construction or retrieval. We use a temperature of 0.7 for mutation and 0.2 for insight distillation, cluster summarization, and recommendation synthesis. Embeddings are produced by Qwen-Embedding-4B (Zhang et al., 2025) and fixed across conditions. We set the cluster threshold to $\tau_{\text{cluster}} = 0.8$, retrieval threshold to $\tau_{\text{ret}} = 0.6$, and centroid temperature to $\tau = 0.1$, and retrieve up to $K_c = 3$ clusters with $K_i = 3$ insights per cluster. Retrieved evidence is limited to 2,500 tokens across all models and harnesses. Each insight may include up to two linked traces, and the synthesizer returns three recommendations. Appendix B provides the full configuration. See Appendix G for the prompts used for SEDIMA.

Table 1: AlgoTune, harmonic mean speedup.

Model	OpenEvolve		ShinkaEvolve	
	Base	+SEDIMA	Base	+SEDIMA
GPT-5.4	1.48	1.56	1.53	1.60
DeepSeek V4 Pro	1.75	1.85	1.80	1.89
Gemini 3 Pro	1.83	1.91	1.87	1.94
Qwen 3.7 Max	1.69	1.80	1.74	1.84
Qwen 3 Coder	1.44	1.55	1.49	1.59
Mean	1.64	1.73	1.69	1.77

Table 2: ALE-Bench LITE, average performance.

Model	OpenEvolve		ShinkaEvolve	
	Base	+SEDIMA	Base	+SEDIMA
GPT-5.4	1613.1	1709.9	1660.0	1743.7
DeepSeek V4 Pro	1007.2	1087.8	1055.1	1123.6
Gemini 3 Pro	1182.8	1259.6	1215.3	1281.8
Qwen 3.7 Max	1186.0	1275.0	1225.2	1304.6
Qwen 3 Coder	482.5	535.6	510.0	555.9
Mean	1094.3	1173.6	1133.1	1201.9

4.2 Effectiveness Across Models and Search Harnesses

Tables 1 and 2 show that SEDIMA improves every model under both OpenEvolve and ShinkaEvolve. Across the 20 matched model–harness–benchmark comparisons, all 20 favor SEDIMA (two-sided exact sign test, $p = 1.9 \times 10^{-6}$), showing a consistent direction of improvement across configurations, although individual main-table cells remain single-run results. Averaged across harnesses, SEDIMA raises performance by **5.5%** on AlgoTune and **6.6%** on ALE-Bench LITE, despite ShinkaEvolve generally providing stronger baselines. The largest relative gains occur for Qwen 3 Coder, whose ALE-Bench score improves by 11.0% under OpenEvolve and 9.0% under ShinkaEvolve, suggesting that persistent guidance is especially valuable when the underlying search is weaker. The improvements also hold for both short- and long-format ALE-Bench problems, with larger gains on short contests where recurring optimization patterns are more transferable; the format-level breakdown and per-problem results are reported in Appendices C and F.

4.3 Search Dynamics

Figure 2 tracks AlgoTune score against evolution step over the full 100-step budget, isolating how quickly each configuration reaches a given quality level rather than only where it ends. With GPT-5.4 under OpenEvolve, SEDIMA first surpasses the baseline-best speedup of 1.48× at step 55, whereas

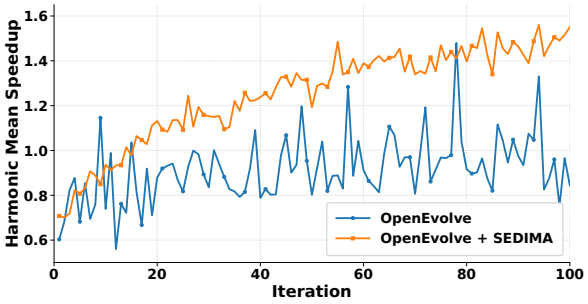

Figure 2: Mean AlgoTune speedup over 100 OpenEvolve iterations with GPT-5.4, with and without SEDIMA, averaged over three seeds.

the baseline reaches this value at step 78. Thus, SEDIMA requires 23 fewer steps, corresponding to a **29.5% reduction** in the iterations needed to reach baseline-best performance. The same trend holds across the remaining models: DeepSeek V4 Pro reaches its baseline-best level in 50 rather than 76 steps (**34.2% fewer**), Gemini 3 Pro in 54 rather than 71 steps (**23.9% fewer**), Qwen 3.7 Max in 52 rather than 84 steps (**38.1% fewer**), and Qwen 3 Coder in 58 rather than 90 steps (**35.6% fewer**). Averaged across the five backbones under OpenEvolve, SEDIMA requires **32.3% fewer iterations** to reach baseline-best performance. These results indicate that persistent retrieval improves not only final solution quality, but also the rate at which evolutionary search discovers high-performing candidates; because the comparison is made by evolution step, this result should not be interpreted as an equivalent reduction in total LLM compute.

4.4 Cross-Run Persistence and Transfer

Table 3 varies memory provenance with evaluation problems fixed. In the cold condition, the memory is empty at deployment but continues to write and retrieve experience online within the run; comparing cold and warm memory therefore isolates the benefit of persistent experience from previous runs beyond within-run memory accumulation. Same-benchmark memory performs best, improving over the cold setting by **4.8%** on AlgoTune and **6.7%** on ALE-Bench. Memory collected from the other benchmark still yields gains of **1.8%** and **2.9%**, respectively, showing that SEDIMA transfers useful optimization knowledge across problem families, although in-domain experience remains more effective.

Additional Analysis A random-retrieval control performs only marginally above no memory (1.62

Table 3: Effect of memory provenance at deployment.

Memory at deployment	AlgoTune	ALE-Bench
Cold (empty)	1.67	1113.7
Warm, same benchmark	1.75	1187.8
Warm, other benchmark	1.70	1146.5

vs. 1.61 on AlgoTune-20 and 1102.6 vs. 1094.3 on ALE-Bench LITE), while full SEDIMA reaches 1.69 and 1173.6, respectively, indicating that the improvement primarily comes from retrieving relevant experience rather than simply adding memory text to the mutation prompt. Appendix E additionally reports component and embedding-model ablations, while Appendix D qualitatively traces an evolved program together with its corresponding memory write and retrieval.

5 Conclusion

We introduced SEDIMA, a persistent hierarchical insight memory for evolutionary search agents. SEDIMA distills raw traces into natural-language insights, organizes them using attention-weighted semantic clusters, and retrieves relevant guidance for future mutations. Through only two interfaces—a write after evaluation and a read before mutation—the same module integrates unchanged with OpenEvolve and ShinkaEvolve without modifying their search operators. Across both harnesses, SEDIMA improves average final performance by **5.5% on AlgoTune** and **6.6% on ALE-Bench LITE** under a fixed budget of 100 evaluated candidates. Under OpenEvolve, it requires **32.3% fewer iterations** on average to reach baseline-best performance across the five evaluated backbones. Because retrieval is semantic rather than genealogical, insights also transfer to problems and benchmark families from which they were not derived. The current store grows without bound, motivating consolidation and forgetting mechanisms at larger scales.

Limitations

SEDIMA depends on LLM-generated insights and embedding-based retrieval, so inaccurate summaries, weak semantic matches, or noisy evaluation reports may store and reuse misleading guidance. Its effectiveness also depends on fixed clustering and retrieval hyperparameters, and the current memory grows without consolidation or forgetting, which may reduce efficiency and relevance at larger scales. Our comparisons match methods by the number of evaluated candidates rather than total LLM calls, tokens, wall-clock time, or monetary cost; because SEDIMA adds distillation, embedding, and recommendation-synthesis operations, reductions in evolution steps should not be interpreted as equivalent reductions in total compute. The main results use one run per model–harness pair because of the cost of full 100-step evolutionary trajectories, while the per-step, ablation, and transfer analyses use three seeds. Finally, we do not directly benchmark against alternative memory-based evolutionary systems, so our experiments establish the benefit of adding SEDIMA to matched memoryless harnesses rather than superiority over all existing memory mechanisms.

Ethical Considerations

SEDIMA inherits the biases, hallucinations, and unsupported reasoning of its underlying language models, and persistent memory may amplify an incorrect insight across later mutations or runs. The additional distillation, retrieval, and synthesis calls also increase computational and environmental cost, although retrieval is bounded by a fixed token budget. Our experiments use public benchmarks, models, and implementations under their respective licenses and involve no human subjects or newly collected personal data. We used LLM assistance only to improve manuscript presentation, with all technical decisions, experiments, and conclusions verified by the authors; released code and artifacts will document dependencies and will redistribute third-party data or model outputs only when permitted by their licenses.

References

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. 2026. Qwen3-coder-next technical report. *Preprint*, arXiv:2603.00729.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2318–2335, Bangkok, Thailand. Association for Computational Linguistics.

DeepSeek-AI. 2026. Deepseek-v4: Towards highly efficient million-token context intelligence. *Preprint*, arXiv:2606.19348.

Simão Gonçalves, Gonçalo Correia, Diogo Pernes, and Afonso Mendes. 2023. Supervising the centroid baseline for extractive multi-document summarization. In *Proceedings of the 4th New Frontiers in Summarization Workshop*, pages 87–96, Singapore. Association for Computational Linguistics.

Google DeepMind. 2025. Gemini 3 Pro.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. 2025. ALE-bench: A benchmark for long-horizon objective-driven algorithm engineering. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Jiachen Jiang, Tianyu Ding, and Zhihui Zhu. 2026. Deltaevolve: Accelerating scientific discovery through momentum-driven evolution. In *Forty-third International Conference on Machine Learning*.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, Hai Wang, Cathy Wu, and Jinhua Zhao. 2025. AlphaOPT: Formulating optimization programs with self-improving LLM experience library. In *NeurIPS Workshop on GPU-Accelerated and Scalable Optimization*.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. 2025. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *Preprint*, arXiv:2509.19349.

Jian Li, Yizhang Jin, Dongqi Liu, Hang Ding, Jiafu Wu, Dongsheng Chen, Yunhang Shen, Yulei Qin, Ying Tai, Chengjie Wang, Xiaotong Yuan, and Yabiao Wang. 2026. Se-search: Self-evolving search agent via memory and dense reward. *Preprint*, arXiv:2603.03293.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *Preprint*, arXiv:2308.03281.

Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Qinglong Hu, Ping Guo, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2026. Llm4ad: A platform for algorithm design with large language model. *Preprint*, arXiv:2412.17287.

Suyash Mishra. 2026. Prism: An evolutionary memory substrate for multi-agent open-ended discovery. *Preprint*, arXiv:2604.19795.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *Preprint*, arXiv:2506.13131.

OpenAI. 2026. Introducing GPT-5.4.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Ori Press, Brandon Amos, Haoyu Zhao, Yikai Wu, Samuel Ainsworth, Dominik Krupke, Patrick Kidger, Touqir Sajed, Bartolomeo Stellato, Jisun Park, Nathanael Bosch, Eli Meril, Albert Steppi, Arman Zharmagambetov, Fangzhao Zhang, David Pérez-Piñeiro, Alberto Mercurio, Ni Zhan, Talor Abramovich, and 5 others. 2025. Algotune: Can language models speed up general-purpose numerical programs? In *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. 2026. CORAL: Towards autonomous multi-agent evolution for open-ended discovery. In *Proceedings of the Conference on Language Modeling*.

Qwen Team. 2026. Qwen3.7: The agent frontier.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. 2023. Mathematical discoveries from program search with large language models. *Nature*.

Asankhaya Sharma. 2025. Openevolve: an open-source evolutionary coding agent.

Chunhui Wan, Xunan Dai, Zhuo Wang, Minglei Li, Yanpeng Wang, Yinan Mao, Yu Lan, and Zhiwen Xiao. 2025. Loongflow: Directed evolutionary search via a cognitive plan-execute-summarize paradigm. *Preprint*, arXiv:2512.24077.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*.

Guibin Zhang, Haotian Ren, Chong Zhan, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng YAN. 2026. Memevolve: Meta-evolution of agent memory systems. In *Forty-third International Conference on Machine Learning*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *Preprint*, arXiv:2506.05176.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642.

A Memory Construction and Retrieval Details

Cluster assignment. Let $e_i \in \mathbb{R}^d$ denote the embedding of a newly distilled insight and let c_k denote the representative of cluster k. We identify the nearest cluster as

$$k^* = \arg\max_k \cos(e_i, c_k). \tag{1}$$

The insight joins k^* when $\cos(e_i, c_{k^*}) \geq \tau_{\text{cluster}}$ and otherwise initializes a new cluster. Problem-level and change-level insights follow the same assignment procedure.

Attention-weighted representative. For a cluster containing embeddings $\{e_1, \ldots, e_n\}$, we compute the semantic centrality of member i as

$$s_i = \frac{1}{n-1} \sum_{j \neq i} \cos(e_i, e_j). \tag{2}$$

We transform these scores into non-parametric attention weights,

$$w_i = \frac{\exp(s_i/\tau)}{\sum_{j=1}^{n} \exp(s_j/\tau)}, \tag{3}$$

and form the cluster representative as

$$c = \sum_{i=1}^{n} w_i e_i. \tag{4}$$

For a singleton cluster, we set $c = e_1$. As $\tau \to \infty$, the weights approach a uniform distribution and the representative approaches mean pooling.

Hierarchical retrieval. Given query embedding e_q, we rank clusters according to

$$r_k^{\text{cluster}} = \cos(e_q, c_k) \tag{5}$$

and retain the top K_c clusters satisfying $r_k^{\text{cluster}} \geq \tau_{\text{ret}}$. Within each retained cluster, member insights are ranked by

$$r_i^{\text{insight}} = \cos(e_q, e_i), \tag{6}$$

and the top K_i insights are selected. Cluster and insight similarities are retained separately rather than combined into a single score.

Evidence and recommendation synthesis. For each retrieved insight, SEDIMA optionally attaches up to two linked Level 1 traces. The synthesis prompt receives the current query, insight text, cluster similarity, insight similarity, and available raw evidence, and returns 3 actionable recommendations grounded in the retrieved records. If no relevant cluster is found, no synthesis call is made and the original mutation prompt is used unchanged.

Online updates. Memory construction and retrieval occur throughout a run. Each newly evaluated candidate immediately produces a Level 1 trace and a Level 2 insight, after which its embedding is assigned to Level 3. Because the store is persisted on disk, later mutations can retrieve insights written earlier in the same run as well as insights retained from previous runs.

B Experimental Setup Details

Benchmarks. AlgoTune consists of numerical programming tasks in which a generated solver must reproduce a reference implementation while reducing execution time. We report the harmonic mean of per-task speedups. ALE-Bench contains AtCoder Heuristic Contest problems with continuous, non-saturating objectives. We use the ten-problem LITE subset and report average performance, a problem-agnostic score derived from the human leaderboard. Following Imajuku et al. (2025), we avoid rating because it can be dominated by a single exceptional result, and additionally report the five short- and five long-format problems separately.

Evaluation protocol. During evolution, candidate fitness is computed only from the public local cases supplied by each benchmark. Final results

are obtained through private evaluation, and private scores are never exposed to the search procedure. Because our 100-step ALE-Bench budget is smaller than that used in published leaderboard submissions, absolute scores should be interpreted only through matched comparisons within our experiments.

Harnesses and search budget. We attach SEDIMA to OpenEvolve and ShinkaEvolve through the evaluation-write and mutation-read hooks. Their original selection, island-based population management, evaluation, and mutation procedures remain unchanged. Each run contains at most 100 evolution steps, with one child generated and evaluated per step. The main tables report one run per model–harness pair because of the cost of full evolutionary trajectories. The per-step, component-ablation, embedding-ablation, and memory-transfer experiments are each averaged over three independent seeds.

Memory isolation. Memory is constructed from a disjoint set of problems before deployment on the evaluation set. No trace, insight, or cluster originating from an evaluation problem is available when that problem is first solved. During evaluation, the store is updated online, allowing later steps to retrieve insights generated earlier in the same trajectory together with eligible knowledge from prior runs.

Models and decoding. All language-model requests are issued through OpenRouter.[2] We evaluate GPT-5.4, DeepSeek V4 Pro, Gemini 3 Pro, Qwen 3.7 Max, and Qwen 3 Coder. Within each run, the same model performs mutation, trace distillation, optional cluster summarization, and recommendation synthesis, preventing memory-assisted conditions from receiving a stronger model than their corresponding baselines. We use a temperature of 0.7 for mutation to preserve search diversity and 0.2 for distillation, cluster summarization, and recommendation synthesis to improve consistency. We use QWEN-EMBEDDING-4B (Zhang et al., 2025) for all insight and query embeddings and keep it fixed across experiments.

Memory hyperparameters. We use a cluster-assignment threshold of $\tau_{\text{cluster}} = 0.80$, a retrieval threshold of $\tau_{\text{ret}} = 0.60$, and a centroid temperature of $\tau = 0.10$. Retrieval expands at most $K_c = 3$ clusters and selects at most $K_i = 3$ insights from each cluster, under a fixed budget of 2,500 tokens. Cluster summaries use at most ten member insights, each retrieved insight may include up to two linked Level 1 traces, and the synthesizer returns exactly three recommendations. These settings are shared across benchmarks, models, and harnesses. If no cluster exceeds τ_{ret}, no recommendation-synthesis call is made and the original mutation prompt is used. Bounding retrieval by K_c, K_i, and the token budget keeps prompt length independent of the total memory size.

[2] https://openrouter.ai/

Table 4: ALE-Bench LITE by contest format, averaged over models.

Method	Short	Long	Overall
OpenEvolve	1166.8	1021.8	1094.3
+ SEDIMA	1261.7	1085.5	1173.6
ShinkaEvolve	1209.4	1056.8	1133.1
+ SEDIMA	1292.6	1111.2	1201.9

C ALE-Bench Results by Contest Format

Table 4 separates ALE-Bench LITE performance into its five short- and five long-format problems. SEDIMA improves short-format performance by 8.1% under OpenEvolve and 6.9% under ShinkaEvolve, while improving long-format performance by 6.2% and 5.1%, respectively. The larger gains on short contests suggest that their recurring optimization strategies transfer more readily, while the consistent long-format improvements show that retrieved insights remain useful for more problem-specific search.

D Qualitative Example of Memory Use

This section illustrates how SEDIMA converts an evolutionary evaluation into reusable knowledge and later retrieves that knowledge to guide a concrete code modification. We consider a capacitated facility-location task in which an intermediate candidate achieves a $2.36\times$ speedup over the original reference implementation. A subsequent candidate replaces the CBC mixed-integer programming backend with OR-Tools CP-SAT and reaches a $3.99\times$ speedup. Because both values are measured relative to the same reference implementation, the CP-SAT candidate is $3.99/2.36 = 1.69\times$ faster than the intermediate candidate, corresponding to a further 40.9% runtime reduction.

D.1 Intermediate Candidate and Remaining Bottleneck

The intermediate candidate formulates the problem using binary facility-opening variables y_i and binary customer-assignment variables x_{ij}. Each customer is assigned to exactly one facility, and facility capacities are enforced through

$$\sum_j d_j x_{ij} \leq C_i y_i. \tag{7}$$

The core of the $2.36\times$ candidate is shown below.

Intermediate candidate: CBC formulation at $2.36\times$

```python
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver("CBC")
if not solver:
    solver = pywraplp.Solver.CreateSolver("SCIP")

# Binary facility-opening and assignment variables
y = [
    solver.BoolVar(f"y_{i}")
    for i in range(n_facilities)
]

x = [
    [
        solver.BoolVar(f"x_{i}_{j}")
        for j in range(n_customers)
    ]
    for i in range(n_facilities)
]

# Objective
total_cost = sum(
    fixed_costs[i] * y[i]
    for i in range(n_facilities)
)
total_cost += sum(
    transportation_costs[i][j] * x[i][j]
    for i in range(n_facilities)
    for j in range(n_customers)
)
solver.Minimize(total_cost)

# Each customer is assigned to exactly one facility
for j in range(n_customers):
    solver.Add(
        sum(x[i][j] for i in range(n_facilities)) == 1
    )

# Facility-capacity constraints
for i in range(n_facilities):
    solver.Add(
        sum(
            demands[j] * x[i][j]
            for j in range(n_customers)
        ) <= capacities[i] * y[i]
    )

status = solver.Solve()
```

Representative evaluator diagnosis

The candidate is correct and improves substantially over the reference implementation, but its remaining runtime is dominated by solving a dense binary mixed-integer program with CBC. The model consists almost entirely of Boolean decisions and repeated exactly-one assignment constraints, yet these structures are represented using generic linear constraints. A Boolean-oriented solver with native cardinality constraints may exploit this structure more effectively.

The formulation contains $n_f n_c$ binary assignment variables in addition to the n_f facility-opening variables. Python-side model construction and dense result extraction introduce some overhead, but the main remaining bottleneck is the CBC branch-and-bound search.

D.2 Writing the Experience to SEDIMA

After evaluation, SEDIMA stores the factual execution trace at Level 1. An LLM then converts the trace into a compact Level 2 insight describing the reusable optimization principle. The insight is embedded and assigned to a Level 3 cluster containing semantically related experiences from assignment, scheduling, routing, and resource-allocation tasks.

Table 5 shows an example snapshot of the resulting memory state.

D.3 Reading the Memory and Applying the Retrieved Insight

Before the next mutation, SEDIMA embeds a query describing the current implementation and its remaining bottleneck. The query is close to the Boolean-assignment cluster because it contains the same structural signals as the stored experiences: binary assignment decisions, exactly-one constraints, capacity-limited resources, and a solver-dominated execution profile.

Retrieved cross-problem experience

A previous capacitated scheduling task used CBC even though nearly all decisions were Boolean. Recasting the model in CP-SAT and replacing generic sum-to-one equalities with native cardinality constraints substantially reduced solver time. Floating-point coefficients had to be converted to bounded integers before constructing the CP-SAT model.

The retrieved experience does not need to originate from the same facility-location task. Retrieval is based on semantic similarity, so an insight obtained from scheduling or resource assignment can

transfer when the underlying computational pattern is similar.

The retrieved guidance results in the following candidate.

Memory-guided candidate: CP-SAT formulation at 3.99×

```python
from ortools.sat.python import cp_model
import numpy as np

fixed_costs = np.asarray(problem["fixed_costs"])
capacities = np.asarray(problem["capacities"])
demands = np.asarray(problem["demands"])
transportation_costs = np.asarray(
    problem["transportation_costs"]
)

# CP-SAT requires integer coefficients
precision = 1_000_000
fixed_costs = (fixed_costs * precision).astype(int
    ↪ )
capacities = (capacities * precision).astype(int)
demands = (demands * precision).astype(int)
transportation_costs = (
    transportation_costs * precision
).astype(int)

model = cp_model.CpModel()

# Binary facility-opening and assignment variables
y = [
    model.NewBoolVar(f"y_{i}")
    for i in range(n_facilities)
]

x = {
    (i, j): model.NewBoolVar(f"x_{i}_{j}")
    for i in range(n_facilities)
    for j in range(n_customers)
}

# Native exactly-one constraints
for j in range(n_customers):
    model.AddExactlyOne(
        [x[i, j] for i in range(n_facilities)]
    )

# Facility-capacity constraints
for i in range(n_facilities):
    model.Add(
        sum(
            demands[j] * x[i, j]
            for j in range(n_customers)
        ) <= capacities[i] * y[i]
    )

# Integer-valued objective
objective = sum(
    fixed_costs[i] * y[i]
    for i in range(n_facilities)
)
objective += sum(
    transportation_costs[i][j] * x[i, j]
    for i in range(n_facilities)
    for j in range(n_customers)
)
model.Minimize(objective)

solver = cp_model.CpSolver()
solver.parameters.max_time_in_seconds = 30.0
status = solver.Solve(model)
```

The retrieved memory leads to three principal modifications. First, the generic CBC mixed-integer solver is replaced with CP-SAT, which is designed for discrete models dominated by Boolean and integer variables. Second, the constraint $\sum_i x_{ij} = 1$ is represented using the native `AddExactlyOne` primitive, exposing its cardinality structure directly to the solver. Third, floating-point costs, capacities, and demands are scaled to integers, as required by CP-SAT.

D.4 Interpretation

This example demonstrates the full SEDIMA write–retrieve–apply cycle. The evaluator first identifies that the 2.36× candidate remains solver-bound. SEDIMA stores the raw evidence, distills it into a solver-selection insight, and clusters it with structurally similar experiences. During a later mutation, the current program retrieves this insight and uses it to reformulate the model with CP-SAT, producing the 3.99× candidate.

The 3.99× implementation also introduces a 30-second solver limit and accepts both `OPTIMAL` and `FEASIBLE` CP-SAT solutions, whereas the CBC implementation returns a solution only after proving optimality. The observed gain may therefore reflect both a more suitable solver formulation and a different termination policy. A controlled evaluation should additionally compare objective values and report how often CP-SAT terminates with an optimal rather than merely feasible solution.

E Ablation Studies

We conduct all ablations under OpenEvolve using three independent seeds for each of the five mutation models and report results averaged across both seeds and models. To control evaluation cost, AlgoTune results use its first 20 tasks, while ALE-Bench results use the complete ten-problem LITE subset. The full-SEDIMA and no-memory ALE-Bench means happen to round to the same values as the corresponding OpenEvolve means in Table 2, but are computed from these separate three-seed ablation runs.

Memory components. Table 7 isolates the contribution of each memory component. The full SEDIMA achieves the strongest performance on both benchmarks, improving over no memory by 5.0% on AlgoTune-20 and 7.2% on ALE-Bench LITE. Replacing the attention-weighted representative with a mean centroid causes a modest decline, indicating that emphasizing prototypical insights improves retrieval. Using Level 2 insights without semantic clustering produces a larger drop, show-

Table 5: Example snapshot of the information written to and retrieved from SEDIMA. Level 1 preserves task-specific evidence, while Levels 2 and 3 support transfer across problems.

Operation	Memory content	Function
Level 1 write	**Task:** capacitated facility location. **Score:** $2.36\times$. **Backend:** CBC. **Structure:** dense binary assignment variables, facility-opening variables, exactly-one customer assignments, and capacity constraints. **Diagnosis:** solver execution remains the dominant cost.	Preserves the candidate score, implementation details, evaluator evidence, and provenance of the higher-level insight.
Level 2 write	Boolean assignment models with exactly-one and capacity constraints may be solved more efficiently using a Boolean-oriented constraint solver. Native cardinality constraints should be exposed directly instead of being represented only as generic linear equalities.	Abstracts the facility-location trace into a reusable principle applicable to other discrete optimization problems.
Level 3 write	**Cluster:** Boolean assignment models and solver selection. Related entries include job assignment, capacitated scheduling, routing, and resource-activation problems.	Groups experiences by computational structure rather than benchmark identity or evolutionary lineage.
Retrieval query	The current program uses CBC for a dense Boolean facility-location model. Runtime is dominated by the solver, and each customer participates in an exactly-one assignment constraint.	Matches the stored cluster through shared concepts: Boolean decisions, assignment structure, capacities, and solver-bound execution.
Retrieved guidance	Replace the generic MILP backend with CP-SAT, use native `AddExactlyOne` constraints, and convert floating-point coefficients to bounded integers.	Provides a concrete mutation direction that can be directly translated into code.
Resulting candidate	CBC is replaced with `CpModel` and `CpSolver`; generic assignment equalities are replaced with `AddExactlyOne`; numerical coefficients are scaled to integers; and a bounded solve time is introduced.	Produces the $3.99\times$ candidate, a further $1.69\times$ improvement over the $2.36\times$ intermediate candidate.

ing the benefit of hierarchical organization. Random retrieval performs only marginally above no memory, confirming that gains arise from retrieving relevant experience rather than merely adding prior text to the mutation prompt.

Embedding model. Table 8 evaluates whether SEDIMA depends on a particular embedding backbone. Qwen3-Embedding-4B performs best on both benchmarks, but Qwen3-Embedding-8B and GTE-Qwen2-7B-Instruct remain close, indicating that the method is robust across strong semantic encoders. Performance declines more noticeably with BGE-M3 and Qwen3-Embedding-0.6B, suggesting that weaker representations reduce cluster assignment and retrieval quality. Nevertheless, every embedding-based configuration outperforms the no-memory baseline, showing that the benefit of SEDIMA is not specific to its default encoder.

F Per-Problem Results

F.1 AlgoTune

Tables 9, 10, 11, and 12 report the full AlgoTune breakdown. Performance is strongly task dependent: many low-level kernels remain near $1\times$, whereas graph, optimization, control, and numerical tasks such as `lp_mdp`, `water_filling`, and `convolve2d_full_fill` admit substantially larger speedups. ShinkaEvolve generally achieves slightly stronger per-task results than OpenEvolve, while the best mutation model varies across tasks, showing that the aggregate gains are not driven by a single model or problem category.

F.2 ALEBench

Tables 13 and 14 provide the ALE-Bench LITE per-problem results. GPT-5.4 achieves the strongest overall scores, followed by Qwen 3.7 Max and Gemini 3 Pro, while ShinkaEvolve is generally stronger across models and tasks. Performance varies considerably across problems and formats, with short contests typically scoring higher, although strong results on long-format tasks such as `ahc027` show that the method is not limited to shorter search problems.

Table 6: Connection between the retrieved memory and the resulting code modifications.

Retrieved information	Resulting code modification
Boolean-dominated optimization models may benefit from CP-SAT.	Replace `pywraplp.Solver` and CBC with `CpModel` and `CpSolver`.
Exactly-one assignment structure should be exposed natively.	Replace generic sum-equals-one constraints with `AddExactlyOne`.
CP-SAT requires bounded integer coefficients.	Scale costs, demands, and capacities by a shared precision factor and convert them to integers.
Difficult solver runs should have bounded execution time.	Set `max_time_in_seconds` to 30.

Table 7: Component ablation under OpenEvolve, averaged over three seeds and five mutation models. AlgoTune results use the first 20 tasks, while ALE-Bench uses the full LITE set.

Variant	AlgoTune-20	ALE-Bench LITE
SEDIMA (full)	1.69	1173.6
Level 2 only	1.65	1138.9
Mean centroid	1.67	1157.4
Random retrieval	1.62	1102.6
No memory	1.61	1094.3

Table 8: Embedding-model ablation under OpenEvolve, averaged over three seeds and five mutation models. AlgoTune results use the first 20 tasks, while ALE-Bench uses the full LITE set.

Embedding model	AlgoTune-20	ALE-Bench LITE
Qwen3-Embedding-4B 2025	1.69	1173.6
Qwen3-Embedding-8B 2025	1.68	1168.4
GTE-Qwen2-7B-Instruct 2023	1.67	1159.7
BGE-M3 2024	1.65	1147.2
Qwen3-Embedding-0.6B 2025	1.64	1136.8

G Prompts

The prompts used by SEDIMA are provided in Prompts G.1, G.2, G.3, and G.4. Prompt G.1 summarizes the main bottleneck from the initial evaluation, while Prompt G.2 describes each modification and its observed fitness effect. Prompt G.3 abstracts the recurring optimization pattern shared by related insights. Finally, Prompt G.4 converts retrieved insights, similarity scores, and linked evidence into exactly three grounded recommendations for the next mutation.

Prompt G.1: Problem-Level Insight Generation

You summarize performance bottlenecks at a high level from a single raw evolution record.
Write a localized insight that identifies the primary bottleneck and the main challenge reflected in the evaluation report.

Requirements

- Write 2–4 concise sentences.
- Focus on algorithmic or architectural issues.
- Do not mention source code, file names, identifiers, or line-level details.
- Ground the insight only in the provided fitness score and evaluation report.

Fitness score
{{fitness_score}}
Raw evaluation report
{{raw_report}}
Return only the resulting insight without additional explanation.

Prompt G.2: Change-Level Insight Generation

You summarize a localized implementation change from a single raw evolution record at a high level.
Write an insight describing the type of change and its observed impact on performance.

Requirements

- Write 1–2 concise sentences.
- Describe the change using general concepts such as caching, pruning, vectorization, data-structure modification, or recursion reduction.
- Do not mention specific source-code identifiers, file names, or line-level details.
- Ground the stated impact in the parent fitness, child fitness, fitness delta, change summary, and evaluation report.

Parent fitness
{{parent_fitness}}
Child fitness
{{child_fitness}}
Fitness delta
{{fitness_delta}}
Changes summary
{{changes_summary}}
Raw evaluation report
{{raw_report}}
Return only the resulting insight without additional explanation.

Table 9: Per-task speedup on AlgoTune using OpenEvolve+SEDIMA (part 1 of 2).

Task	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
aes_gcm_encryption	1.12	1.06	1.18	1.05	1.02
affine_transform_2d	1.00	1.00	1.00	1.00	1.00
aircraft_wing_design	1.08	1.12	2.13	1.21	1.14
articulation_points	2.37	4.95	2.39	3.87	2.77
base64_encoding	1.06	1.00	1.10	1.00	1.00
battery_scheduling	8.54	14.85	31.72	15.62	12.94
btsp	1.19	2.39	2.34	2.00	1.30
capacitated_facility_location	4.58	10.34	14.01	7.37	2.45
chacha_encryption	1.22	1.18	1.24	1.16	1.08
channel_capacity	1.07	1.14	1.45	1.15	1.10
chebyshev_center	2.99	4.58	7.98	4.63	3.91
cholesky_factorization	1.07	1.04	1.08	1.04	1.02
clustering_outliers	1.16	1.29	1.20	1.25	1.17
communicability	23.50	54.32	146.67	59.25	50.18
convex_hull	1.41	3.83	6.02	3.19	1.65
convolve2d_full_fill	46.10	99.73	174.44	95.49	68.89
convolve_1d	1.04	1.10	1.13	1.08	1.03
correlate2d_full_fill	34.41	100.21	133.94	90.74	63.02
correlate_1d	1.05	1.08	1.25	1.08	1.04
count_connected_components	2.49	5.44	5.07	4.71	3.48
count_riemann_zeta_zeros	1.00	1.00	1.00	1.00	1.00
cumulative_simpson_1d	5.49	9.37	12.95	6.89	2.42
cumulative_simpson_multid	1.03	1.00	1.05	1.00	1.00
cvar_projection	1.37	2.65	1.52	2.23	1.73
cyclic_independent_set	1.47	10.57	2.12	5.43	1.76
dct_type_I_scipy_fftpack	1.06	1.08	1.45	1.11	1.06
delaunay	1.85	3.29	1.95	2.50	1.43
dijkstra_from_indices	1.00	1.00	1.00	1.00	1.00
discrete_log	1.04	1.35	1.11	1.24	1.07
dst_type_II_scipy_fftpack	1.28	1.63	1.33	1.47	1.17
dynamic_assortment_planning	19.75	32.40	47.10	22.44	5.23
earth_movers_distance	1.01	1.01	1.01	1.01	1.00
edge_expansion	3.90	13.45	3.12	7.35	2.27
eigenvalues_complex	1.32	1.70	2.15	1.66	1.49
eigenvalues_real	1.85	2.90	4.04	2.79	2.32
eigenvectors_complex	1.03	1.05	1.06	1.04	1.02
eigenvectors_real	1.02	1.04	1.12	1.04	1.03
elementwise_integration	1.00	1.00	1.00	1.00	1.00
feedback_controller_design	21.53	102.99	79.46	53.35	6.32
fft_cmplx_scipy_fftpack	1.71	2.65	2.57	2.48	2.22
fft_convolution	1.00	1.00	1.00	1.00	1.00
fft_real_scipy_fftpack	1.08	1.67	1.29	1.79	2.35
firls	1.00	1.00	1.00	1.01	1.01
generalized_eigenvalues_complex	2.50	4.76	4.23	4.07	3.05
generalized_eigenvalues_real	1.86	3.27	3.92	3.00	2.35
generalized_eigenvectors_complex	1.68	3.21	4.14	2.74	1.57
generalized_eigenvectors_real	1.41	2.09	4.20	2.20	1.95
graph_coloring_assign	3.44	2.68	2.54	2.44	1.60
graph_global_efficiency	2.87	10.11	18.73	9.67	8.08
graph_isomorphism	19.01	44.19	65.78	37.46	21.06
graph_laplacian	1.00	1.00	1.00	1.00	1.00
group_lasso	1.00	1.00	1.00	1.00	1.00
gzip_compression	1.12	1.10	1.11	1.09	1.05
integer_factorization	1.00	1.00	1.00	1.00	1.00
job_shop_scheduling	1.43	1.95	2.17	1.75	1.30
kalman_filter	5.72	12.54	14.94	9.17	2.85
kcenters	1.44	2.03	1.49	1.74	1.25
kd_tree	1.06	1.07	1.06	1.06	1.03
kernel_density_estimation	1.00	1.00	1.00	1.00	1.00
kmeans	6.30	12.69	21.17	12.59	10.18
ks_test_2samp	1.03	1.00	1.05	1.00	1.00
l0_pruning	1.23	1.74	3.26	1.94	2.12
l1_pruning	2.84	2.92	3.57	2.76	1.92
lasso	1.12	1.59	1.21	1.41	1.12
least_squares	1.16	2.25	1.89	2.05	1.81
linear_system_solver	1.09	1.15	1.30	1.14	1.09
lp_box	6.54	14.39	22.13	13.38	9.63
lp_centering	1.01	1.02	1.01	1.02	1.01
lp_mdp	87.53	163.16	291.43	110.55	21.50
lqr	1.08	1.32	1.58	1.27	1.10
lti_simulation	1.51	6.99	3.63	4.43	1.73
lu_factorization	1.00	1.05	1.01	1.08	1.15
lyapunov_stability	38.44	101.27	109.17	86.73	58.43
markowitz	1.01	1.00	1.02	1.00	1.00
matrix_completion	1.00	1.01	1.01	1.01	1.02
matrix_exponential	1.00	1.00	1.00	1.00	1.00
matrix_exponential_sparse	1.00	1.00	1.00	1.00	1.00

Table 10: Per-task speedup on AlgoTune using OpenEvolve+SEDIMA (part 2 of 2).

Task	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
matrix_multiplication	1.03	1.02	1.03	1.02	1.01
matrix_sqrt	1.02	1.02	1.02	1.02	1.01
max_clique_cpsat	4.67	12.24	13.52	10.57	6.01
max_common_subgraph	4.92	4.19	12.34	4.38	2.49
max_flow_min_cost	2.68	6.90	4.03	5.99	6.21
max_independent_set_cpsat	4.54	3.18	3.31	2.87	1.74
max_weighted_independent_set	1.20	1.18	1.63	1.21	1.12
min_dominating_set	1.08	1.83	1.21	1.65	1.48
min_weight_assignment	1.16	1.64	1.32	1.43	1.12
minimum_spanning_tree	3.76	6.24	3.17	4.11	1.76
minimum_volume_ellipsoid	3.53	18.21	18.53	11.59	2.93
multi_dim_knapsack	2.60	16.97	9.41	9.83	2.94
nmf	1.03	1.23	1.13	1.21	1.20
ode_brusselator	6.42	3.67	2.95	3.26	1.89
ode_fitzhughnagumo	1.05	1.12	1.09	1.08	1.03
ode_hires	5.69	17.23	29.18	13.76	5.76
ode_hodgkinhuxley	19.23	40.29	91.74	33.97	11.25
ode_lorenz96_nonchaotic	1.52	2.79	2.92	2.47	1.84
ode_lotkavolterra	16.93	11.91	48.16	12.96	5.58
ode_nbodyproblem	16.06	32.05	33.01	25.60	14.15
ode_seirs	24.48	6.58	43.39	10.82	11.96
ode_stiff_robertson	4.34	22.37	6.37	12.03	3.75
ode_stiff_vanderpol	24.79	45.97	6.12	21.65	4.88
odr	1.01	1.02	1.01	1.02	1.01
optimal_advertising	1.04	1.31	1.10	1.22	1.06
outer_product	1.13	1.03	1.21	1.02	1.01
pagerank	1.45	1.97	16.91	3.03	3.46
pca	2.14	3.75	4.06	3.01	1.61
pde_burgers1d	1.34	3.74	4.56	3.55	3.07
pde_heat1d	1.30	1.96	1.40	1.67	1.21
polynomial_mixed	4.96	5.55	2.93	4.16	1.96
polynomial_real	6.57	35.49	4.16	14.86	2.90
power_control	22.66	139.92	122.11	97.36	27.97
procrustes	1.32	1.28	1.29	1.26	1.13
psd_cone_projection	3.15	4.65	10.56	5.38	5.56
qp	1.38	1.90	2.56	1.87	1.68
qr_factorization	1.90	1.62	1.80	1.63	1.41
quantile_regression	1.16	1.34	1.88	1.35	1.24
queens_with_obstacles	1.67	3.18	4.26	2.92	2.05
queuing	1.06	1.04	1.07	1.04	1.02
qz_factorization	1.21	1.17	1.19	1.16	1.08
randomized_svd	2.04	3.70	2.14	2.73	1.47
rbf_interpolation	1.01	1.01	1.01	1.01	1.00
rectanglepacking	1.33	1.33	3.03	1.46	1.25
robust_kalman_filter	2.43	3.45	9.61	3.97	3.43
robust_linear_program	2.14	1.88	7.41	2.23	1.67
rocket_landing_optimization	1.07	1.59	1.24	1.40	1.11
rotate_2d	1.00	1.00	1.00	1.00	1.00
set_cover	4.05	6.38	4.48	4.65	2.00
set_cover_conflicts	1.95	2.72	3.44	2.69	2.24
sha256_hashing	1.00	1.00	1.00	1.00	1.00
shift_2d	1.00	1.00	1.00	1.00	1.00
shortest_path_dijkstra	1.71	2.50	3.87	2.25	1.45
sinkhorn	1.36	2.05	3.14	1.92	1.35
sparse_eigenvectors_complex	1.00	1.00	1.00	1.00	1.00
sparse_lowest_eigenvalues_posdef	1.47	2.13	2.56	2.04	1.76
sparse_lowest_eigenvectors_posdef	1.52	2.55	2.82	2.29	1.71
sparse_pca	2.65	6.86	8.32	5.45	2.60
spectral_clustering	3.99	8.57	14.73	6.43	2.31
stable_matching	1.37	1.83	2.36	1.79	1.57
svd	1.19	1.17	1.18	1.16	1.08
svm	1.00	1.00	1.00	1.00	1.00
sylvester_solver	1.02	1.01	1.02	1.01	1.01
tensor_completion_3d	14.09	23.09	40.25	18.72	6.10
toeplitz_solver	1.00	1.00	1.00	1.00	1.00
tsp	1.16	1.37	1.29	1.29	1.19
two_eigenvalues_around_0	1.52	2.04	2.77	1.99	1.72
unit_simplex_projection	1.71	3.18	1.80	2.44	1.42
upfirdn1d	1.02	1.16	1.05	1.11	1.03
vector_quantization	1.00	1.00	1.00	1.01	1.01
vectorized_newton	1.00	1.00	1.00	1.00	1.00
vehicle_routing	1.22	1.54	3.44	1.69	1.53
vertex_cover	1.39	1.62	3.43	1.68	1.31
voronoi_diagram	2.24	2.13	4.91	2.49	2.35
wasserstein_dist	4.48	8.48	9.47	7.16	4.52
water_filling	60.54	86.86	195.98	95.57	82.87
zoom_2d	1.00	1.00	1.00	1.00	1.00

Table 11: Per-task speedup on AlgoTune using ShinkaEvolve+SEDIMA (part 1 of 2).

Task	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
aes_gcm_encryption	1.13	1.07	1.19	1.06	1.02
affine_transform_2d	1.00	1.00	1.00	1.00	1.00
aircraft_wing_design	1.09	1.13	2.16	1.22	1.16
articulation_points	2.66	5.17	2.53	4.04	2.91
base64_encoding	1.07	1.00	1.11	1.00	1.00
battery_scheduling	9.85	16.16	34.92	17.21	14.44
btsp	1.21	2.62	2.41	2.11	1.36
capacitated_facility_location	5.34	10.95	14.87	8.11	2.57
chacha_encryption	1.26	1.21	1.25	1.17	1.09
channel_capacity	1.08	1.16	1.48	1.17	1.12
chebyshev_center	3.14	4.75	8.46	5.26	4.26
cholesky_factorization	1.08	1.04	1.08	1.04	1.02
clustering_outliers	1.19	1.32	1.20	1.28	1.20
communicability	26.14	59.51	159.87	67.34	57.50
convex_hull	1.45	3.94	6.03	3.56	1.79
convolve2d_full_fill	54.19	113.56	185.31	108.58	81.33
convolve_1d	1.05	1.12	1.14	1.09	1.04
correlate2d_full_fill	37.90	111.36	140.01	104.27	72.26
correlate_1d	1.06	1.09	1.27	1.09	1.05
count_connected_components	2.77	6.18	5.34	5.02	3.88
count_riemann_zeta_zeros	1.00	1.00	1.00	1.00	1.00
cumulative_simpson_1d	6.34	10.97	13.73	7.49	2.71
cumulative_simpson_multid	1.03	1.00	1.05	1.00	1.00
cvar_projection	1.44	2.79	1.58	2.28	1.82
cyclic_independent_set	1.51	11.99	2.28	5.69	1.92
dct_type_I_scipy_fftpack	1.07	1.08	1.50	1.12	1.07
delaunay	1.97	3.59	2.02	2.70	1.50
dijkstra_from_indices	1.00	1.00	1.00	1.00	1.00
discrete_log	1.04	1.40	1.11	1.27	1.08
dst_type_II_scipy_fftpack	1.34	1.71	1.33	1.50	1.20
dynamic_assortment_planning	22.07	36.89	49.50	24.26	5.69
earth_movers_distance	1.01	1.01	1.01	1.01	1.00
edge_expansion	4.36	15.20	3.32	7.84	2.46
eigenvalues_complex	1.37	1.79	2.28	1.77	1.55
eigenvalues_real	1.98	3.11	4.05	3.04	2.59
eigenvectors_complex	1.04	1.06	1.06	1.05	1.02
eigenvectors_real	1.02	1.04	1.13	1.05	1.04
elementwise_integration	1.00	1.00	1.00	1.00	1.00
feedback_controller_design	23.81	118.57	83.39	61.71	6.82
fft_cmplx_scipy_fftpack	1.76	2.94	2.65	2.61	2.33
fft_convolution	1.00	1.00	1.00	1.00	1.00
fft_real_scipy_fftpack	1.10	1.71	1.30	1.90	2.57
firls	1.00	1.00	1.00	1.01	1.01
generalized_eigenvalues_complex	2.85	5.06	4.69	4.39	3.23
generalized_eigenvalues_real	2.00	3.35	4.13	3.36	2.56
generalized_eigenvectors_complex	1.79	3.56	4.43	2.87	1.62
generalized_eigenvectors_real	1.45	2.28	4.35	2.24	2.14
graph_coloring_assign	3.71	2.78	2.58	2.59	1.71
graph_global_efficiency	3.05	11.17	20.36	11.33	9.56
graph_isomorphism	21.29	50.99	67.89	42.08	23.94
graph_laplacian	1.00	1.00	1.00	1.00	1.00
group_lasso	1.00	1.00	1.00	1.00	1.00
gzip_compression	1.13	1.11	1.12	1.10	1.06
integer_factorization	1.00	1.00	1.00	1.00	1.00
job_shop_scheduling	1.46	2.03	2.22	1.81	1.32
kalman_filter	6.66	13.52	16.43	10.01	3.12
kcenters	1.49	2.10	1.50	1.85	1.29
kd_tree	1.07	1.08	1.06	1.06	1.04
kernel_density_estimation	1.00	1.00	1.00	1.00	1.00
kmeans	6.79	14.33	23.46	13.54	11.07
ks_test_2samp	1.03	1.00	1.06	1.00	1.00
l0_pruning	1.27	1.84	3.45	2.01	2.37
l1_pruning	3.20	3.01	3.87	2.82	2.08
lasso	1.14	1.64	1.24	1.48	1.14
least_squares	1.17	2.45	1.91	2.16	1.98
linear_system_solver	1.11	1.16	1.34	1.15	1.11
lp_box	7.15	15.92	23.28	14.17	11.54
lp_centering	1.01	1.02	1.01	1.02	1.01
lp_mdp	96.46	181.89	320.19	123.88	24.08
lqr	1.09	1.37	1.60	1.28	1.11
lti_simulation	1.57	7.37	3.74	4.89	1.86
lu_factorization	1.00	1.06	1.01	1.10	1.18
lyapunov_stability	44.91	116.73	116.92	94.38	64.85
markowitz	1.01	1.00	1.02	1.00	1.00
matrix_completion	1.00	1.01	1.01	1.01	1.02
matrix_exponential	1.00	1.00	1.00	1.00	1.00
matrix_exponential_sparse	1.00	1.00	1.00	1.00	1.00

Table 12: Per-task speedup on AlgoTune using ShinkaEvolve+SEDIMA (part 2 of 2).

Task	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
matrix_multiplication	1.04	1.02	1.03	1.02	1.01
matrix_sqrt	1.02	1.02	1.02	1.02	1.01
max_clique_cpsat	5.37	13.14	13.90	11.77	6.93
max_common_subgraph	5.64	4.52	13.20	5.00	2.67
max_flow_min_cost	3.00	7.47	4.45	6.83	7.32
max_independent_set_cpsat	5.17	3.41	3.61	3.21	1.90
max_weighted_independent_set	1.25	1.20	1.72	1.23	1.13
min_dominating_set	1.09	1.91	1.23	1.75	1.52
min_weight_assignment	1.19	1.76	1.32	1.46	1.14
minimum_spanning_tree	4.16	6.52	3.39	4.32	1.84
minimum_volume_ellipsoid	3.71	20.67	20.30	12.31	3.06
multi_dim_knapsack	2.94	19.15	10.53	10.31	3.07
nmf	1.03	1.27	1.13	1.23	1.23
ode_brusselator	7.08	4.08	3.00	3.71	1.97
ode_fitzhughnagumo	1.05	1.14	1.10	1.09	1.04
ode_hires	6.00	18.36	30.37	14.97	6.09
ode_hodgkinhuxley	22.73	43.41	97.11	38.21	13.10
ode_lorenz96_nonchaotic	1.57	2.86	3.17	2.52	1.93
ode_lotkavolterra	18.70	13.30	51.79	14.89	6.37
ode_nbodyproblem	18.33	35.92	34.02	27.55	16.67
ode_seirs	28.06	7.09	46.26	12.15	13.29
ode_stiff_robertson	4.66	23.90	6.52	12.89	4.31
ode_stiff_vanderpol	29.20	50.03	6.63	24.76	5.25
odr	1.01	1.02	1.01	1.02	1.01
optimal_advertising	1.04	1.34	1.10	1.25	1.07
outer_product	1.14	1.03	1.23	1.02	1.01
pagerank	1.50	2.05	18.29	3.34	4.00
pca	2.38	4.27	4.06	3.08	1.69
pde_burgers1d	1.38	3.85	5.07	3.77	3.30
pde_heat1d	1.32	2.01	1.44	1.76	1.25
polynomial_mixed	5.30	5.84	3.23	4.27	2.16
polynomial_real	7.38	39.65	4.55	16.70	3.33
power_control	25.10	149.57	131.41	110.22	32.83
procrustes	1.38	1.30	1.30	1.27	1.14
psd_cone_projection	3.45	5.32	11.22	6.18	6.39
qp	1.46	1.98	2.79	2.02	1.74
qr_factorization	2.10	1.72	1.84	1.66	1.45
quantile_regression	1.18	1.40	2.01	1.41	1.29
queens_with_obstacles	1.77	3.31	4.31	3.20	2.11
queuing	1.07	1.04	1.08	1.05	1.02
qz_factorization	1.23	1.19	1.20	1.18	1.09
randomized_svd	2.27	3.80	2.28	2.91	1.52
rbf_interpolation	1.01	1.01	1.01	1.01	1.00
rectanglepacking	1.40	1.39	3.02	1.48	1.27
robust_kalman_filter	2.59	3.87	9.57	4.39	3.66
robust_linear_program	2.32	1.96	7.62	2.40	1.82
rocket_landing_optimization	1.08	1.64	1.24	1.42	1.13
rotate_2d	1.00	1.00	1.00	1.00	1.00
set_cover	4.56	6.71	4.84	4.93	2.13
set_cover_conflicts	2.12	2.83	3.48	2.75	2.34
sha256_hashing	1.00	1.00	1.00	1.00	1.00
shift_2d	1.00	1.00	1.00	1.00	1.00
shortest_path_dijkstra	1.76	2.70	4.21	2.43	1.53
sinkhorn	1.43	2.09	3.29	1.98	1.42
sparse_eigenvectors_complex	1.00	1.00	1.00	1.00	1.00
sparse_lowest_eigenvalues_posdef	1.57	2.33	2.78	2.19	1.87
sparse_lowest_eigenvectors_posdef	1.59	2.76	3.06	2.44	1.79
sparse_pca	2.95	7.61	8.47	5.75	2.91
spectral_clustering	4.34	8.79	16.33	7.12	2.49
stable_matching	1.45	1.89	2.55	1.91	1.64
svd	1.23	1.20	1.19	1.19	1.09
svm	1.00	1.00	1.00	1.00	1.00
sylvester_solver	1.02	1.01	1.02	1.01	1.01
tensor_completion_3d	16.11	24.58	43.48	20.36	6.89
toeplitz_solver	1.00	1.00	1.00	1.00	1.00
tsp	1.19	1.42	1.31	1.33	1.21
two_eigenvalues_around_0	1.59	2.08	2.99	2.14	1.79
unit_simplex_projection	1.87	3.58	1.84	2.58	1.45
upfirdn1d	1.02	1.17	1.05	1.12	1.03
vector_quantization	1.00	1.00	1.00	1.01	1.01
vectorized_newton	1.00	1.00	1.00	1.00	1.00
vehicle_routing	1.26	1.59	3.80	1.79	1.61
vertex_cover	1.42	1.67	3.80	1.75	1.33
voronoi_diagram	2.49	2.27	4.97	2.55	2.43
wasserstein_dist	5.09	9.81	10.77	7.37	4.74
water_filling	69.41	97.82	203.62	109.59	95.36
zoom_2d	1.00	1.00	1.00	1.00	1.00
Harmonic mean	1.60	1.89	1.94	1.84	1.59

Table 13: Per-task performance on ALE-Bench LITE using OpenEvolve+SEDIMA.

Problem	Format	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
ahc008	Long	1206.4	660.1	830.8	777.5	258.5
ahc011	Long	1767.2	1177.8	1287.5	1388.6	584.1
ahc015	Short	1508.7	870.8	1154.4	1061.0	411.7
ahc016	Long	1306.9	743.3	1055.4	847.9	295.2
ahc024	Short	1900.3	1313.3	1358.2	1491.4	715.2
ahc025	Long	1495.7	885.2	1166.4	1032.3	369.6
ahc026	Short	1866.1	1123.4	1309.4	1432.3	602.9
ahc027	Long	2172.8	1586.6	1580.9	1853.7	806.1
ahc039	Short	1989.1	1348.7	1421.7	1485.4	715.6
ahc046	Short	1885.8	1168.8	1431.3	1379.9	597.1

Table 14: Per-task performance on ALE-Bench LITE using ShinkaEvolve+SEDIMA.

Problem	Format	GPT-5.4	DeepSeek V4 Pro	Gemini 3 Pro	Qwen 3.7 Max	Qwen 3 Coder
ahc008	Long	1254.7	683.3	851.7	793.5	277.9
ahc011	Long	1801.8	1219.1	1306.8	1445.6	609.8
ahc015	Short	1537.5	915.2	1169.1	1095.7	434.3
ahc016	Long	1359.2	769.4	1071.1	865.4	314.3
ahc024	Short	1936.7	1353.2	1389.2	1525.0	739.7
ahc025	Long	1525.1	934.7	1183.9	1053.5	385.8
ahc026	Short	1901.8	1157.6	1352.6	1464.5	623.5
ahc027	Long	2171.2	1609.5	1604.5	1873.0	816.2
ahc039	Short	2027.1	1389.7	1439.7	1518.8	740.0
ahc046	Short	1921.9	1204.3	1449.4	1411.0	617.5

Prompt G.3: Cluster Summary Generation

You summarize a cluster of semantically related local optimization insights.
Produce a concise practical summary that captures the shared optimization pattern, its recurring effect, and the general lesson represented by the cluster.

Requirements

- Write 2–3 concise sentences.
- Generalize across the provided member insights rather than repeating them individually.
- Preserve important distinctions when the reported effects differ.
- Do not introduce claims that are unsupported by the member insights.

Cluster member insights

{{member_insights}}

Return only the cluster summary without additional explanation.

Prompt G.4: Recommendation Synthesis

You synthesize actionable guidance from retrieved local insights and their associated raw evidence. Given the current problem and retrieved evidence, produce concrete recommendations for the next mutation attempt.

Requirements

- Return 3 concise and actionable recommendations.
- Ground every recommendation in the retrieved insights or linked raw evidence.
- Focus on changes that can guide the next code mutation.
- Avoid unsupported claims, speculative conclusions, and irrelevant advice.
- Do not merely repeat the retrieved insights; translate them into practical next steps.

Current query or problem

{{query_text}}

Retrieved evidence

Each retrieved item may contain an insight, its cluster-level similarity, its insight-level similarity, and optional linked raw evidence.

{{retrieved_evidence}}

Return only the numbered or bulleted recommendations without additional explanation.