OPTICOM: A UNIFIED FRAMEWORK FOR STATE-CONDITIONED COMPOSITION IN LLM-DRIVEN OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly deployed to solve complex scientific and practical problems via iterative optimization. However, dynamically coordinating diverse search mechanisms as candidate quality, failure modes, and resource budgets evolve remains a critical open challenge. Targeted empirical diagnostics reveal that mechanism effectiveness is highly state-dependent. Motivated by this, we analyze how individual decisions drive final outcomes, decomposing the expected terminal improvement under a shared budget into cumulative decision opportunities minus cumulative selection losses. Guided by this opportunity-loss theoretical foundation, we propose OptiCom, a unified framework that represents LLM-driven optimizers within a shared configuration space: $C = (A, Q, O, E, M, S)$, corresponding to artifact, query, operator, evaluation, memory, and strategy. Operating within this space, a fast LLM-based Optimization Controller dynamically composes immediate mechanisms through structured Action Packages, while a slower Strategy Adapter refines long-term selection preferences, operator weights, and templates based on accumulated trajectory feedback. Comprehensive evaluations across 32 benchmark groups demonstrate the superiority of framework: OptiCom achieves an average Max-score rank of 1.72 among 14 evaluated configurations, securing the top score in 23 groups. Ultimately, these results highlight the broad applicability and high extensibility of OptiCom as a general-purpose paradigm for robust LLM test-time scaling.

1 INTRODUCTION

Large language models (LLMs) are increasingly used to improve solutions to scientific and practical problems through iterative generation and evaluation Snell et al. (2025); Liu et al. (2026c). Applications span mathematical discovery Romera-Paredes et al. (2023); Tsoukalas et al. (2026), computational efficiency Ouyang et al. (2025), and image generation Zhang et al. (2025b). These settings impose concrete requirements: a mathematical construction must remain feasible while improving a target bound Romera-Paredes et al. (2023); Wei et al. (2026b); a GPU kernel must preserve correctness while reducing execution time Ouyang et al. (2025); and a generated image must satisfy semantic and visual requirements Zhang et al. (2025b). Challenging reasoning benchmarks likewise distinguish plausible outputs from solutions that satisfy the task Chollet et al. (2026); Foundation (2026). Despite differences in representation and evaluation, these diverse problems require searching for better candidates under constraints and limited computational resources Wu et al. (2025).

Existing approaches provide complementary mechanisms for this search Zhang et al. (2026); ang Gao et al. (2026). OPRO conditions candidate generation on evaluated solutions and their scores Yang et al. (2024), while TextGrad propagates natural-language feedback to update variables in computational graphs Yuksekgonul et al. (2025). FunSearch and AlphaEvolve combine program generation with evolutionary search and automated evaluation Romera-Paredes et al. (2023); Novikov et al. (2025), alongside broader progress in algorithm discovery Mankowitz et al. (2023); Liu et al. (2024); Zheng et al. (2025); Ye et al. (2024). Reflexion and ExpeL reuse experience to guide later attempts Shinn et al. (2023); Zhao et al. (2024), and GEPA combines reflection with complementary candidate reuse Agrawal et al. (2025). These methods illustrate different ways to organize proposals, feedback, and experience into an improvement loop Wei et al. (2026a).

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.

From an optimization perspective, this improvement loop involves a sequence of interconnected decisions: what information to acquire, which candidates to revise, how to modify them, how thoroughly to evaluate them, and what experience to retain. Crucially, the utility of these mechanisms shifts dynamically during optimization. An invalid candidate may require targeted repair; a feasible but stagnant solution might benefit from recombination or broader exploration; and a high-quality candidate often warrants a precise, structure-preserving local update. This perspective connects LLM-driven optimization to classical studies of problem-dependent search performance and hyper-heuristics Wolpert & Macready (1997); Burke et al. (2013). It motivates treating the decisions that govern candidate improvement as objects of optimization themselves, raising a central question: *How can we represent the decisions made by different optimizers in a common space, and adaptively compose them as the task, optimization state, and remaining budget evolve?*

Recent progress in shared abstractions and optimizer adaptation provides a foundation for this view Khattab et al. (2024); Cemri et al. (2026); Liu et al. (2026b); Xu et al. (2024); Zhang et al. (2025a). Building on these directions, we seek a common representation of optimization responsibilities and principles for coordinating them. Our diagnostic study provides empirical evidence that mechanism preferences vary across the constructed optimization states (Section 3). To rigorously understand how these dynamic decisions affect the complete optimization process, we analyze general policies under a shared budget and terminal evaluation criterion (Section 4). We decompose the expected terminal improvement into *cumulative decision opportunities* (Δ_t) minus *cumulative selection losses* (ℓ_t). Under a near-optimal selection assumption, we derive a sufficient condition for positive terminal improvement. This analysis accounts for exploration costs and unfavorable decisions through their cumulative effect, yielding two complementary design principles: make valuable alternatives available, and improve their selection using the current state and accumulated evidence.

Guided by this theoretical perspective, we introduce OptiCom, a unified framework that represents optimizers as configurations $C = (A, Q, O, E, M, S)$ and adapts their component compositions. An Optimization Controller dynamically selects Action Packages based on the current state, while a slower Strategy Adapter refines selection preferences over time. Across 32 benchmark groups, OptiCom obtains the lowest average rank (1.72 for Max, 2.08 for Mean) among 14 evaluated configurations, supported by additional checks using official SkyDiscover implementations on seven benchmarks. Our contributions are fourfold:

- **Shared representation.** We define optimization decisions across six functional dimensions (AQOEMS), enabling disparate existing methods and complementary mechanisms to interact within a common, extensible space (Section 2).
- **Theoretical analysis.** We formalize terminal improvement through the lens of cumulative opportunities and selection losses, yielding theoretically grounded principles for adaptive optimization (Section 4).
- **Adaptive framework.** We instantiate these principles via OptiCom, featuring state-conditioned Action Package composition, a task-specific execution harness, and two-timescale strategy adaptation (Section 5).
- **Empirical validation.** Through cross-domain evaluations, state-conditioned motivation studies, targeted ablations, and token-aware case studies, we evaluate final quality and examine the contributions of adaptation, experience reuse, and continued search expenditure (Section 6).

2 A Shared Configuration Space for LLM-Driven Optimizers

2.1 Optimization Concepts and the AQOEMS Representation

Formally, given a task d, an optimizer explores a candidate space $\mathcal{X}_d$ under a computational budget B to find solutions for:

$$\max_{x \in \mathcal{X}_d} f_d(x) \qquad \text{subject to } x \in \mathcal{F}_d, \tag{1}$$

where $\mathcal{F}_d$ and f_d denote the feasible set and the objective function, respectively. We conceptualize the underlying mechanisms and coordinating rules of any such optimizer as a unified configuration tuple:

$$C = (A, Q, O, E, M, S). \tag{2}$$

Table 1: Representative AQOEMS mappings across three research directions. Each method has a framework instantiation; mapping and alignment details appear in Appendix A.

	Feedback-guided revision and experience reuse	**Evolutionary search and adaptive control**	**Shared abstractions and meta-optimization**
	TextGrad Yuksekgonul et al. (2025)	AdaEvolve Cemri et al. (2026)	DSPy Khattab et al. (2024)
A	Optimizable graph variables	Candidate programs	LM program instructions and demonstrations
Q	Graph context and propagated feedback	Selected programs and search evidence	Training examples and execution traces
O	Textual-feedback-guided updates	LLM-generated program variations	Instruction and demonstration updates
E	Objective feedback	Program fitness	User-specified task metric
M	Graph and update state	Populations and improvement statistics	Optimizer-dependent candidates and search state
S	Feedback propagation and update rules	Adaptive exploration, allocation, and guidance	Selected optimizer's compilation and search rules

Here, A defines the artifact representation; Q dictates information acquisition and context construction; O governs candidate generation and updating operators; E specifies quality and feasibility evaluation; M manages retained experience and candidate archives; and S provides the overarching strategy for search coordination and resource allocation.

Crucially, AQOEMS serves as a functional taxonomy rather than a strict architectural blueprint for six isolated modules or sequential steps. Components may share implementations or be entirely omitted (taking null forms). The instantiation of these functions dynamically adapts to the current optimization state—such as candidate quality, encountered failures, historical experience, and remaining budget. For instance, while E assesses the candidate and generates feedback, Q selectively routes this evidence for future inspection. Consequently, even a visually "fixed" configuration C can encapsulate state-dependent or adaptive rules within its strategy S, meaning it need not execute the identical action at every iteration. Further details on component responsibilities and boundaries are provided in Appendix A.3.

2.2 Existing Optimizers through the AQOEMS Lens

AQOEMS naturally captures candidate–feedback loops across diverse optimization families, encompassing stateful, stochastic, and internally adaptive mechanisms. We organize related work into three overlapping directions, with one implemented representative from each detailed in Table 1.

Feedback-guided revision and experience reuse. TextGrad connects evaluation (E) to variable updates (O) through graph-specific context (Q) Yuksekgonul et al. (2025). OPRO uses scored candidate history Yang et al. (2024), Reflexion retains verbal reflections Shinn et al. (2023), and ExpeL extracts reusable knowledge Zhao et al. (2024). These methods exemplify distinct strategies for acquiring and applying evidence via Q, M, and O.

Evolutionary search and adaptive control. AdaEvolve uses improvement statistics (M) to adapt exploration and allocation (S) Cemri et al. (2026). FunSearch and AlphaEvolve combine program generation, automated evaluation, and population-based search Romera-Paredes et al. (2023); Novikov et al. (2025); GEPA adds reflective updates and complementary candidate reuse Agrawal et al. (2025); and EvoX evolves search procedures Liu et al. (2026b). These frameworks showcase rich interactions among O, E, M, and an internally adaptive S.

Shared abstractions and meta-optimization. DSPy optimizes LM programs against user-specified metrics, with concrete operations and search rules supplied by the selected optimizer Khattab et al. (2024). Trace uses execution traces and feedback for generative optimization Cheng et al. (2024), while metaTextGrad improves optimizer prompts and compositions Xu et al. (2024).

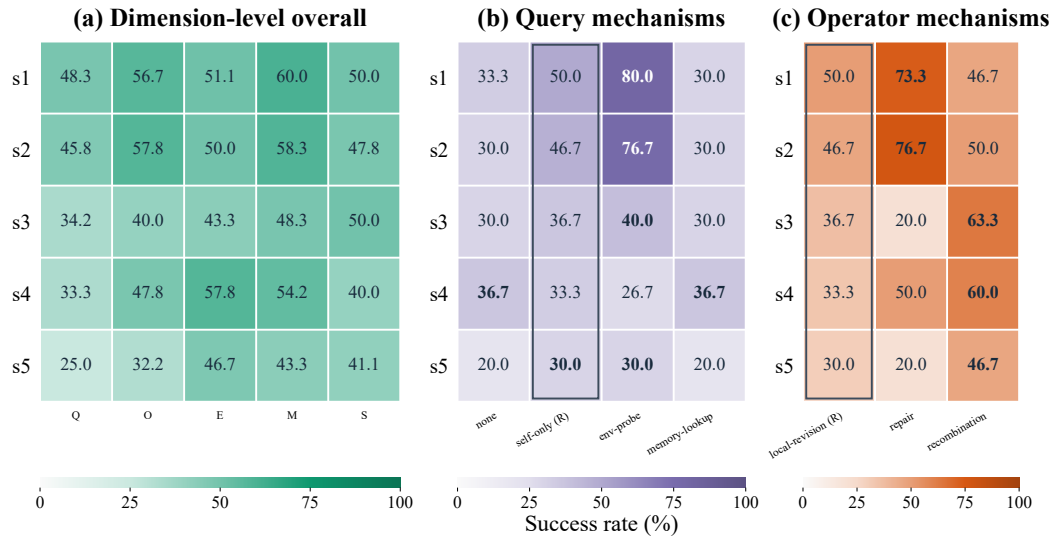

Figure 1: State-conditioned diagnostics on Heilbronn Triangle: three distinct situations per state and ten trials per mechanism–situation pair (30 trials per mechanism–state rate). **(a)** Averages over Q/O/E/M/S alternatives; **(b)** query and **(c)** operator comparisons. R marks the reference setting. Memory comparisons enable retrieval; other dimensions change one mechanism from the reference. Appendix B details the protocol and complete six-panel results.

Unlike existing universal frameworks that enforce static search pipelines or monolithic optimization APIs, OptiCom dynamically reshapes its underlying mechanism composition during execution to adapt to the evolving optimization state, while offering superior extensibility to integrate novel mechanisms. Methods enter the framework through these extensible candidate, feedback, memory, and action interfaces. We defer detailed discussions of existing larger framework compositions (e.g., LLM4AD Liu et al. (2026a), SkyDiscover Liu et al. (2026c), and optimize_anything Agrawal et al. (2026)) to Appendix A.1, and details regarding the scope of our contribution to Appendix A.2.

3 Motivating State-Conditioned Optimization

We examine state-dependent mechanism preferences on Heilbronn Triangle using 15 starting situations across five state families: s1 (execution/interface failures), s2 (failed task-specific checks), s3 (objective stagnation), s4 (limited remaining budget), and s5 (high performance variability). The strict intervention matrix changes one mechanism from the reference configuration ($Q2$ self-only + $O1$ local revision + $E2$ textual feedback + $M1$ no persistent memory + $S1$ fixed strategy). Memory comparisons additionally enable Q4 retrieval so that stored experience can affect later decisions; these are conditional comparisons rather than single-coordinate interventions. Each mechanism–situation pair contributes ten trials, with success defined by resolution of the starting condition (Appendix B). Figure 1 shows changes in relative mechanism effectiveness. Environment probing exceeds self-only querying in s1 and s2 (80.0% and 76.7% versus 50.0% and 46.7%), but has a lower observed rate in s4 (26.7% versus 33.3%). Repair reaches 73.3% and 76.7% in s1 and s2, yet falls to 20.0% under stagnation, where recombination reaches 63.3%. Across the s2 situations, joint Q+O modification reaches 83.3%, compared with 76.7% for either single change and 46.7% for the reference. Conditional mean iterations among successful trials are 2.1 for Q+O versus 2.9, 3.6, and 5.2 for Q-only, O-only, and the reference, respectively (Appendix B.5). These descriptive results motivate state-conditioned selection and complementary composition; they do not establish universal preferences or a general interaction effect.

4 Theoretical Analysis of Adaptive Optimization

The motivation study illustrates how different decisions may be useful in different states. We now ask how those decisions affect the final outcome of an optimization run. Our analysis applies to a

general optimization policy π, not specifically to OptiCom. It first identifies the common structure of terminal improvement and then bounds the effect of imperfect selection. Let $J_B(\pi)$ be expected terminal quality under budget B, with terminal utility in $[0, 1]$. At history h_t, let $V^{\pi_0}(h_t)$ be the expected terminal quality obtained by continuing with a reference policy π_0, and let $q(h_t, a)$ be the value of executing a before returning to π_0. Define

$$\begin{aligned} q^\star(h_t) &= \max_{a \in \mathcal{A}(h_t)} q(h_t, a), \\ \Delta_t &= q^\star(h_t) - V^{\pi_0}(h_t), \\ \ell_t &= q^\star(h_t) - \mathbb{E}_\pi[q(h_t, A_t) \mid h_t]. \end{aligned} \tag{3}$$

Here, Δ_t is the best available one-decision improvement relative to reference continuation, while ℓ_t is the expected loss in selecting among those actions. The complete process and reference continuation are specified in Appendix C.1.

4.1 Cumulative Opportunity–Loss Decomposition

Our first result specializes the performance-difference identity Schulman et al. (2017) to this budgeted process. It characterizes the complete run without assuming that individual decisions improve the incumbent. The proof is given in Appendix C.2.

Theorem 1 (Cumulative Opportunity–Loss Decomposition)**.** *Under the shared budgeted process,*

$$J_B(\pi) - J_B(\pi_0) = \mathbb{E}_\pi \left[\sum_{t<T} (\Delta_t - \ell_t) \right]. \tag{4}$$

Remark. A larger opportunity Δ_t benefits performance only when selection realizes the additional value. At a fixed history, expanding $\mathcal{A}(h_t)$ while preserving existing action values and costs cannot decrease $q^\star(h_t)$. However, if the selected-action distribution remains unchanged, any increase in $q^\star(h_t)$ raises Δ_t and ℓ_t equally, leaving $\Delta_t - \ell_t$ unchanged. Thus, catalog expansion should be paired with selection that exploits the added choices.

4.2 Selection Quality and Global Improvement

Following the use of explicit model-capability assumptions in LLM selection theory Chen et al. (2025), suppose that, at each reached history,

$$\Pr_\pi(q^\star(h_t) - q(h_t, A_t) \le \epsilon_t \mid h_t) \ge 1 - \delta_t, \qquad 0 \le \epsilon_t, \delta_t \le 1. \tag{5}$$

Since terminal utility lies in $[0, 1]$, the action-value gap is at most one, giving $\ell_t \le (1 - \delta_t)\epsilon_t + \delta_t$. Combining this bound with Theorem 1 gives a lower bound on terminal improvement, proved in Appendix C.3. Detailed theoretical remarks and discussions on the applicability of these assumptions to our empirical implementations are provided in Appendix C.4.

Theorem 2 (Global Performance Bound)**.** *Under the conditional selection assumption and bounded terminal utility,*

$$J_B(\pi) - J_B(\pi_0) \ge \mathbb{E}_\pi \sum_{t<T} \left[\Delta_t - (1 - \delta_t)\epsilon_t - \delta_t \right]. \tag{6}$$

Remark. For fixed δ_t, reducing ϵ_t leaves a selection-loss bound approaching δ_t. Thus, refining near-optimal choices alone may be insufficient when the probability of missing them remains high. This motivates addressing recurring poor selections alongside improving near-optimal choices; the appropriate allocation of effort depends on their attainable improvements and costs.

Design implications. Together, the results motivate two complementary design principles: expose valuable alternatives to create opportunities (Δ_t), and use diagnostic feedback alongside accumulated experience to improve selection (ℓ_t). Allocating decision effort according to the current state aims to realize these opportunities while accounting for selection errors and resource costs. Section 5 instantiates these theory-motivated principles in OptiCom.

5 THE OPTICOM FRAMEWORK

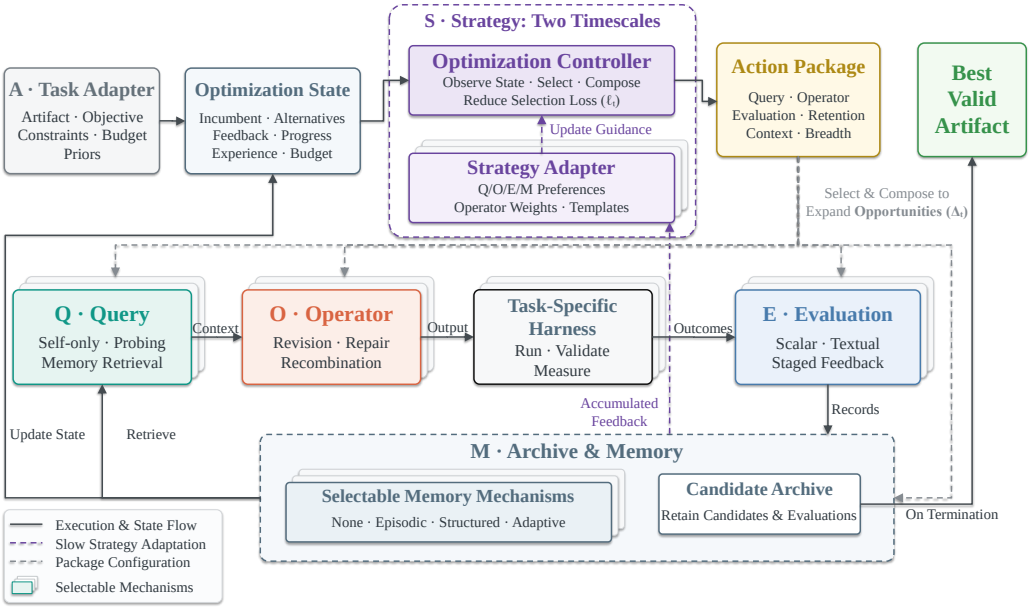

Figure 2: Overview of OptiCom. The Optimization Controller selects and composes Q/O/E/M mechanisms through an Action Package, while the slower Strategy Adapter updates the guidance used by subsequent Controller decisions. Solid arrows show execution and state flow; gray dashed arrows indicate configuration, and purple dashed arrows indicate slower strategy adaptation. Annotations involving Δ_t and ℓ_t highlight how specific architectural components are explicitly designed to expand search opportunities (Δ_t) and reduce selection losses (ℓ_t).

Composing optimization mechanisms. The analysis in Section 4 identifies two factors governing adaptive optimization: the opportunities provided by available actions, Δ_t, and the loss incurred when selecting among them, ℓ_t. Guided by this decomposition, OptiCom dynamically combines complementary mechanisms within the shared configuration space. To maximize opportunities while mitigating selection loss, the framework operates through a continuous, state-conditioned execution loop driven by two core components: an Optimization Controller for immediate mechanism composition, and a slower Strategy Adapter for long-term guidance revision. Both components operate through LLM API calls with fixed model parameters (see Algorithm 1 in Appendix D.1).

State-conditioned execution and evaluation. At each iteration, the Optimization Controller observes the current optimization state—the incumbent, available alternatives, recent feedback, progress history, and remaining budget. It produces a structured Action Package specifying query and context scope, the update operator, generation breadth, evaluation effort, and retention policy. The framework acquires the requested evidence, generates candidates, and submits them to a task-specific Execution Harness. The harness runs the artifacts and returns the checks and measurements provided by the adopted benchmark pipeline (see Appendix E.1 for detailed benchmark configurations and pipeline specifications).

Adapting strategy from accumulated experience. Following evaluation, candidates and outcomes are retained according to the selected archive and memory policies. The Strategy Adapter reviews accumulated successes, failures, and progress on a slower timescale, revising mechanism-selection preferences and prompt templates for subsequent Controller decisions. For example, recurring runtime errors may motivate greater emphasis on diagnostic queries. These updates change optimization guidance without modifying model weights or resetting the archive. The design uses accumulated evidence to inform later decisions; it does not estimate the theoretical continuation values or guarantee that every exploratory step will yield a long-term benefit. Complete component definitions, Action Package fields, and harness details appear in Appendix D.

Table 2: Main results across seven representative benchmarks over five runs. Unqualified method names denote method-inspired framework profiles; "official" denotes the official SkyDiscover implementation. Scores follow task-specific scales. Higher scores are better. Bold and underlined values indicate the best and second-best distinct results.

Domain	Math		Systems		GPU		Algorithms		Reasoning		Prompts		Quantum	
Benchmark	Heilbronn		LLM-SQL		TriMul		Frontier-CS		ARC		HotpotQA		QNN	
Method	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean
ProTeGi	0.828	0.813	0.630	0.622	2.541	2.531	<u>0.632</u>	<u>0.619</u>	<u>0.473</u>	<u>0.461</u>	<u>0.441</u>	<u>0.427</u>	<u>0.8833</u>	0.8367
LATS	0.887	0.867	0.686	<u>0.682</u>	2.576	2.542	0.627	0.609	0.457	0.432	0.413	0.409	0.8500	0.8300
EvoX	0.943	0.931	0.684	0.677	<u>2.593</u>	<u>2.583</u>	0.626	0.617	0.461	0.443	0.386	0.357	0.8167	0.8067
EvoX (official)	<u>0.956</u>	<u>0.941</u>	<u>0.696</u>	0.681	2.592	2.579	0.613	0.597	<u>0.473</u>	0.451	0.397	0.363	0.8333	0.8133
AdaEvolve	0.769	0.744	0.619	0.613	2.573	2.569	0.621	<u>0.619</u>	0.413	0.406	0.397	0.339	0.8333	0.8200
AdaEvolve (official)	0.769	0.744	0.619	0.613	2.573	2.569	0.619	0.616	0.423	0.409	0.409	0.359	0.8500	<u>0.8400</u>
GEPA	0.719	0.701	0.659	0.651	2.437	2.421	0.613	0.607	0.396	0.371	0.381	0.313	0.8000	0.7900
OptiCom	**0.961**	**0.953**	**0.698**	**0.696**	**2.654**	**2.616**	**0.693**	**0.683**	**0.503**	**0.491**	**0.513**	**0.501**	**0.9000**	**0.8967**
Relative gain	+0.52%	+1.28%	+0.29%	+2.05%	+2.35%	+1.28%	+9.65%	+10.34%	+6.34%	+6.51%	+16.33%	+17.33%	+1.89%	+6.75%

6 EXPERIMENTS

6.1 EXPERIMENTAL SETUP

We evaluate OptiCom across 32 benchmark groups spanning mathematics, systems, GPU kernels, algorithms, reasoning, creative generation, prompts, and quantum circuits, integrated from SkyDiscover Liu et al. (2026c). The overall comparison comprises OptiCom and 13 method-inspired profiles (e.g., LATS, TextGrad, ProTeGi) executed within our shared framework to ensure evaluation consistency. Table 2 additionally incorporates official SkyDiscover implementations of EvoX and AdaEvolve as external implementation comparisons. All main comparisons use Doubao-Seed-2.0-pro as the optimization backbone, capped at 30 iterations across five independent runs. We report both Mean and Max final scores. A common iteration cap does not equalize candidate counts, API calls, tokens, or elapsed time. Comprehensive details on scoring, aggregation rules, and model budgets are provided in Appendix E.

6.2 MAIN RESULTS

Overall performance. Figure 3(a) shows that OptiCom obtains the lowest average rank among the 14 evaluated configurations across 32 benchmark groups: 1.72 for five-run Max and 2.08 for five-run Mean, compared with 6.33 and 6.35 for the strongest competing profile, EvoX. Under Max, OptiCom attains the highest score on 23 groups, including two ties. Its leading position under both summaries indicates that the aggregate advantage is not confined to best-of-five performance. Table 2 reports the highest Max and Mean for OptiCom on all seven representative benchmarks, including comparisons with the two official-code entries. Relative gains over the strongest baseline in each column range from 0.29% to 16.33% for Max and from 1.28% to 17.33% for Mean. These results concern the evaluated configurations under the reported iteration limit; the official-code comparisons cover two methods on seven benchmarks.

Optimization progress. Figure 3(b) illustrates how improvements accumulate during a run. On Heilbronn Triangle, OptiCom reaches approximately 0.774 at iteration 2, makes limited progress through iteration 8, and then improves to 0.961 at iteration 13. This score exceeds the final scores of the displayed baseline profiles and remains unchanged through iteration 30. The trajectory shows that a temporary plateau need not indicate exhausted improvement opportunities. Additional trajectories reveal task-dependent refinement horizons and cases where specialized profiles finish higher (Appendix F.2). These observations are consistent with the cumulative perspective in Section 4, but score histories alone do not identify the effects of individual mechanism changes or establish equal-cost efficiency.

6.3 ABLATION STUDIES

Table 3 examines the roles of immediate selection, slower adaptation, and experience reuse. Removing the Strategy Adapter while retaining the state-conditioned Controller lowers mean scores

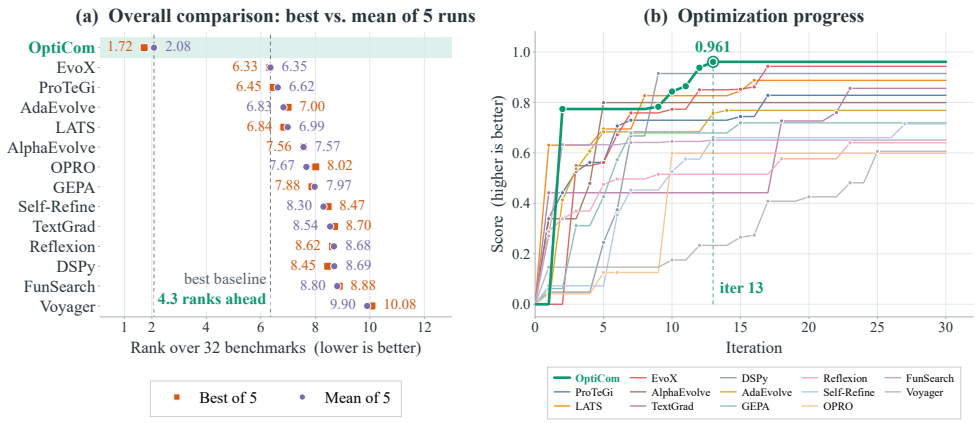

Figure 3: **(a)** Average within-group ranks of OptiCom and 13 method-inspired profiles across 32 benchmark groups, computed separately from five-run Max and Mean scores (lower is better). **(b)** Archived best-so-far scores over optimization iterations on Heilbronn Triangle.

by 0.240 on Heilbronn Triangle and 0.050 on HotpotQA. With the Adapter disabled in both variants, allowing the Controller to adjust Q/E/M and other action settings in addition to O improves mean scores over Operator-only adaptation by 0.016 and 0.023, respectively. Removing experience memory lowers the full framework's means by 0.060 and 0.018. Full OptiCom has the highest observed mean and lowest standard deviation among these core variants. Together, the comparisons support the benefits of broader action selection, strategy adaptation, and experience reuse in the tested settings. Full versus Fixed or Random composition changes both immediate selection and slower adaptation, so those contrasts do not isolate the Controller.

Adaptation frequency and initialization robustness. On Heilbronn Triangle, event-triggered adaptation reaches a mean score of 0.953 with 10 Adapter calls, compared with 0.954 and 29 calls for every-iteration adaptation (Appendix F.4). This is a 65.5% reduction in Adapter calls, while total API tokens decrease only from 877K to 865K (approximately 1.37%). The result supports similar observed quality with fewer adaptation calls, rather than a substantial reduction in total computation. Across three initial compositions, adaptive mean scores range from 0.951 to 0.956, whereas fixed-composition means range from 0.813 to 0.857. This smaller range suggests reduced sensitivity to the tested initial choices; it does not establish an optimality bound or statistical equivalence across initializations.

Table 3: Core ablations with Doubao-Seed-2.0-pro. Scores report mean ± std over 5 runs.

Config.	Heilbronn	HotpotQA
Full OptiCom	$\mathbf{0.953 \pm 0.008}$	$\mathbf{0.501 \pm 0.017}$
Fixed composition	0.704 ± 0.034	0.409 ± 0.127
Random composition	0.672 ± 0.171	0.381 ± 0.153
w/o Strat. Adapter	0.713 ± 0.027	0.451 ± 0.058
Operator-only adapt.	0.697 ± 0.056	0.428 ± 0.076
w/o Exp. Memory	0.893 ± 0.024	0.483 ± 0.039

Cross-backbone robustness and model elevation. Table 4 evaluates five diverse foundational models under identical optimization constraints. The results establish the universal applicability of OptiCom: across every evaluated backbone–task pair, enabling the Strategy Adapter yields substantial mean-score improvements and strictly compresses variance compared to the static configuration. This consistent delta yields two critical insights. First, the framework's adaptive guidance is fundamentally robust across different model alignments and pre-training distributions; it is not overfitted to the idiosyncrasies of a single LLM. Second, the Adapter operates simultaneously as a performance "floor raiser" for standard models (e.g., driving a remarkable +0.240 gain for Doubao on Heilbronn) and a "ceiling breaker" for frontier models. For instance, equipping GPT-5.5 with Full OptiCom pushes its Heilbronn performance from a static 0.796 to a near-optimal 0.987, while collapsing the standard deviation to a mere ± 0.002. Ultimately, this demonstrates that dynamic strategy adaptation structurally stabilizes the search process, acting as an indispensable performance multiplier regardless of the underlying model's raw capabilities.

6.4 Token-Aware Optimization Case Study

Figure 4 details a token-aware Heilbronn Triangle trajectory, exposing the stark nonlinearities of test-time compute scaling. After finding an initial feasible solution (0.774) at iteration 2, the optimizer

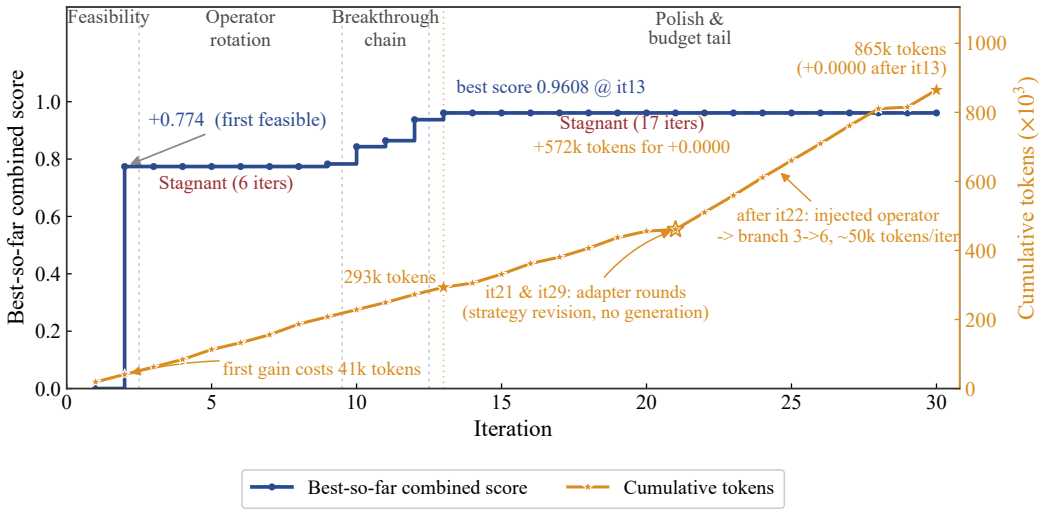

Figure 4: Heilbronn Triangle case study: best-so-far combined score (blue) and recorded cumulative API tokens (orange). The run reaches 0.9608 at iteration 13 using ≈293k tokens, then spends another 572k through iteration 30 without improvement. Selected execution events are annotated; their causal effects and the impossibility of further improvement are not established.

Table 4: Cross-backbone evaluation: mean ± SD over five runs (at most 30 iterations). Gain denotes improvement from enabling the Adapter. Bold and underlining mark the best and second-best distinct values per column, separately for means (↑), SDs (↓), and gains (↑).

	Heilbronn Triangle			**HotpotQA**		
Backbone	w/o Adapter	Full OptiCom	Gain	w/o Adapter	Full OptiCom	Gain
Doubao-Seed-2.0-pro	0.713 ± 0.027	0.953 ± 0.008	$+\mathbf{0.240}$	0.451 ± 0.058	0.501 ± 0.017	$+0.050$
GPT-5.5	$\mathbf{0.796 \pm 0.011}$	$\mathbf{0.987 \pm 0.002}$	$\underline{+0.191}$	$\mathbf{0.471 \pm 0.037}$	$\underline{0.553} \pm \underline{0.012}$	$\underline{+0.082}$
Claude Opus 4.6	$0.787 \pm \underline{0.012}$	0.973 ± 0.007	$+0.186$	$\underline{0.469} \pm \underline{0.041}$	$\mathbf{0.554 \pm 0.010}$	$+\mathbf{0.085}$
GLM-5.3	0.783 ± 0.014	0.969 ± 0.008	$+0.186$	0.461 ± 0.059	0.523 ± 0.019	$+0.062$
Kimi-K3	$\underline{0.791} \pm \mathbf{0.011}$	$\underline{0.974} \pm \underline{0.004}$	$+0.183$	0.465 ± 0.053	0.547 ± 0.016	$\underline{+0.082}$

endures a six-iteration plateau before surging to a peak score of 0.9608 at iteration 13. This sequence yields a crucial insight: *early stagnation is often deceptive*. Relying on static, low-resource operators based on lack of immediate progress would forfeit significant delayed returns, underscoring the necessity of dynamically triggered exploration to break plateaus.

However, the post-peak phase reveals the severe marginal cost of unguided scaling. From iteration 13 onward, the score flatlines while cumulative token consumption explodes from roughly 293k to 865k. Driven by Adapter-triggered exploration efforts (e.g., doubling generation branch width after iteration 22), per-iteration costs soar to ∼50k tokens without yielding better candidates. This exposes a fundamental asymmetry in LLM optimization: while broadening search is essential to shatter early local optima, maintaining aggressive exploration at the performance frontier risks massive computational waste. This token-aware perspective directly validates OptiCom's core premise: rather than deploying static, resource-heavy mechanisms throughout an entire run, an optimizer must dynamically modulate its execution strategy to balance breakthrough opportunities (Δ_t) against their cumulative evaluation and selection costs (ℓ_t).

7 Conclusions and Limitations

We introduced OptiCom, a shared AQOEMS representation and framework for state-conditioned composition of optimization mechanisms. Evaluations across 32 benchmark groups and targeted ablations support its effectiveness under the tested iteration limits. The token-aware case study highlights the cost of continued search without further improvement, while task-specific evaluation limitations and incomplete run-level records constrain the conclusions.

AI USE STATEMENT

We have not used generative AI tools for any of the tasks whose disclosure is required—proposing or refining hypotheses, developing conceptual frameworks, designing methodology or experiments, implementing methods, processing or cleaning data, analysing or interpreting results, formulating or proving mathematical claims, or translation—all of which were carried out entirely by the authors. We did use generative AI tools for two tasks whose disclosure is recommended: creating and refining the figures in this paper, and copy-editing the manuscript to improve grammar and readability. All AI-assisted figures were verified against the raw experimental outputs by the authors, and all AI-suggested text edits were reviewed by an author and restricted to wording, with no change to the technical content, claims, or conclusions. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025. URL https://openreview.net/forum?id=4oo6XTL6Oj.

Lakshya A Agrawal, Donghyun Lee, Shangyin Tan, Wenjie Ma, Karim Elmaaroufi, Sanjit A. Seshia, Koushik Sen, Dan Klein, Ion Stoica, Joseph E. Gonzalez, Omar Khattab, Alexandros G. Dimakis, and Matei Zaharia. optimize_anything: A universal api for optimizing any text parameter. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 13001304, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400724152. doi: 10.1145/3786335.3813190. URL https://doi.org/10.1145/3786335.3813190.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence, 2026. URL https://arxiv.org/abs/2507.21046.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive llm driven zeroth-order optimization, 2026. URL https://arxiv.org/abs/2602.20133.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. Provable scaling laws for the test-time compute of large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=GBMzJLhsRj.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and LLMs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=rYs2Dmn9tD.

Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems, 2026. URL https://arxiv.org/abs/2505.11831.

ARC Prize Foundation. Arc-agi-3: A new challenge for frontier agentic intelligence, 2026. URL https://arxiv.org/abs/2603.24621.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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. Llm4ad: A platform for algorithm design with large language model, 2026a. URL https://arxiv.org/abs/2412.17287.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alvin Cheung, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. Evox: Meta-evolution for automated discovery, 2026b. URL https://arxiv.org/abs/2602.23413.

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Matei Zaharia, Alexandros G. Dimakis, and Ion Stoica. Skydiscover: A flexible, adaptive framework for ai-driven scientific and algorithmic discovery. In *Proceedings of the ACM Conference on AI and Agentic Systems*, CAIS '26, pp. 1223–1227. Association for Computing Machinery, 2026c. doi: 10.1145/3786335.3813221. URL https://doi.org/10.1145/3786335.3813221.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Daniel J. Mankowitz, Andrea Michi, Anton Zhernov, Marco Gelmi, Marco Selvi, Cosmin Paduraru, Edouard Leurent, Shariq Iqbal, Jean-Baptiste Lespiau, Alex Ahern, Thomas Koppe, Kevin Millikin, Stephen Gaffney, Sophie Elster, Jackson Broshear, Chris Gamble, Kieran Milan, Robert Tung, Minjae Hwang, Taylan Cemgil, Mohammadamin Barekatain, Yujia Li, Amol Mandhane, Thomas Hubert, Julian Schrittwieser, Demis Hassabis, Pushmeet Kohli, Martin Riedmiller, Oriol Vinyals, and David Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618(7964):257–263, 2023. doi: 10.1038/s41586-023-06004-9.

Alexander Novikov, Ngân V, 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. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. Kernelbench: Can LLMs write efficient GPU kernels? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=yeoN1iQT1x.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

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. Mathematical discoveries from program search with large language models. *Nature*, 2023. doi: 10.1038/s41586-023-06924-6.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017. URL https://arxiv.org/abs/1502.05477.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

George Tsoukalas, Anton Kovsharov, Sergey Shirobokov, Anja Surina, Moritz Firsching, Gergely Bérczi, Francisco J. R. Ruiz, Arun Suggala, Adam Zsolt Wagner, Eric Wieser, Lei Yu, Aja Huang, Miklós Z. Horváth, Andrew Ferraiuolo, Henryk Michalewski, Edward Lockhart, Codrut Grosu, Thomas Hubert, Matej Balog, Pushmeet Kohli, and Swarat Chaudhuri. Advancing mathematics research with ai-driven formal proof search, 2026. URL https://arxiv.org/abs/2605.22763.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Chenxing Wei, Hong Wang, Ying He, Zhongxiang Dai, Bo Jiang, Fei Yu, and Yao Shu. Words & weights: Streamlining multi-turn interactions via co-adaptation. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=rEpmIPMftW.

Chenxing Wei, Hong Wang, Ying He, Yao Shu, and Fei Yu. Test-time policy adaptation for enhanced multi-turn interactions with LLMs, 2026b. URL https://openreview.net/forum?id=B291oHzQq0.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Trans. Evol. Comp*, 1(1):6782, April 1997. ISSN 1089-778X. doi: 10.1109/4235.585893. URL https://doi.org/10.1109/4235.585893.

Xingyu Wu, Sheng-Hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. Evolutionary computation in the era of large language model: Survey and roadmap. *Trans. Evol. Comp*, 29(2):534554, April 2025. ISSN 1089-778X. doi: 10.1109/TEVC.2024.3506731. URL https://doi.org/10.1109/TEVC.2024.3506731.

Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Zou. metatextgrad: Learning to learn with language models as optimizers. In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*, 2024. URL https://openreview.net/forum?id=yzieYIT9hu.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024. https://github.com/ai4co/reevo.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025a.

Xinchen Zhang, Ling Yang, Guohao Li, YaQi Cai, xie jiake, Yong Tang, Yujiu Yang, Mengdi Wang, and Bin CUI. Itercomp: Iterative composition-aware feedback learning from model gallery for text-to-image generation. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=4w99NAikOE.

Yisong Zhang, Ran Cheng, Guoxing Yi, and Kay Chen Tan. A systematic survey on large language models for evolutionary optimization: From modeling to solving, 2026. URL https://arxiv.org/abs/2509.08269.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=Do1OdZzYHr.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

A Optimizer Mapping and Implementation Alignment

A.1 Expanded Related Frameworks

To fully contextualize OptiCom, we analyze its relationship with recent large-scale LLM optimization frameworks and platforms, explicitly delineating our unique contributions against each:

LLM4AD (Large Language Models for Algorithm Design) Liu et al. (2026a): LLM4AD focuses on automating the discovery of programmatic algorithms using LLMs. It typically employs fixed evolutionary search pipelines (e.g., Evolution of Heuristics) to iteratively generate, evaluate, and mutate code-based solutions. *Difference:* While LLM4AD is domain-centric (algorithm design) and prescribes specific, hardcoded evolutionary search topologies, OptiCom is a general-purpose, state-conditioned framework. Rather than executing a static pipeline, OptiCom dynamically alters its foundational search topologyfluidly switching from evolutionary recombination to error-guided local revisionbased on real-time optimization state and stagnation signals.

SkyDiscover Liu et al. (2026c): SkyDiscover serves as a comprehensive, multi-domain benchmarking platform designed to standardize the evaluation of LLM optimizers across diverse fields (e.g., mathematics, systems, quantum circuits). It provides execution harnesses and static baseline implementations of various methods. *Difference:* SkyDiscover is the *evaluation environment*, whereas OptiCom is the *active optimization agent*. We utilize SkyDiscover's rigorous tasks as our testbed, but fundamentally differ by introducing a dynamic Controller-Adapter architecture capable of state-conditioned mechanism routing, rather than acting as another static algorithm profile on the benchmark.

Universal Optimization APIs (e.g., optimize_anything Agrawal et al. (2026)): Recent works like `optimize_anything` propose unified APIs that treat diverse tasks (from agent architecture search to CUDA kernel generation) as general text optimization problems. They typically rely on a single, powerful LLM proposer guided by textual feedback to iteratively improve an artifact. *Difference:* While universal APIs provide a vital abstraction for defining tasks, they often rely on a monolithic search strategy (e.g., a fixed prompt template generating candidates based on recent feedback). OptiCom subsumes such operations into its unified AQOEMS configuration space. Instead of committing to a single proposal strategy for an entire run, OptiCom treats different search mechanisms (e.g., broad tree exploration, local diagnostic querying) as composable Action Packages, transitioning between their behaviors when the trajectory demands it.

A.2 Scope of the Contribution

The primary contribution of this work lies in formalizing the unified AQOEMS representation and providing an executable, state-conditioned composition interface (OptiCom), validated through extensive empirical evaluation across 32 benchmark groups. Rather than introducing new individual heuristics for adaptive algorithm selection, evolutionary search, reflection, or self-modification, our work provides the rigorous architectural foundation for their dynamic coordination.

Furthermore, the theoretical decomposition presented in Section 4 is purposefully designed to explicitly distinguish between search opportunities and selection losses in LLM optimization; it specializes existing performance-difference identities to motivate our framework's design, rather than claiming a fundamentally new mathematical theorem. Finally, the framework's extensibility ensures that new query, operator, evaluation, and memory mechanisms can be seamlessly registered via strict task contracts, though minor structural alignments may be required to express highly idiosyncratic legacy optimizers within our shared interfaces.

A.3 AQOEMS Responsibilities and Mapping Criteria

AQOEMS provides a functional decomposition of an optimizer through $C = (A, Q, O, E, M, S)$. These dimensions describe what is optimized, what information is acquired, how candidates are updated, how outcomes are evaluated, what experience is retained, and how these decisions are coordinated. They do not prescribe six separate modules or a fixed execution order. Functions may share an implementation, and not every dimension requires an explicit mechanism: unused functions

may take null or minimal forms. Table 5 summarizes the responsibilities and illustrative realizations of each dimension.

Table 5: Functional dimensions of AQOEMS. The dimensions describe responsibilities rather than mandatory modules or sequential execution steps. The examples are possible realizations, not required components; their use may depend on the optimization state through S.

Dimension	**Optimization concept**	**Functional responsibility**	**Illustrative realizations**
A: Artifact	Solution representation	What is optimized, and how candidates are represented.	Source code, numerical parameters, mathematical constructions, prompts, or images.
Q: Query	Information acquisition	What evidence and context are acquired or assembled to inform an update.	Inspect selected failures, request profiling evidence, retrieve past attempts, or use only the current candidate and feedback.
O: Operator	Search and update operations	How candidates are generated or modified.	Generate a new candidate, repair an error, apply a local revision, or recombine existing candidates.
E: Evaluation	Objective and constraint assessment	How candidate quality and feasibility are assessed, and what feedback is returned.	Correctness checks, runtime measurements, diagnostic tests, or higher-fidelity verification.
M: Memory	Search history and experience	What information persists across iterations, and how it is maintained.	Retain an incumbent, a candidate archive, evaluation records, or reusable lessons; omit persistent experience storage when unused.
S: Strategy	Search control	How the other functions are coordinated and resources are allocated as optimization proceeds.	Follow a prescribed schedule, apply state-dependent rules, or adapt selection preferences using accumulated outcomes.

Functional boundaries. The dimensions distinguish roles within an optimization process, even when those roles share code or model calls. For example, E may execute a candidate program and return correctness results, runtime measurements, and failure records. M specifies how these records and associated candidates persist across iterations, while Q specifies how selected failures, additional measurements, or stored experiences are obtained and assembled for a subsequent update. O defines the transformation applied to the candidate, and S coordinates which information-acquisition and update mechanisms are used and with what resources. Thus, retrieving a past attempt involves both the storage functionality of M and the information-access functionality of Q. Likewise, Q may request additional measurements through procedures defined by E. These interactions do not require separate implementations for each responsibility.

State-dependent realization. The mechanisms used to fulfill these responsibilities may depend on the current candidate, unresolved failures, accumulated experience, and remaining budget. For instance, an invalid program may trigger failure inspection and repair, whereas a valid but slow program may trigger profiling and a targeted performance revision. This does not require every dimension to change at every iteration: the artifact representation may remain fixed while context selection, update operators, and evaluation depth vary. Such variation must remain consistent with the task objective and feasibility requirements; changing an evaluation procedure does not by itself redefine the optimization problem.

Configurations and execution policies. A configuration specifies mechanisms and coordinating rules rather than a predetermined sequence of actions. In particular, S may include conditional decisions or internal adaptation based on the optimization state. Representing an existing optimizer as one point in the configuration space can express its adaptive behavior; a mapping alone does not verify that an experimental profile reproduces it. Holding that configuration fixed means retaining its

defining mechanisms and rules, not forcing identical actions across iterations. The reference policy in Section 4 uses this distinction: it may contain state-dependent or internally adaptive rules. The theory does not equate a fixed configuration with a constant action.

Mapping criteria. We map existing systems according to their implemented functionality, rather than their original module names. A mapping identifies the candidate representation, information sources, update operations, evaluation procedures, persistent state, and coordinating rules. Where a responsibility is implicit, shared with another component, or absent, the mapping records that fact instead of introducing an additional mechanism. Internal adaptation is included in S, and task-specific assumptions are retained. A conceptual mapping alone does not establish implementation equivalence: any changes to prompts, operators, feedback access, or resource accounting must be documented when instantiating a method within the framework. Complete mappings are provided in Appendix A.4, with implementation differences and alignment checks in Appendix A.5.

A.4 Complete AQOEMS Mappings

Table 6 provides functional mappings for the methods associated with our baseline configurations and for two additional conceptual references, Trace and metaTextGrad. The mappings describe the published mechanisms; they are not, by themselves, claims that every mechanism is reproduced in our experimental implementation. The correspondence between these mechanisms and the registered implementations is addressed in Appendix A.5.

We distinguish methods with a corresponding registered configuration (**B**) from conceptual mappings only (**C**). The B/C annotations distinguish the 13 method-inspired baseline profiles evaluated here from conceptual references; conceptual entries are not empirical comparisons. For framework-level entries, the mapping describes a family of configurations whose concrete behavior depends on the selected optimizer.

Table 6: Complete functional mappings. B denotes a corresponding baseline profile in the supplied registry; C denotes conceptual coverage only. B does not certify equivalence to the original implementation. The two right columns jointly specify all six AQOEMS dimensions.

Method / status	Artifact, query, and operator	Evaluation, memory, and strategy
OPRO Yang et al. (2024) **B**	**A:** Candidate solutions, including prompts. **Q:** Task description and selected scored solutions. **O:** Generate proposals conditioned on that history.	**E:** Task objective values. **M:** Evaluated candidate–score records. **S:** Repeated generation and evaluation with history selection and stopping rules.
TextGrad Yuksekgonul et al. (2025) **B**	**A:** Optimizable variables in a computational graph. **Q:** Relevant graph context and propagated feedback. **O:** Variable updates guided by textual feedback.	**E:** Objective evaluation and associated criticism. **M:** Graph state, current variables, and optimizer-dependent update history. **S:** Feedback propagation and variable-update procedure.
ProTeGi Pryzant et al. (2023) **B**	**A:** Task prompts. **Q:** Minibatch examples and observed errors. **O:** Generate textual critiques and corresponding prompt revisions.	**E:** Predictive performance on evaluation examples. **M:** Beam candidates and evaluation statistics. **S:** Beam expansion and bandit-based candidate selection.
Reflexion Shinn et al. (2023) **B**	**A:** Task attempts, including programs or action trajectories. **Q:** Current feedback and previous verbal reflections. **O:** Produce a subsequent attempt informed by reflection.	**E:** Task-dependent external or internal feedback. **M:** Episodic verbal reflections. **S:** Attempt, evaluate, reflect, and retry.

Continued on next page

Table 6 continued

Method / status	Artifact, query, and operator	Evaluation, memory, and strategy
GEPA Agrawal et al. (2025) **B**	**A:** Prompts in an AI system. **Q:** Execution trajectories, diagnostics, and selected candidate context. **O:** Reflective mutation and combination of complementary candidates.	**E:** Task evaluations, including performance across examples. **M:** Candidate pool, scores, and ancestry. **S:** Pareto-aware selection and evolutionary updates under an evaluation budget.
FunSearch Romera-Paredes et al. (2023) **B**	**A:** Functions within a program specification. **Q:** Selected prior functions assembled into a prompt. **O:** LLM-generated function variants.	**E:** Automated execution and scoring. **M:** An island-structured database of evaluated programs. **S:** Program sampling, insertion, and island reset rules.
AlphaEvolve Novikov et al. (2025) **B**	**A:** Programs implementing candidate solutions. **Q:** Selected programs, evaluation feedback, and task context. **O:** LLM-proposed code changes.	**E:** Automated evaluators, potentially organized in stages. **M:** Evaluated program database and associated metadata. **S:** Evolutionary sampling, evaluation, and population management.
AdaEvolve Cemri et al. (2026) **B**	**A:** Candidate programs. **Q:** Selected program context and improvement evidence. **O:** LLM-generated program variations.	**E:** Program fitness. **M:** Island populations and accumulated improvement statistics. **S:** Adaptive exploration intensity, inter-island resource allocation, and meta-guidance.
EvoX Liu et al. (2026b) **B**	**A:** Candidate programs; search procedures at the meta level. **Q:** Candidate history and evidence about search performance. **O:** Program variation and meta-level modification of search procedures.	**E:** Candidate quality and the performance induced by search strategies. **M:** Candidate and strategy histories. **S:** Coupled evolution of solutions and their generation and management procedures.
Voyager Wang et al. (2024) **B**	**A:** Executable skill programs. **Q:** Environment observations, errors, and retrieved skills. **O:** Skill generation and iterative program revision.	**E:** Environment feedback and task-success verification. **M:** A reusable skill library. **S:** Automatic curriculum and iterative skill acquisition.
DSPy Khattab et al. (2024) **B**	**A:** Parameters of an LM program, such as instructions and demonstrations. **Q:** Training examples and optimizer-dependent traces. **O:** Updates supplied by the selected DSPy optimizer.	**E:** A user-specified task metric. **M:** Optimizer-dependent candidate, demonstration, and search state. **S:** The selected compilation and search procedure; DSPy does not specify a single universal optimizer.
Self-Refine Madaan et al. (2023) **B**	**A:** Generated outputs. **Q:** The current output and feedback context. **O:** Revision using self-generated feedback.	**E:** LLM-generated assessment of the output. **M:** Outputs and feedback retained in the refinement context. **S:** Alternating feedback and refinement until a stopping condition is met.
LATS Zhou et al. (2024) **B**	**A:** Partial and complete action or reasoning trajectories. **Q:** Environment observations and reflections. **O:** Generate and expand candidate trajectory continuations.	**E:** Environment rewards and LM-based value estimates. **M:** Search-tree statistics and reflection history. **S:** Monte Carlo tree search with selection, expansion, evaluation, and backpropagation.

Continued on next page

Table 6 continued

Method / status	Artifact, query, and operator	Evaluation, memory, and strategy
Trace / OptoPrime Cheng et al. (2024) **C**	**A:** Optimizable parameters of computational workflows. **Q:** Execution traces and associated feedback. **O:** Updates supplied by OptoPrime or another optimizer connected to Trace.	**E:** Workflow feedback supplied through a trace oracle. **M:** Workflow state and optimizer-dependent history. **S:** The selected optimizer's update procedure.
metaTextGrad Xu et al. (2024) **C**	**A:** Optimizer prompts and compositions at the outer level. **Q:** Reference optimizers and validation evidence. **O:** Prompt refinement and structural composition of optimizers.	**E:** Downstream validation performance after optimization. **M:** Reference and previously evaluated optimizer candidates. **S:** Meta-level proposal, evaluation, and selection, with separate prompt and structure optimization.

Frameworks and nested optimization. A framework such as DSPy or Trace corresponds to multiple possible configurations because its update rules depend on the optimizer attached to it. Naming the framework alone is therefore insufficient to identify an experimental baseline. Trace supplies an execution-trace abstraction for generative optimization, while AQOEMS describes the responsibilities that constitute the optimizer operating on such information. For metaTextGrad, the mapping in Table 6 is made at the outer level, where optimizers are the artifacts being improved. Its inner optimization process can be represented by a separate AQOEMS configuration. The same distinction applies when EvoX modifies the procedures used by its candidate search. These nested representations preserve the level at which each decision is made.

Task adaptation. The artifact and evaluator in an experimental configuration may differ from those in the source method's original application. For example, transferring a prompt-update mechanism to program optimization changes the artifact representation and evaluation interface. Such an instance is a task-adapted configuration, whose relationship to the source method depends on the mechanisms retained. Similarly, representing Voyager's skill-improvement loop does not make a bounded optimization benchmark equivalent to its original open-ended environment. The shared representation exposes these adaptations so that they can be reported explicitly.

Memory and strategy are functional, not nominal. A population, search tree, beam, or in-prompt candidate history constitutes memory even when the original implementation has no module named "memory." Likewise, a method with no explicit strategy-adaptation module still has coordinating rules in S. Conversely, an adaptive S can be part of one fixed configuration. These conventions prevent implementation labels such as "no memory" or "no strategy" from obscuring state and control that are present elsewhere in the execution process.

A.5 Implementation Differences and Alignment Checks

The mappings in Appendix A.4 describe how existing optimizers can be instantiated within the shared configuration space. A functional mapping identifies the responsibilities of an optimizer, but does not by itself establish equivalence to its original implementation. We therefore distinguish the mechanism that defines a method from the interfaces used to execute it. For example, an evolutionary method is characterized by how it selects and combines candidates, whereas a feedback-guided method is characterized by how feedback informs subsequent revisions. These mechanisms can operate through shared candidate, evaluation, and memory interfaces without requiring identical prompts, storage formats, or execution infrastructure.

An instantiation should preserve the dependencies that make its defining mechanism effective. Retaining a reflection module, for instance, is insufficient if its output is never supplied to subsequent attempts; similarly, retaining a candidate archive is insufficient if the selection procedure does not use it. Table 7 organizes alignment checks around such observable dependencies. Components may be stateful or randomized, and a fixed configuration may contain an adaptive strategy. Alignment

therefore concerns the method's decision rules and information flow, rather than requiring the same action at every iteration.

Table 7: Criteria for checking framework instantiations. These checks distinguish preservation of a method's defining mechanism from changes introduced by shared interfaces. The table specifies verification criteria rather than certifying equivalence to official implementations.

Aspect	**Mechanism to preserve**	**Alignment check**
Candidate generation	The information and update rule used to produce new candidates.	Check that generation receives the intended parents, examples, feedback, or revision instructions.
Selection and search	The method's candidate selection, branching, population, or tree-search rule.	Check that these rules affect executed actions, rather than appearing only in configuration descriptions.
Feedback use	The feedback representation and its role in subsequent optimization.	Check that evaluation records, reflections, or textual critiques reach the decisions they are intended to guide.
Memory and reuse	The information retained across iterations and the rule for retrieving it.	Check that retained candidates or experiences remain available and are retrieved according to the configured mechanism.
Strategy adaptation	The trigger, evidence, and scope of changes to the optimization procedure.	Check that strategy updates affect later decisions; distinguish adaptive rules within a configuration from changes to the configuration itself.
Evaluation and budget	Task validity requirements, scoring semantics, and resource constraints.	Check that candidates use the shared benchmark evaluator and that reported limits include the relevant optimization operations.

Implementation differences should be documented at the level at which they can affect behavior. Relevant differences include prompt templates, population or branch sizes, retrieval rules, adaptation schedules, stopping conditions, and access to external tools. A component that is omitted or replaced should be identified explicitly rather than inferred from the method name. Likewise, translating a method to a new artifact domain may require a different proposal template while retaining its selection and feedback mechanisms. Comparisons in this paper therefore concern the evaluated framework instantiations, rather than unrestricted claims about every implementation of the corresponding methods.

This distinction also separates optimizer alignment from benchmark alignment. As described in Section 6.1, the benchmark evaluation pipelines are integrated from SkyDiscover, with study-specific aggregation stated explicitly in Appendix E.1. Sharing this evaluation pipeline makes candidate outcomes comparable; it does not alone establish that each optimizer reproduces all details of its official implementation. The common representation provides a basis for making these retained mechanisms and implementation differences explicit.

B Motivation Study: Protocol and Supplementary Results

B.1 Task, States, and Reference Configuration

We use the Heilbronn triangle task in `benchmarks/math/heilbronn_triangle`. A candidate program implements `heilbronn_triangle11()` and returns an array of shape $(11, 2)$ containing points within or on the boundary of the equilateral triangle $\mathcal{T}$ with vertices $(0, 0)$, $(1, 0)$, and $(1/2, \sqrt{3}/2)$. The objective is to maximize the smallest triangle area among all triples of points:

$$\max_{p_1,\ldots,p_{11}\in\mathcal{T}} \min_{1\le i<j<k\le 11} \frac{\mathrm{Area}(p_i, p_j, p_k)}{\mathrm{Area}(\mathcal{T})}. \tag{7}$$

The evaluator checks output shape and triangle containment, evaluates all $\binom{11}{3} = 165$ triples, and returns the benchmark score and associated execution evidence. We distinguish these objective measurements from the binary state-resolution outcomes used in this diagnostic study and from the bounded terminal utility in Section 4.

The reference configuration is $R = Q2$ self-only + $O1$ local revision + $E2$ textual feedback + $M1$ no persistent experience memory + $S1$ fixed strategy. In the base intervention matrix, one mechanism is changed while the remaining reference rules are retained. Each of the five state families contains three frozen starting situations, and each mechanism–situation pair is evaluated over ten trials with the same maximum ten-iteration horizon. Thus, a displayed mechanism–state rate aggregates 30 trials with equal weight across the three situations. Reference trials are shared when the reference setting reappears across dimensions; they are not counted as independent repetitions.

B.2 State Definitions and Success Criteria

The five state families are: s1 execution or hard-constraint failure; s2 failed correctness tests; s3 objective stagnation; s4 limited remaining budget; and s5 high performance variability. A trial is successful if it resolves the targeted state condition within the declared budget while preserving all requirements that should already hold at that stage. Table 8 summarizes the criteria. GPT-5.5 judges state resolution according to explicitly defined, pre-specified criteria for the starting situations. The reported success rates therefore measure criterion-based LLM judgments of state resolution, rather than the benchmark objective score itself. Benchmark execution checks and objective measurements remain distinct from this diagnostic judgment.

Table 8: Optimization states and state-resolution criteria in the Heilbronn diagnostic study.

State	Initial condition	Success criterion
s1	Execution failure or hard-constraint violation	Recover a candidate that executes and satisfies the required hard constraints.
s2	Failed correctness tests	Preserve feasibility and pass all required correctness checks.
s3	Objective stagnation	Preserve feasibility and correctness and exceed the predefined objective-improvement criterion.
s4	Limited remaining budget	Reach the s3 objective criterion within the declared remaining budget.
s5	High performance variability	Reduce variability beyond the predefined criterion without exceeding the allowed degradation in mean quality, feasibility, or correctness.

B.3 Three Situations per State

The three situations within each state represent different failure modes rather than repeated copies of one snapshot. All mechanisms compared within a state use the same three starting situations and the same situation-specific resource limits. The 15 situations are listed below.

Table 9: Starting situations for the expanded Heilbronn Triangle diagnostic study. Each state contains three situations with ten trials per mechanism and situation.

ID	Situation	Starting condition
s1-1	Runtime exception	Point generation or updating raises an exception, such as an out-of-range index or incompatible array dimensions, and terminates without returning a point set.
s1-2	Construction timeout	Excessive search, too many internal iterations, or an ineffective termination condition prevents the construction from finishing within the declared execution limit.
s1-3	Invalid output interface	The program terminates but returns the wrong number of points, an array other than shape $(11, 2)$, or coordinates containing NaN or infinity.
s2-1	Incorrect boundary handling	The program returns a well-formed point set, but rectangular clipping or an incorrect projection leaves points outside the sloping boundaries of the equilateral triangle.

Continued on next page

Table 9 continued

ID	Situation	Starting condition
s2-2	Incorrect area objective	The search uses an incorrect internal area calculation, such as omitting the absolute determinant or using an incorrect normalization factor, so its internal objective disagrees with the specified geometric objective.
s2-3	Incomplete triple enumeration	The internal objective evaluates only a subset of triples, such as consecutive or locally selected points, and can miss the triple determining the true minimum area.
s3-1	Local-perturbation plateau	A valid construction repeatedly applies small coordinate perturbations near the incumbent layout, without a qualifying improvement during the predefined observation window.
s3-2	Structure-restricted plateau	Search retains a fixed layout structure, such as symmetry constraints, a fixed number of boundary points, or a fixed grouping, and parameter updates fail to improve the score during the observation window.
s3-3	Unproductive restarts	Repeated restarts use the same initialization distribution and search rules, producing similarly scored solutions without improving the retained best score during the observation window.
s4-1	Expensive construction	The current valid candidate requires substantial internal search or local optimization; continuing that procedure consumes most of the remaining budget and leaves little room for further attempts.
s4-2	Too many active branches	Several exploration directions remain active, but the remaining budget cannot support continued substantial allocation to every branch under the current schedule.
s4-3	Budget-mismatched updates	The current procedure proposes large structural changes or complete restarts whose search and verification demands are poorly matched to the limited remaining budget.
s5-1	Initialization sensitivity	Different random initial point sets lead the same construction procedure to substantially different final scores, with strong outcomes in some executions and weak outcomes in others.
s5-2	Update-path sensitivity	Starting from the same initial point set, random choices of updated points, perturbation directions, or update order produce substantially different final scores.
s5-3	Strategy-selection sensitivity	The construction procedure randomly selects among search or construction strategies with differing reliability, producing substantial variation in final performance across executions.

Thirty-trial aggregation. For state s, dimension d, mechanism v, situation $c \in \{1,2,3\}$, and repetition $r \in \{1,\dots,10\}$, let $Y^{(r)}_{s,d,v,c}$ be the binary state-resolution outcome. We report

$$\widehat{p}_{s,d,v} = \frac{1}{30}\sum_{c=1}^{3}\sum_{r=1}^{10} Y^{(r)}_{s,d,v,c} = \frac{1}{3}\sum_{c=1}^{3}\widehat{p}_{s,d,v,c}, \qquad \widehat{p}_{s,d,v,c} = \frac{1}{10}\sum_{r=1}^{10} Y^{(r)}_{s,d,v,c}. \tag{8}$$

Dimension-level summaries average these mechanism rates over the displayed alternatives. Because each state has a different resolution criterion, rates should be compared primarily within a state rather than interpreted as equal changes in a shared objective scale.

B.4 Mechanisms, Memory Gating, and Complete Situation-Level Results

The query alternatives are none, self-only, environment probing, and memory lookup. Operator alternatives are local revision, repair, and recombination. Evaluation alternatives are scalar-only, textual feedback, and staged evaluation. Strategy alternatives are fixed rules, bandit adaptation, and meta-evolution. In the strict one-dimension reference intervention, memory lookup is inert when $M1$ stores no reusable library, and episodic or structured memory is inert when $Q2$ never reads it. We therefore report the strict base matrix separately from a gate-open memory analysis: episodic,

structured, and adaptive memory are paired with $Q4$ memory lookup as the reader, while the $Q4$ row itself remains evaluated under $M1$. This qualification is important because the memory panel is a conditional memory comparison rather than a literal one-coordinate intervention.

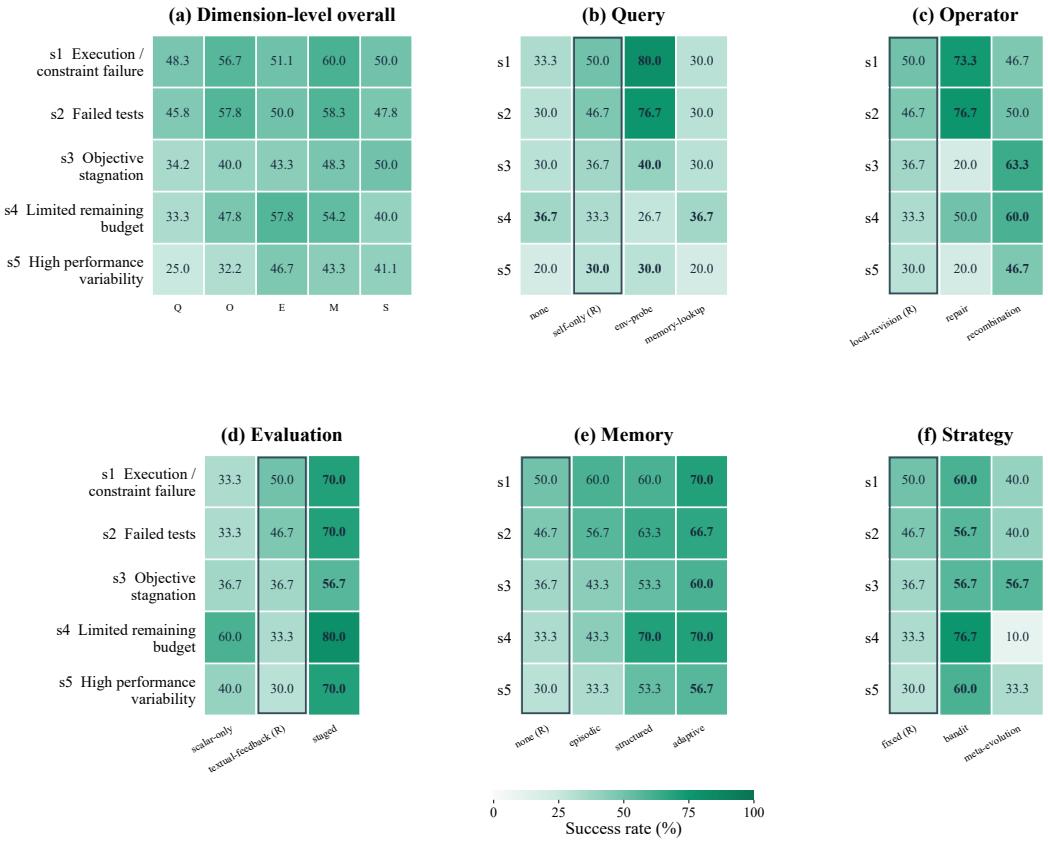

Figure 5: Complete mechanism-level view of the Heilbronn Triangle motivation study. Each displayed mechanism–state value averages the three situations in that state, with ten trials per situation. **(a)** Dimension-level averages. **(b)–(f)** Query, operator, evaluation, memory, and strategy mechanisms. R denotes the reference setting. The memory panel uses the gate-open comparison described in this appendix so that stored episodic, structured, and adaptive memory can be read by the memory-lookup query mechanism.

The state-level rates plotted in Figure 1 follow directly from these counts. Examples highlight distinct mechanism roles. In s4-2, where too many branches compete for the remaining budget, bandit adaptation reaches 9/10 compared with 3/10 for the fixed reference, while staged evaluation reaches 8/10 by reducing evaluation waste rather than reallocating branch budget. In s3-2, the search structure itself restricts progress: meta-evolution reaches 6/10, recombination also reaches 6/10, and repair reaches only 2/10. The hardest execution-failure situation is s1-2, construction timeout, where even environment probing and repair drop to 7/10 and 6/10 because locating the bottleneck does not by itself perform the required algorithmic restructuring. These differences are why the main text reports state averages but the appendix retains situation-level outcomes.

B.5 Joint Query and Operator Composition

We additionally examine query/operator composition across all three s2 situations. The reference uses self-only queries and local revision. Q-only replaces self-only with environment probing; O-only replaces local revision with repair; and Q+O applies both changes. Evaluation, memory, and strategy retain their reference settings. Each configuration therefore contributes 30 trials (ten per s2

Table 10: Successful trials out of ten for all 15 situations in the strict base intervention matrix. R marks the reference mechanism. Memory rows $M2$ and $M3$ are inert here because the reference query does not read persistent memory; the gate-open memory comparison is reported separately in Table 11.

ID	Mechanism	s1-1	s1-2	s1-3	s2-1	s2-2	s2-3	s3-1	s3-2	s3-3	s4-1	s4-2	s4-3	s5-1	s5-2	s5-3
Q1	none	3	3	4	3	3	3	3	3	3	4	3	4	2	2	2
Q2	self-only (R)	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
Q3	env-probe	9	7	8	8	8	7	4	4	4	2	3	3	3	3	3
Q4	memory-lookup	3	2	4	3	3	3	3	3	3	4	3	4	2	2	2
O1	local-revision (R)	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
O2	repair	8	6	8	8	8	7	2	2	2	5	4	6	2	2	2
O3	recombination	5	4	5	5	5	5	7	6	6	6	6	6	5	5	4
E1	scalar-only	3	3	4	4	3	3	4	3	4	6	6	6	4	4	4
E2	textual-feedback (R)	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
E3	staged	7	6	8	7	7	7	6	5	6	8	8	8	7	7	7
M1	none (R)	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
M2	episodic	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
M3	structured	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
S1	fixed (R)	5	4	6	5	5	4	4	3	4	4	3	3	3	3	3
S2	bandit	6	5	7	6	6	5	6	5	6	7	9	7	5	5	8
S3	meta-evolution	4	3	5	4	4	4	5	6	6	1	1	1	3	3	4

Table 11: Gate-open memory comparison. Episodic, structured, and adaptive memory are paired with $Q4$ memory lookup so that stored experience can affect later decisions. Entries are successful trials out of ten.

ID	Mechanism	s1-1	s1-2	s1-3	s2-1	s2-2	s2-3	s3-1	s3-2	s3-3	s4-1	s4-2	s4-3	s5-1	s5-2	s5-3
M2	episodic + Q4	6	5	7	6	6	5	5	3	5	5	4	4	3	3	4
M3	structured + Q4	6	5	7	7	6	6	6	4	6	7	7	7	5	5	6
M4	adaptive + Q4	7	6	8	7	6	7	6	5	7	7	7	7	5	5	7

situation). Average iterations measure the iteration of first success, conditioned on the trial succeeding.

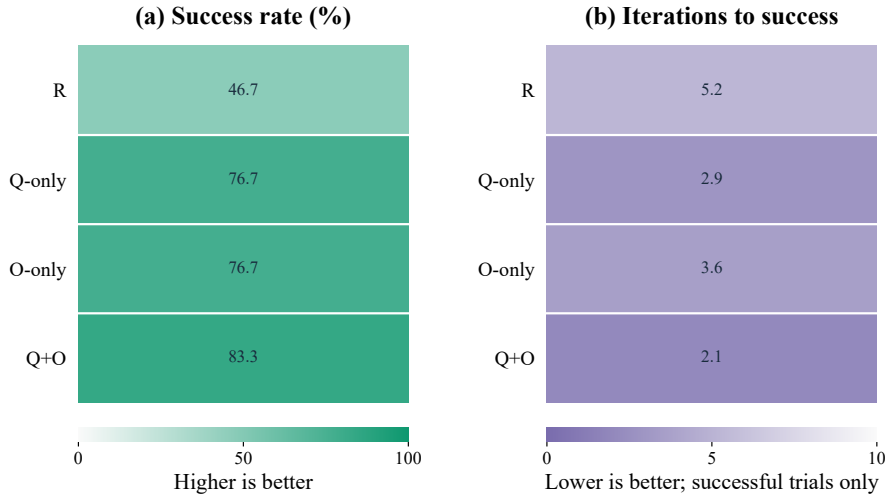

Figure 6: Separate and joint query/operator changes across the three s2 situations. **(a)** Success rate over 30 trials per configuration; higher is better. **(b)** Mean iterations to first success among successful trials only; lower is better. The reference, Q-only, O-only, and Q+O settings obtain 46.7%, 76.7%, 76.7%, and 83.3% success, respectively, with conditional mean iterations of 5.2, 2.9, 3.6, and 2.1.

The joint intervention improves the observed state-resolution rate by approximately 6.7 percentage points over either single change ($(25-23)/30 \times 100$) and shortens successful trajectories relative to both. This result is consistent with complementary composition on these constructed s2 situations,

Table 12: Joint Q/O results across the three s2 situations. Average iterations are conditional on success.

Configuration	Query	Operator	Successes	Avg. iterations
R	self-only	local-revision	14/30	5.2
Q-only	env-probe	local-revision	23/30	2.9
O-only	self-only	repair	23/30	3.6
Q+O	env-probe	repair	25/30	2.1

but it does not establish general superadditivity: the experiment covers one pair of dimensions on one task, and iteration counts need not correspond to equal token or wall-clock cost.

C Proofs of Theoretical Results

C.1 Budgeted Process and Notation

We analyze one task at a time, with initial history h_0, budget B, and a finite decision cap H. A history h_t contains the task, candidate records, observations, remaining resources, and the internal state needed to specify continuation policies. A policy π selects an action from a finite nonempty feasible set $\mathcal{A}(h_t)$, possibly through an observation summary $\phi(h_t)$. The history representation avoids assuming that the summary is Markov or fully informative.

Shared transitions and costs. Actions follow a common transition kernel $K(dh' \mid h, a)$. If $\kappa_t \geq 0$ is the cost incurred at decision t, then

$$b_0 = B, \qquad b_{t+1} = b_t - \kappa_t \geq 0, \qquad \sum_{t<T} \kappa_t \leq B. \tag{9}$$

The model includes the costs of decision formation as well as query, generation, evaluation, memory maintenance, and strategy adaptation. If two decision procedures incur different control costs, that difference must be represented in their execution modes or explicit control operations. It cannot be hidden inside a policy-dependent transition for an otherwise identical action. A structured Action Package is therefore the implementation's main decision record, while the theoretical history and action description also account for the computation needed to produce and execute it.

We use a scalar budget for notation. Multiple resource limits can instead be carried in the state with componentwise feasibility checks; the proofs rely on a common feasible process and terminal boundary, not a particular cost unit. Random execution costs are handled by enforced limits and specified interruption outcomes. No operation is treated as free merely because it fails.

Stopping and terminal utility. Stop is a feasible action from every active history and enters an absorbing state. Let $T \leq H$ be the number of decisions up to termination; when termination is explicit, the stopping action is included among $A_0, \ldots, A_{T-1}$. After absorption, the only action is a zero-cost no-op and the retained result remains unchanged. We pad trajectories to H for the proof.

The terminal utility $F(h_H)$ is measured by a fixed task-specific rule in $[0, 1]$, including a failure value in the same interval. If no valid candidate exists, the rule returns this failure value; otherwise it scores the best retained candidate meeting the final requirements. Optimization-time evaluation may use different feedback formats or stages, but does not redefine F. Boundedness is a property of the specified terminal utility, not a consequence of dividing an arbitrary benchmark score by a reference target. For a utility bounded in a known interval $[L, U]$ with $U > L$, a fixed positive affine normalization gives the stated form; the corresponding raw-unit loss bound is scaled by $U - L$. No such bounded interval is asserted for every raw metric in our experiments, so the numerical bound is not applied to the reported GPU, systems, or reference-relative scores. The qualitative success rates in the motivation study are not estimates of the continuation values below.

Reference continuation and action values. Fix a reference optimizer π_0 with a defined continuation from every relevant history. It may be randomized and internally adaptive. A stateful reference

requires a specified way to reconstruct or maintain its internal state on histories reached by π. Define

$$J_B(\pi) = \mathbb{E}_\pi[F(h_H) \mid h_0], \tag{10}$$

$$V^{\pi_0}(h_t) = \mathbb{E}_{\pi_0}[F(h_H) \mid h_t], \tag{11}$$

$$q(h_t, a) = \int V^{\pi_0}(h')\, K(dh' \mid h_t, a). \tag{12}$$

The terminal boundary and common initial history imply

$$V^{\pi_0}(h_H) = F(h_H), \qquad V^{\pi_0}(h_0) = J_B(\pi_0). \tag{13}$$

Since these quantities measure terminal quality, no additional immediate improvement reward is added to Equation equation 12. Action cost instead affects continuation through the remaining budget in h'.

Opportunities and losses. For the feasible choices exposed to π, write

$$q^\star(h) = \max_{a \in \mathcal{A}(h)} q(h, a), \tag{14}$$

$$\Delta(h) = q^\star(h) - V^{\pi_0}(h), \tag{15}$$

$$\ell(h) = q^\star(h) - \mathbb{E}_{A \sim \pi(\cdot|h)}[q(h, A)]. \tag{16}$$

The notation $\pi(\cdot \mid h)$ includes policies implemented through $\phi(h)$. Selection loss is nonnegative. Opportunity is nonnegative when the reference choices are retained with identical behavior and cost:

$$V^{\pi_0}(h) = \mathbb{E}_{a \sim \pi_0(\cdot|h)} q(h, a) \le \max_{a \in \mathcal{A}(h)} q(h, a). \tag{17}$$

The decomposition itself does not require $\Delta(h) \ge 0$. Different action catalogs change the split into Δ and ℓ; their difference remains the expected advantage of the selected action over the reference continuation.

C.2 Proof of the Cumulative Opportunity–Loss Decomposition

Proof of Theorem 1. We first identify the contribution of a single decision and then accumulate these contributions through a sequence of hybrid policies.

Conditional decision advantage. Let $g(h)$ be the expected advantage of the action selected by π over the reference continuation. Adding and subtracting the best feasible action value gives

$$\begin{aligned} g(h) &:= \mathbb{E}_{A \sim \pi(\cdot|h)}[q(h, A)] - V^{\pi_0}(h) \\ &= q^\star(h) - V^{\pi_0}(h) - \big(q^\star(h) - \mathbb{E}_{A \sim \pi(\cdot|h)}[q(h, A)]\big) \\ &= \Delta(h) - \ell(h). \end{aligned} \tag{18}$$

This identity holds regardless of whether $g(h)$ is positive.

One decision replacement. For $m = 0, \ldots, H$, define $\pi^{(m)}$ to use π for its first m decisions and π_0 thereafter. Thus, $\pi^{(0)} = \pi_0$ and $\pi^{(H)} = \pi$. The adjacent policies $\pi^{(m)}$ and $\pi^{(m+1)}$ share the same prefix distribution d_m^π of h_m. Conditional on that history, the former uses the reference continuation and the latter executes one π action before returning to it. Consequently,

$$J_B(\pi^{(m)}) = \mathbb{E}_{h_m \sim d_m^\pi}[V^{\pi_0}(h_m)], \tag{19}$$

$$J_B(\pi^{(m+1)}) = \mathbb{E}_{h_m \sim d_m^\pi}\left[\mathbb{E}_{A_m \sim \pi(\cdot|h_m)} q(h_m, A_m)\right]. \tag{20}$$

Subtracting yields

$$\begin{aligned} J_B(\pi^{(m+1)}) - J_B(\pi^{(m)}) &= \mathbb{E}_{h_m \sim d_m^\pi}\left[\mathbb{E}[q(h_m, A_m) \mid h_m] - V^{\pi_0}(h_m)\right] \\ &= \mathbb{E}_{h_m \sim d_m^\pi}[g(h_m)] \\ &= \mathbb{E}_\pi[\Delta_m - \ell_m]. \end{aligned} \tag{21}$$

All changes to later histories are already included in the continuation values; the argument does not couple future trajectories to be identical.

Accumulation over the run. Summing the adjacent differences telescopes:

$$\begin{aligned} J_B(\pi) - J_B(\pi_0) &= J_B(\pi^{(H)}) - J_B(\pi^{(0)}) \\ &= \sum_{m=0}^{H-1} \Big(J_B(\pi^{(m+1)}) - J_B(\pi^{(m)}) \Big) \\ &= \sum_{m=0}^{H-1} \mathbb{E}_\pi[\Delta_m - \ell_m] \\ &= \mathbb{E}_\pi \sum_{m=0}^{H-1} (\Delta_m - \ell_m) \\ &= \mathbb{E}_\pi \sum_{m<T} (\Delta_m - \ell_m). \end{aligned} \tag{22}$$

The finite horizon and bounded values justify exchanging summation and expectation. The last equality uses the zero contributions after absorption. □

Equivalent value-function telescoping. The same result follows directly from conditional expectation:

$$\begin{aligned} \mathbb{E}_\pi \sum_{t=0}^{H-1} (\Delta_t - \ell_t) &= \sum_{t=0}^{H-1} \mathbb{E}_\pi \left[\mathbb{E}_\pi[V^{\pi_0}(h_{t+1}) \mid h_t] - V^{\pi_0}(h_t) \right] \\ &= \mathbb{E}_\pi \sum_{t=0}^{H-1} (V^{\pi_0}(h_{t+1}) - V^{\pi_0}(h_t)) \\ &= \mathbb{E}_\pi[V^{\pi_0}(h_H)] - V^{\pi_0}(h_0) \\ &= J_B(\pi) - J_B(\pi_0). \end{aligned} \tag{23}$$

This is the finite-horizon, terminal-utility form of a performance-difference identity Schulman et al. (2017).

Why the stopping action matters. If π stops at an active history h, then

$$q(h, \mathrm{Stop}) - V^{\pi_0}(h) = F(h) - V^{\pi_0}(h). \tag{24}$$

This accounts for any reference continuation value forgone by stopping. Omitting that decision would omit a potentially negative contribution. After absorption, both continuation values equal the retained quality and the subsequent terms are zero.

C.3 Selection-Loss Bound and Global Improvement

The capability assumption in Equation equation 5 is inspired by analyses that state explicit generation and comparison conditions for LLM selection Chen et al. (2025). That work analyzes answer correctness and particular selection algorithms. Here the assumed probability concerns near-optimal action continuation value under the current budget; it is not supplied by that result. The parameters may vary with history, and no independence across optimization decisions is required.

Bounding the conditional selection loss. Fix h_t and let $X_t = q^\star(h_t) - q(h_t, A_t)$. Bounded terminal utility implies $0 \le X_t \le 1$. Let $\mathcal{G}_t = \{X_t \le \epsilon_t\}$ and $p_t = \Pr(\mathcal{G}_t^c \mid h_t) \le \delta_t$. Splitting the conditional expectation gives

$$\begin{aligned} \ell_t &= \mathbb{E}[X_t \mid h_t] \\ &= \mathbb{E}[X_t \mathbf{1}_{\mathcal{G}_t} \mid h_t] + \mathbb{E}[X_t \mathbf{1}_{\mathcal{G}_t^c} \mid h_t] \\ &\le \epsilon_t \Pr(\mathcal{G}_t \mid h_t) + \Pr(\mathcal{G}_t^c \mid h_t) \\ &= (1 - p_t)\epsilon_t + p_t \\ &= \epsilon_t + p_t(1 - \epsilon_t) \\ &\le \epsilon_t + \delta_t(1 - \epsilon_t) \\ &= (1 - \delta_t)\epsilon_t + \delta_t. \end{aligned} \tag{25}$$

The penultimate step uses $0 \le \epsilon_t \le 1$. Substituting into the conditional advantage gives the single-decision bound:

$$\begin{aligned}\mathbb{E}[q(h_t, A_t) \mid h_t] - V^{\pi_0}(h_t) &= \Delta_t - \ell_t \\ &\ge \Delta_t - (1-\delta_t)\epsilon_t - \delta_t. \end{aligned} \tag{26}$$

Proof of Theorem 2. Apply Equation equation 25 conditionally at every active history reached by π. Combining it with Theorem 1 yields

$$\begin{aligned} J_B(\pi) - J_B(\pi_0) &= \mathbb{E}_\pi \sum_{t<T} (\Delta_t - \ell_t) \\ &\ge \mathbb{E}_\pi \sum_{t<T} (\Delta_t - (1-\delta_t)\epsilon_t - \delta_t) \\ &= \mathbb{E}_\pi \sum_{t<T} \Delta_t - \mathbb{E}_\pi \sum_{t<T} \left((1-\delta_t)\epsilon_t + \delta_t\right). \end{aligned} \tag{27}$$

This is Equation equation 6. □

Interpretation and scope. The bound yields the sufficient condition

$$\begin{aligned} \mathbb{E}_\pi \sum_{t<T} \Delta_t > \mathbb{E}_\pi \sum_{t<T} \big((1-\delta_t)\epsilon_t + \delta_t\big) \\ \implies \quad J_B(\pi) > J_B(\pi_0). \end{aligned} \tag{28}$$

It does not establish that the premise holds for every adaptive policy. The exact decomposition also permits negative local advantages. Writing $g_t^+ = \max\{g(h_t), 0\}$ and $g_t^- = \max\{-g(h_t), 0\}$ gives

$$J_B(\pi) - J_B(\pi_0) = \mathbb{E}_\pi \sum_{t<T} g_t^+ - \mathbb{E}_\pi \sum_{t<T} g_t^-. \tag{29}$$

Thus, positive contributions may compensate for unfavorable decisions. The result compares expected terminal quality at a shared budget, not monotonic separation of two realized trajectories or strict superiority in the infinite-budget limit. We do not establish the conditional selection premise for our Controller or infer it from endpoint scores. Finite-catalog expansion holds existing action values and costs fixed; catalog-dependent prompt overhead or changed continuation policies fall outside that particular comparison.

C.4 Theoretical Remarks and Applicability

Remark on Theorem 1. The contribution of a decision is $\Delta_t - \ell_t$, so a larger Δ_t is useful only to the extent that selection realizes the additional value. At a fixed history, expanding the action set while preserving its existing actions and costs cannot decrease $q^\star(h_t)$. However, if the selected-action distribution remains unchanged, the increase in $q^\star(h_t)$ raises Δ_t and ℓ_t equally, leaving their difference unchanged. Thus, component diversity must be paired with selection that actually exploits the added choices. The summation also permits negative $\Delta_t - \ell_t$ at some steps: such decisions are compatible with overall improvement when positive contributions elsewhere outweigh them.

Remark on Theorem 2. Writing the bound on ℓ_t as $\epsilon_t + \delta_t(1-\epsilon_t)$ reveals that, for fixed δ_t, making ϵ_t small still leaves a bound close to δ_t. Refining already near-optimal choices is therefore insufficient to tighten the guarantee when failures to select them remain frequent. This motivates using diagnostics and accumulated experience to address recurring poor selections, as well as improving the quality of successful choices. Such evidence is not free: additional querying or reflection consumes budget and changes $q(h_t, a)$ itself. The analysis therefore motivates allocating decision effort according to the state, rather than increasing it uniformly. A positive lower bound certifies improvement; a nonpositive one leaves the comparison unresolved.

Theoretical perspective and implementation scope. Theorem 1 provides an exact performance-difference identity for the shared finite-horizon process, while Theorem 2 establishes a general performance envelope under the conditional selection assumption (Equation 5). In practice, our LLM-based Controller and Strategy Adapter operate directly on observed textual and metric feedback without explicitly computing or estimating abstract analytical quantities such as q, Δ_t, or ℓ_t. Furthermore, while the theoretical analysis assumes a normalized bounded utility to cleanly isolate cumulative credit, empirical evaluations involve task-specific, unnormalized score scales. This formulation positions the theory as a foundational design guide rather than an online runtime estimator, offering rigorous first-principles motivation for why state-conditioned mechanism coordination is essential.

D Framework Specification

This appendix specifies the architecture in Section 5: the information exposed to each component, the decisions that may change, and the execution conditions that remain fixed. Task-specific parameter values, prompts used in a particular run, and experimental implementation alignment belong to the experimental record.

D.1 OptiCom Algorithm and Execution Order

Algorithm 1 provides the pseudo-code logic driving the OptiCom framework loop.

Algorithm 1 OptiCom: Adaptive Optimization through Component Composition

Input: Task d, mechanisms $\mathcal{R}$, initial candidates $\mathcal{P}_0$, budget B.
$\mathcal{P} \leftarrow \mathcal{P}_0$; $\mathcal{M} \leftarrow \emptyset$; $g \leftarrow g_0$; $b \leftarrow B$
while $\neg\,\mathrm{Stop}(\mathcal{P}, \mathcal{M}, b)$ **do**
  // Step 1: Select a composition from the current state
  $z \leftarrow \mathrm{Summarize}(d, \mathcal{P}, \mathcal{M}, b)$
  $(a, c_{\mathrm{ctrl}}) \leftarrow \mathrm{Controller}(z, \mathcal{R}, g, b)$
  $b \leftarrow b - c_{\mathrm{ctrl}}$
  if $\neg\,\mathrm{Executable}(a, b)$ **then**
    break
  end if
  // Step 2: Query, generate, and evaluate within the remaining budget
  $(\mathcal{X}, r, c_{\mathrm{exec}}) \leftarrow \mathrm{ExecuteWithHarness}(a, d, \mathcal{P}, \mathcal{M}, b)$
  $b \leftarrow b - c_{\mathrm{exec}}$
  // Step 3: Retain candidates and experience using the selected policy
  $(\mathcal{P}, \mathcal{M}) \leftarrow \mathrm{Update}(\mathcal{P}, \mathcal{M}, a, \mathcal{X}, r)$
  // Step 4: Revise strategy guidance from accumulated outcomes
  if $\mathrm{AdaptNow}(\mathcal{M}, b)$ **then**
    $(g, c_{\mathrm{adapt}}) \leftarrow \mathrm{Adapter}(g, \mathcal{M}, b)$
    $b \leftarrow b - c_{\mathrm{adapt}}$
  end if
end while
return $\mathrm{BestValid}(\mathcal{P})$ // Return task-defined failure if no valid candidate exists

D.2 Component Interfaces and Action Packages

Task contract and registry. The task adapter defines artifact parsing and serialization, the editable region, the optimization objective, mandatory feasibility and correctness checks, permitted evidence sources, the final scoring rule, and resource limits. A registry entry identifies a component's accepted inputs, outputs, applicability conditions, parameter schema, and execution limits. Null or minimal entries permit configurations that do not use a particular function. A new mechanism can be registered when it satisfies a contract; requirements beyond the contract must be exposed as extensions rather than hidden inside an opaque component.

State records. A candidate record contains an identifier, serialized artifact or artifact reference, parent identifiers, the producing action and strategy version, evaluation status, and links to evaluation

records. Evaluation records distinguish validity, objective measurements, diagnostics, and costs. Missing or incomplete checks remain explicitly unknown. Experience entries link an observation or lesson to the candidates, actions, and evidence from which it was derived. These records support reconstruction of the visible context and prevent an unverified proposal from being confused with an evaluated incumbent.

Table 13: Action Package fields and their execution responsibilities. Fields select registered functionality; they do not redefine the task contract.

Field	Decision	Consumer and constraints
q_t	Query plan	Query executor: permitted tools, evidence requests, and retrieval operations; may be empty.
c_t	Context scope	Context builder: candidate identifiers, diagnostics, and experience to expose within the context limit.
o_t	Update operator	Operator executor: a registered mechanism with compatible artifact inputs and required parent candidates.
w_t	Branch width	Scheduler: a positive bounded number of candidate proposals, subject to available resources.
e_t	Evaluation plan	Evaluator: registered stages, effort limits, and feedback form; final required checks remain unchanged.
ρ_t	Retention policy	Archive and experience stores: which evaluated candidates, records, and lessons to retain or expose.

Functional correspondence. The task adapter supplies A; query and context fields invoke Q; the selected update supplies O; and the evaluation plan invokes E. Archive and experience operations implement M. The Controller and Adapter jointly implement S. Branch width and evaluation effort are resource-allocation decisions coordinated by S, rather than additional AQOEMS dimensions. A retained memory entry belongs to M, while the decision to retrieve it for a particular update belongs to Q.

Records versus theoretical quantities. The runtime records observed scores, feedback, and resource use. It does not require a learned value function, an explicit estimate of $q(h, a)$, or a numerical estimate of Δ_t and ℓ_t. Theoretical histories can retain more information than the compact state summary sent to the Controller.

D.3 Controller Context and Action Validation

Decision context. The Controller receives the task contract, current incumbent and selected alternatives, recent diagnostics, a summary of progress and resource consumption, relevant experience, applicable registry entries, and current strategy preferences. Its instruction specifies the Action Package schema and asks it to choose a feasible next decision. The policy may preserve a working configuration; adaptation does not require changing every dimension at every iteration. Context limits and retrieval rules determine what information is actually visible.

Validation sequence. The runtime first parses the structured output and checks required fields and types. It then checks that component identifiers exist, input artifacts and parent references are available, and any operator preconditions are satisfied. For example, recombination requires compatible parents; a tool query requires task permission and an available tool. Finally, it checks branch, context, evaluation, and execution limits against the updated budget after charging the Controller call.

A bounded correction policy may request a repaired package or select a declared feasible fallback. Every additional model call is charged, and the configured correction limit is finite. If no valid affordable action remains, the run terminates with the best verified incumbent or the task-defined failure outcome. No malformed package authorizes arbitrary tool access or an unbounded retry loop.

Execution order. Let u_t be the context assembled from the selected query and context scope. One iteration follows

$$u_t = \mathrm{QueryContext}(d, z_t, q_t, c_t), \tag{30}$$
$$\mathcal{Y}_t = \mathrm{Propose}(d, u_t, o_t, w_t), \tag{31}$$
$$\mathcal{R}_t = \mathrm{HarnessEvaluate}(d, \mathcal{Y}_t, e_t), \tag{32}$$
$$(\mathcal{P}_{t+1}, \mathcal{D}_{t+1}, \mathcal{H}_{t+1}) = \mathrm{Retain}(\mathcal{P}_t, \mathcal{D}_t, \mathcal{H}_t, \mathcal{Y}_t, \mathcal{R}_t, \rho_t). \tag{33}$$

Each execution stage observes its resource limit and reports actual usage. The initial query is downstream of package selection: its new evidence can guide proposal generation and subsequent decisions but is not retroactively part of the Controller's input. A separate replanning call, if explicitly configured, is another charged decision and must be recorded as such.

D.4 Strategy Adaptation Rules

Adaptable state. The strategy state θ contains selection preferences, weights over registered operators, and templates used to form component instructions. A strategy version identifies the resolved choices used by each Action Package. The Adapter reads accumulated records, including successful and failed attempts, validity outcomes, observed progress, and consumed resources. It proposes changes to these preferences and templates while retaining the task and component contracts.

Trigger and update. The trigger is a configured predicate over the outcome history and elapsed iterations since the last adaptation. It may combine a stagnation window or a recurring-failure condition with a minimum adaptation interval. These windows and thresholds are task or run parameters, not universal constants. The scheduler checks that adaptation is affordable before invoking it. A proposal must preserve the schema, use registered identifiers, and satisfy any configured ranges; invalid proposals leave the previous strategy active. Accepted changes are recorded with their supporting outcomes and version.

Changes to preferences or templates are policy updates, not evidence of improvement by themselves. An Adapter update can be ineffective or harmful, so its value must be evaluated through subsequent behavior. In particular, the architecture does not assume monotonically decreasing selection error or cost-free adaptation.

Scope of modification. The Adapter cannot silently change the final objective, required tests, artifact permissions, or resource accounting. A newly introduced executable mechanism requires registration and contract validation. This keeps strategy revision distinct from unrestricted changes to runtime logic and makes its effects traceable through the action records.

D.5 Archive, Memory, and Context Construction

Candidate archive. The archive stores artifacts with their lineage and evaluation records. Retention policies may preserve high-quality candidates, diverse alternatives, or candidates relevant to unresolved failures. The best candidate that has completed all required checks remains available for return. Candidates with only intermediate evaluation are explicitly marked as provisional, and a higher provisional score alone does not displace a verified incumbent.

Experience memory. Experience entries describe observations such as a repeated failure mode, a useful local modification, or a combination that did not justify its cost. Entries retain their source evidence and scope instead of converting one outcome into an unconditional rule. A configuration with no additional experience memory still retains the current candidate and the minimal records required to execute and account for the loop.

Retrieval and context. The query policy chooses whether to use the current candidate only, inspect additional diagnostics, or retrieve relevant archive and experience entries. The context builder applies the selected scope and size limit, keeping candidate identities and evaluation provenance available. Retrieval can improve the information presented to the Controller in a later state or to the current operator after package selection; those two information paths are recorded separately.

D.6 Execution Reliability and Resource Accounting

Evaluation and final eligibility. The evaluation protocol may return scalar scores, textual diagnostics, or staged feedback. Staging can reject invalid candidates before expensive measurements, but omitted checks remain incomplete rather than passed. Final eligibility is determined by the task contract, and the returned candidate must satisfy its mandatory checks. Changing feedback richness does not change the final scoring rule.

Failure handling. Execution records distinguish malformed packages, invalid artifact edits, parse or build failures, failed checks, timeouts, and budget interruptions. Artifact updates are applied to a candidate copy so that a failed edit does not destroy the retained incumbent. Task-appropriate process isolation and time limits bound candidate execution. A failure returns a typed record with the available diagnostics and actual costs, which can inform later state summaries.

Resource ledger. For each decision, the ledger records control, query, generation, evaluation, memory-processing, and adaptation usage, including failures and bounded retries. Token counts, elapsed execution time, and API charges remain distinguishable rather than being treated as interchangeable. A run declares its controlling budget or conversion rule and checks remaining resources between stages. Any final verification or adaptation must fit within the declared allocation. A finite iteration cap additionally prevents unlimited zero-cost decision cycles.

Architecture and experimental alignment. The contracts in this appendix specify the intended executable architecture. A reported experimental configuration must identify its implemented Controller, Adapter, prompts, registry, task adapter, and accounting behavior. A rule-based controller or a descriptive evaluation-depth field that does not affect evaluator execution is a different implementation choice and must be reported as such. Appendix A.5 distinguishes functional mapping criteria from evidence of implementation equivalence; the criteria are not themselves a completed equivalence audit.

E Experimental Protocols

E.1 Benchmarks and Evaluation Pipelines

We integrate the benchmark evaluation pipelines from SkyDiscover Liu et al. (2026c) into our framework. The integration adapts candidate submission and the return of evaluation records. The study uses the task definitions and available checks from these pipelines, with the final score fields and aggregation rules specified below. All compared methods therefore use the same task-specific evaluation criteria. Instantiating an optimizer within the shared configuration space changes its optimization procedure, not the benchmark objective.

The evaluation covers 32 groups: 17 mathematical tasks, five systems tasks, three GPU Mode tasks, and one group each for Frontier-CS, ARC, Sky Festival, HotpotQA, QNN, ALE-Bench, and KernelBench. Table 14 summarizes these domains. The main results table presents seven representative benchmarks; aggregate rankings use all 32 groups in Table 15. MLA Decode is excluded.

Where a benchmark separates optimization feedback from final evaluation, this separation is retained. Within-evaluator aggregation is distinguished from the aggregation across independent optimization subtasks. For example, runtime aggregation within a GPU benchmark remains part of that evaluator and is distinct from averaging scores across independent optimization subtasks.

Score fields and aggregation. GPU Mode (VecAdd, Grayscale, and TriMul) reports $3000/t_g$, where t_g is the geometric mean execution time in microseconds after the required correctness tests; these are scaled inverse runtimes, not speedup ratios. KernelBench instead reports speedup relative to PyTorch eager. Frontier-CS averages the bounded scores over all 172 problems and divides by 100, counting failed or missing problems as zero. ARC uses final held-out test pass@2, not optimization-time cell accuracy. ALE-Bench-Lite averages private `final_performance` over ten problems without dividing by 100. HotpotQA uses exact-match percentage divided by 100. QNN uses held-out classification accuracy with 60 test examples per evaluation; five-run summaries are computed

Table 14: Benchmark domains and evaluation targets. Evaluation uses task-specific benchmark harnesses and the stated aggregation rules.

Domain	Benchmark family	Evaluation target
Math	Mathematical optimization	Numerical objectives subject to task-specific feasibility constraints.
Systems	ADRS	Workload performance, cost, and composite system objectives.
GPU	GPU Mode; KernelBench	Numerical correctness and kernel execution performance; scaled inverse runtime or eager-baseline speedup, respectively.
Algorithms	Frontier-CS; ALE-Bench-Lite	Bounded algorithmic-problem scores or private contest-performance scores over the stated problem sets.
Reasoning	ARC	Correctness of candidate transformations on benchmark inputs.
Creative	Sky Festival	Satisfaction of semantic and compositional image requirements.
Prompts	HotpotQA	Question-answering performance obtained with optimized instructions.
Quantum	QNN circuit topology	Classification performance obtained with candidate circuit structures.

before rounding. Sky Festival averages no physical execution metric: its score is the sum of seven vision-model rubric categories divided by 100.

For PRISM, one run is scored by the arithmetic mean of the per-configuration `score` outputs across the full configuration set. Table 15 reports the maximum of these run-level means over five runs. This study's aggregation is not the inverse of an averaged KV-cache pressure plus a success-rate term. PRISM scores are ranked in the higher-is-better direction used in the result matrix. Mathematical tasks use their task-specific reference-relative scores, which can exceed one; no cross-task mean of raw scores is reported.

E.2 Models, Baselines, and Budgets

The main experiments use Doubao-Seed-2.0-pro as the optimization backbone. Each method is evaluated over five independent runs with a budget of 30 optimization iterations per task. Model parameters remain fixed throughout optimization. Task-specific evaluation models, where applicable, are shared across the compared methods.

The 13 method-inspired profiles in the overall comparison are TextGrad, OPRO, ProTeGi, Reflexion, GEPA, AdaEvolve, EvoX, FunSearch, AlphaEvolve, Voyager, DSPy, Self-Refine, and LATS. The main table presents five of these profiles plus the official SkyDiscover EvoX and AdaEvolve implementations. The overall ranking includes only the 13 profiles and OptiCom; official-code entries are not additional overall methods. Their mappings into the shared configuration space and implementation differences are described in Appendix A.

The shared iteration budget controls the number of optimization rounds. It does not imply identical numbers of generated candidates, LLM calls, or tokens. Accordingly, the main comparison concerns final quality under a common iteration limit; the iteration-wise trajectories do not establish equal-time or equal-cost performance.

E.3 Aggregation and Reporting

Benchmark-level scores. Let $y_{m,t,r}$ denote the final task score of method m on subtask t in run r. For a benchmark group g containing a fixed collection of independent subtasks $\mathcal{T}_g$, we compute

$$G_{m,g,r} = \frac{1}{|\mathcal{T}_g|} \sum_{t \in \mathcal{T}_g} y_{m,t,r}. \tag{34}$$

The average uses the comparable scores specified for that group, rather than heterogeneous physical quantities. Single-task benchmarks correspond to $|\mathcal{T}_g| = 1$. Test cases evaluated jointly by a task's evaluator are not counted as separate optimization subtasks.

Mean and maximum scores. For the higher-is-better scores reported in the main table, the two summaries are

$$\mathrm{Mean}_{m,g} = \frac{1}{5}\sum_{r=1}^{5} G_{m,g,r}, \qquad \mathrm{Max}_{m,g} = \max_{r\in\{1,\ldots,5\}} G_{m,g,r}. \tag{35}$$

Both use the same five runs. For a group containing multiple subtasks, group aggregation precedes taking the maximum over runs. Thus, Max does not combine a separately selected best run from each subtask.

Overall ranking. Figure 3(a) reports two rankings computed from the same 14 configurations and 32 groups. The Max ranking uses $\mathrm{Max}_{m,g}$ and the Mean ranking uses $\mathrm{Mean}_{m,g}$. For each statistic, configurations are ranked within each group in descending score order; ties at the retained reporting precision receive the arithmetic mean of their occupied rank positions. A method's overall rank is the arithmetic mean of its 32 within-group ranks, with every group receiving equal weight regardless of its number of subtasks. The two official-code entries are excluded. Numeric zero scores remain in the ranking and are not treated as missing. Under this convention, OptiCom has average ranks 1.72 (Max) and 2.08 (Mean). The Max matrix contains all 448 profile entries; the corresponding aggregate Mean matrix is used for the Mean-based ranking.

Reporting standards. Overall ranks are computed from the retained four-decimal aggregate score matrices following standard multi-task algorithmic evaluation protocols. Rounding can create ties, especially on Cloudcast and EPLB. The main table reports five-run Max and Mean; ablation tables additionally retain the supplied standard-deviation summaries. Best-of-five performance reflects peak search capability under a fixed budget, serving as a standard evaluation metric across complex heuristic search spaces.

Relative gains. For each column of Table 2, the relative gain over the strongest compared baseline is

$$\mathrm{Gain} = \frac{s_{\mathrm{OptiCom}} - s_{\mathrm{baseline}}}{s_{\mathrm{baseline}}} \times 100\%. \tag{36}$$

The baseline is selected separately for the Mean and Max columns. These values describe relative score improvements, not percentage-point changes or statistical significance.

E.4 Ablation Configurations

The core ablations use Doubao-Seed-2.0-pro on Heilbronn Triangle, LLM-SQL, and HotpotQA. Each configuration is evaluated over five independent runs with a maximum of 30 optimization iterations. We report the mean and standard deviation of the final retained scores, rather than statistics over intermediate iterations. Task definitions and final evaluation criteria remain unchanged. The iteration cap does not impose an equal token, API-call, or monetary budget across configurations.

Core interventions. The variants modify the following parts of the optimizer:

- **Fixed composition.** The optimizer retains a predefined configuration of query, operator, evaluation, and memory mechanisms throughout the run, with the Strategy Adapter disabled. Candidates, evaluation records, and memory contents continue to update under these fixed rules. Thus, a fixed composition does not imply repeatedly generating the same candidate or ignoring new feedback.
- **Random composition.** Mechanisms are selected randomly from executable combinations in the same registry, without state-conditioned selection preferences or Strategy Adapter updates. Applicability checks and mandatory task constraints remain in force.
- **Without Strategy Adapter.** The Controller continues to select Action Packages from the current optimization state, but the initial strategy guidance, selection weights, and templates remain fixed. Immediate action selection is therefore adaptive even though its longer-term guidance is not revised.

- **Operator-only adaptation.** The Strategy Adapter is disabled, and the Controller can change only the update operator. Query, evaluation, memory, and other action settings follow fixed rules. Comparing this variant with the preceding one tests the value of allowing a broader range of optimization decisions to change.
- **Without Experience Memory.** Cross-iteration experience summaries and their retrieval are removed. The candidate archive, current evaluation feedback, and progress and budget records remain available, so candidate retention and operations requiring archived candidates remain executable. The Controller and Strategy Adapter otherwise remain enabled.

These interventions distinguish component contents from the rules governing their use. For example, freezing a memory mechanism does not prevent it from storing new observations, while removing experience memory does not remove the candidate archive. Comparisons involving the Adapter assess its contribution within the iteration-limited process, including its additional API calls.

Adaptation frequency. On Heilbronn Triangle, we compare the default event-triggered Adapter with no adaptation, periodic adaptation every five iterations, and adaptation after every iteration. No Adapter call is made after the final iteration because its output would have no subsequent optimization step to influence. The observed mean call counts are therefore 0, 10, 5, and 29, respectively. We additionally report the mean total API token consumption for each complete run, including optimization-related calls beyond candidate generation.

Feedback access. The feedback experiment on Heilbronn Triangle crosses two information conditions with the presence or absence of the Strategy Adapter. Basic feedback exposes validity status and a scalar score; rich feedback additionally exposes failure reasons and task diagnostics. The underlying evaluator and final scoring criterion remain unchanged. The intended intervention concerns information available to the optimizer, rather than changes to what constitutes a valid or high-quality solution. Results are reported with total API token consumption because richer feedback can also increase the amount of processed context.

Initialization sensitivity. We evaluate three deliberately different initial Q/O/E/M configurations on Heilbronn Triangle. The local-revision configuration uses self-only context, local candidate updates, scalar evaluation feedback, and a quality-oriented archive with recent records. The diagnosis-and-repair configuration uses environment probing, targeted repair or bottleneck-directed updates, diagnostic feedback, and structured experience. The exploration-and-recombination configuration retrieves alternative candidates, combines their structures or construction procedures, and maintains a quality-and-diversity-oriented archive. All configurations retain the same mandatory validity checks and final objective.

For each initialization, the fixed variant retains its mechanism rules throughout optimization. The adaptive variant executes the specified configuration in the first iteration and subsequently permits the Controller and Strategy Adapter to modify it. Within each comparison, the initial candidate set and evaluation protocol are held constant. These three fixed configurations are distinct from the default Fixed composition variant in Table 3; they provide additional, deliberately designed reference configurations rather than repeated measurements of that row.

F Additional Experimental Results

F.1 Complete Max-Score Results

Table 15 contains the retained Max scores for 32 groups and 14 configurations, totaling 448 entries. It replaces the earlier partial 22-task archive. The corresponding aggregate Mean matrix is used for the Mean-based ranking in Figure 3(a) but is not reproduced here to avoid duplicating another full 32-by-14 table. Unqualified baseline names denote method-inspired profiles; the two official-code comparisons appear only in Table 2. Values are reported to four decimal places and follow the task-specific higher-is-better score definitions. They should be compared within a task, not averaged across heterogeneous raw scales.

This matrix is sufficient to reproduce the reported Max-based ranks at the retained precision. It is not a replacement for the underlying five-run logs or candidate artifacts. Ties share the highest/second-

highest distinct-score annotations; ranking instead uses average occupied positions as specified in Appendix E.3.

F.2 Optimization Trajectories

Figures 7–12 complement the aggregate results with optimization trajectories on twelve mathematical benchmarks. The plots show best-so-far scores from individual archived runs of the framework profiles, rather than averages over five runs or trajectories of the official-code entries. Highlighted annotations indicate the iteration at which OptiCom reaches its final recorded best score. These trajectories reveal when improvements occur and whether early advantages persist. Iterations describe search progress rather than equal computational cost, and benchmark scores should be interpreted within each task; a score close to one does not, by itself, certify proximity to a mathematical optimum.

Early progress and final quality capture different properties. On Erdos Min Overlap, OptiCom establishes a substantial advantage within the first few iterations and subsequently makes smaller improvements, reaching a recorded score of 0.997 at iteration 16 (Figure 7a). A different pattern appears on Circle Packing Rect: several baselines obtain strong candidates before OptiCom, but OptiCom subsequently overtakes them and reaches 0.998 at iteration 9 (Figure 8a). Early progress can also fail to translate into the strongest final result. On Circle Packing, OptiCom reaches 0.968 at iteration 3, while AlphaEvolve later obtains a higher score (Figure 11a). These comparisons show why optimizer quality cannot be characterized solely by either the first few iterations or the final score: their relative importance depends on the available budget and the quality required by the task.

Temporary stagnation need not imply exhausted improvement opportunities. On Heilbronn Triangle, OptiCom remains near 0.774 for several iterations before improving to 0.961 at iteration 13 (Figure 9a). On Matmul, it initially trails several methods at approximately 0.582, then improves to 0.800 and finally 0.842 at iteration 11, matching the strongest final result shown (Figure 12a). These trajectories illustrate that an optimizer can recover from an unproductive interval or an initially unfavorable position. They motivate retaining alternatives and reconsidering optimization decisions when progress stalls, rather than treating a short plateau as evidence that further search is unproductive. The score histories alone do not identify which mechanism produced each improvement, but they demonstrate the importance of evaluating decisions over the remaining optimization horizon.

The useful refinement horizon varies across tasks. Some trajectories are dominated by large early gains, whereas others continue to accumulate small improvements after reaching a strong candidate. For example, OptiCom reaches a high score early on both Minimizing Max Min Dist tasks, but its final recorded improvements occur at iterations 26 and 13, respectively (Figure 10). On Circle Packing Rect and First Autocorr Ineq, the full-scale curves appear nearly indistinguishable among several methods, while the insets expose meaningful differences in the timing and magnitude of subsequent refinements (Figure 8). Similarly, Sums Diffs Finite Sets shows incremental progress up to iteration 20, with OptiCom finishing close to the strongest baseline (Figure 11b). These patterns motivate adapting the balance between exploration and refinement to recent progress and remaining budget. They also show why the iteration of the final improvement is insufficient as a standalone efficiency measure: a late, small refinement may follow a much earlier attainment of practically useful quality.

Adaptive composition does not dominate every task. The additional trajectories expose limitations alongside favorable results. EvoX exceeds OptiCom on both autocorrelation tasks (Figures 7b and 8b), and several methods attain higher final scores on Hexagon Packing 12 (Figure 12b). On the latter task, OptiCom improves to approximately 0.764 but does not close the gap to the strongest alternatives. Thus, continued improvement within a run does not necessarily imply competitive final performance. These cases are consistent with the distinction in Section 4: making complementary mechanisms available creates opportunities, but realizing their value also requires selecting suitable actions. The trajectories do not distinguish insufficient alternatives from inaccurate selection or

Table 15: Max scores over five runs on all 32 benchmark groups. Higher is better within each task. Baseline columns are method-inspired framework profiles. Bold and underlining mark the highest and second-highest distinct values at four-decimal precision. The final row gives average within-task ranks with average ranks for ties; lower is better. Official-code entries are excluded.

Task	TextGrad	OPRO	ProTeGi	Reflexion	GEPA	AdaEvolve	EvoX	FunSearch	AlphaEvolve	Voyager	DSPy	Self-Refine	LATS	OptiCom
Circle packing	0.9256	0.8892	0.8806	0.7652	0.8607	0.8259	0.8833	0.8277	**1.0003**	0.6818	0.7830	0.7547	0.9282	0.9682
Circle packing (rect.)	0.9869	0.9895	0.9968	0.9964	0.9896	0.9876	0.8577	0.9895	0.9913	0.9967	0.9879	0.7342	0.9975	**0.9982**
Erdos minimum overlap	0.7618	0.7920	0.7633	0.7637	0.7618	0.7618	0.7600	0.7530	0.8204	0.7695	0.7637	0.0000	0.8044	**0.9972**
Autocorrelation inequality 1	0.9939	0.9950	0.9945	0.9928	0.9920	0.9945	**0.9971**	0.9948	0.9957	0.9925	0.9949	0.9934	0.9960	0.9966
Heilbronn convex (13)	0.6415	0.7213	0.4706	0.7141	0.5436	0.4591	0.8242	0.0878	0.0425	0.4211	0.7136	0.5160	0.7147	**0.9386**
Heilbronn convex (14)	0.7384	0.5486	0.6956	0.6117	0.6052	0.3543	0.7236	0.6689	0.5780	0.4616	0.6672	0.6592	0.4531	**0.8591**
Heilbronn triangle	0.8557	0.5984	0.8282	0.6401	0.7191	0.7685	0.9431	0.6508	0.7993	0.6060	0.9145	0.7151	0.8874	**0.9608**
Hexagon packing (11)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	**0.2399**
Hexagon packing (12)	0.7884	**0.8942**	0.6706	0.6838	0.6362	0.7167	0.8871	0.6555	0.6570	0.6362	0.6570	0.7167	0.8731	0.7642
Matrix multiplication	0.8000	0.8000	0.8000	0.8000	0.8000	0.8000	0.8000	0.8000	**0.8421**	0.0000	0.0000	0.8000	0.0000	**0.8421**
Min-max distance (2D)	0.8675	0.7159	0.8475	0.9989	0.8559	0.7167	0.7161	0.8718	0.8741	0.7161	0.9494	0.8625	0.9942	**0.9996**
Min-max distance (3D)	0.9415	0.7307	0.8803	0.9083	0.8803	0.8994	0.8803	0.9170	0.8803	0.6943	0.6943	0.9031	0.8778	**0.9968**
Autocorrelation inequality 2	0.9549	0.9827	0.9879	0.8344	0.9606	0.9864	**1.0135**	0.8347	0.9833	0.9549	0.9574	0.9794	0.9466	0.9872
Signal processing	0.5012	0.5298	0.5354	0.5124	0.5442	**0.6086**	0.5558	0.5183	0.5232	0.5209	0.5914	0.5343	0.5147	0.5484
Sums/differences of finite sets	0.8880	0.0000	0.9384	0.9497	0.9295	0.9066	0.0000	0.8667	**0.9531**	0.9188	0.0000	0.9242	0.8800	0.9525
Autocorrelation inequality 3	**0.9952**	0.9913	0.9941	0.9940	0.9951	0.9912	0.9939	0.9919	0.9920	0.9934	0.9921	0.9924	0.9938	**0.9952**
Uncertainty inequality	0.8956	0.8958	0.9120	0.8804	0.8995	0.9034	0.9133	0.8933	0.9090	0.8804	0.9133	0.8960	0.9016	**0.9195**
Cloudcast	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	0.0010	**0.0011**
EPLB	0.1274	0.1274	0.1274	0.1275	0.1275	0.1275	0.1274	0.1275	0.1274	0.1266	0.1275	0.1275	0.1274	**0.1344**
LLM-SQL	0.6769	0.6488	0.6299	0.6385	0.6589	0.6190	0.6835	0.5829	0.5876	0.6451	0.1146	0.6554	0.6863	**0.6980**
PRISM	0.8630	0.9584	0.9619	0.0389	0.9603	0.9613	0.1734	0.9614	0.9589	0.9614	**0.9620**	0.9589	**0.9620**	0.9612
Transaction scheduling	3875.9690	3984.0637	3759.3985	3937.0079	3906.2500	4016.0643	3663.0037	3690.0369	3952.5692	3690.0369	3690.0369	3690.0369	3690.0369	**4032.2581**
Grayscale	42.1582	58.7491	65.3024	38.9915	51.4823	**69.1047**	47.6258	55.8310	62.4172	36.5749	44.8931	53.2185	60.7564	68.3296
TriMul	2.3921	2.3773	2.5410	2.4127	2.4370	2.5730	2.5930	2.3875	2.5912	2.5691	2.5131	2.4918	2.5760	**2.6540**
VecAdd	89.4521	120.3184	95.7632	112.4891	86.1298	131.0543	127.3412	98.6504	115.8237	109.4310	91.2785	124.7659	85.9934	**133.2014**
HotpotQA	0.3452	0.3128	0.4410	0.3891	0.3810	0.3970	0.3860	0.3605	0.3274	0.3719	0.3386	0.3540	0.4130	**0.5130**
ARC	0.3142	0.3875	0.4730	0.3421	0.3960	0.4130	0.4610	0.3956	0.3208	0.3764	0.3559	0.3093	0.4570	**0.5030**
Frontier-CS	0.5381	0.6124	0.6320	0.5709	0.6130	0.6210	0.6260	0.5042	0.5933	0.5487	0.5215	0.6078	0.6270	**0.6930**
Sky Festival	0.2451	0.3102	0.1987	0.2845	0.2134	**0.3291**	0.2678	0.2950	0.1842	0.2319	0.2765	0.3014	0.2056	0.2983
KernelBench	0.6234	0.8412	1.0539	0.5127	0.9845	0.7321	1.0892	0.6750	0.9104	0.5899	0.8876	0.7954	0.6481	**1.1123**
ALE-Bench	1842.1593	1950.4821	1715.3094	1888.7512	1921.6045	1795.8236	1913.1962	1973.9548	1742.0671	1811.3925	1905.7482	1768.5139	1837.9264	**1991.2057**
QNN	0.7833	0.8167	0.8833	0.7667	0.8000	0.8333	0.8167	0.8333	0.8000	0.8333	0.7667	0.8333	0.8500	**0.9000**
Average rank	8.703125	8.015625	6.453125	8.625000	7.875000	7.000000	6.328125	8.875000	7.562500	10.078125	8.453125	8.468750	6.843750	**1.718750**

inadequate feedback; resolving those explanations requires controlled comparisons or action-level records.

Taken together, the trajectories support a design centered on evolving optimization states: preserve strong candidates, retain the ability to make further improvements after stagnation, and adjust the search effort as the remaining opportunities change. This interpretation connects to the cumulative opportunity–loss decomposition in Section 4, under which intermediate decisions matter through their contribution to the final outcome. The plots illustrate the resulting temporal patterns without directly estimating Δ_t or ℓ_t, or attributing the observed gains to an individual component.

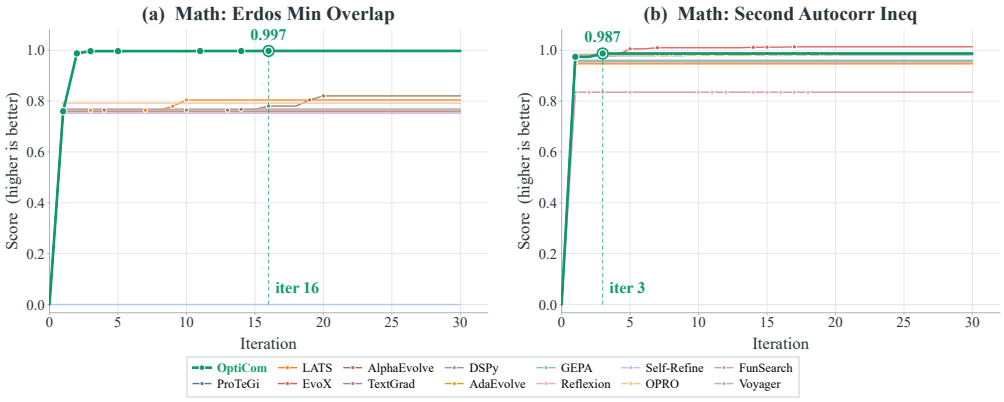

Figure 7: Archived best-so-far scores on **(a)** Erdos Min Overlap and **(b)** Second Autocorr Ineq. OptiCom establishes an early advantage on Erdos Min Overlap, whereas EvoX obtains a higher final score on Second Autocorr Ineq. Highlighted annotations mark the final recorded best score of OptiCom and its corresponding iteration.

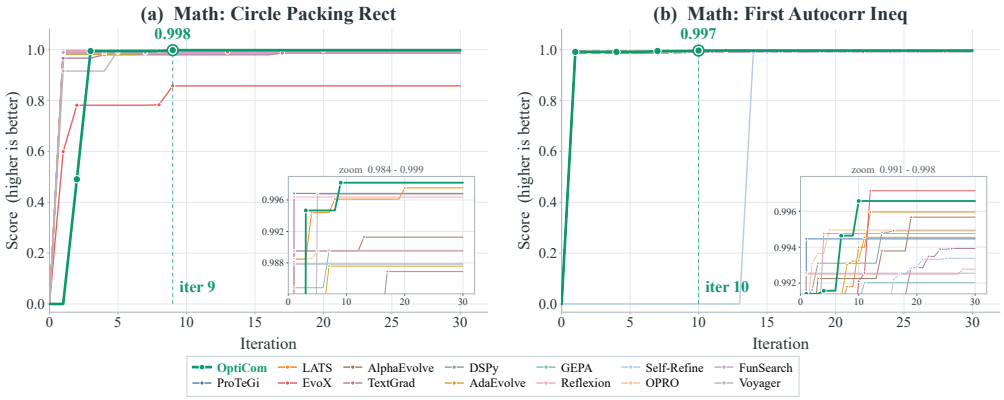

Figure 8: Archived best-so-far scores on **(a)** Circle Packing Rect and **(b)** First Autocorr Ineq. Insets reveal differences among high-scoring candidates that are difficult to distinguish on the full vertical scale. OptiCom achieves the highest final score on Circle Packing Rect, while EvoX finishes slightly higher on First Autocorr Ineq.

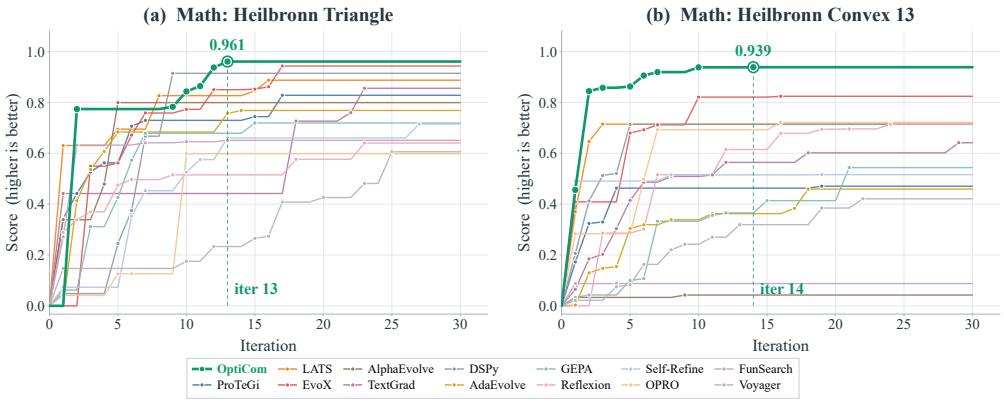

Figure 9: Archived best-so-far scores on **(a)** Heilbronn Triangle and **(b)** Heilbronn Convex 13. On Heilbronn Triangle, OptiCom resumes improvement after an initial plateau and reaches 0.961 at iteration 13. On Heilbronn Convex 13, successive improvements produce a final recorded score of 0.939 at iteration 14.

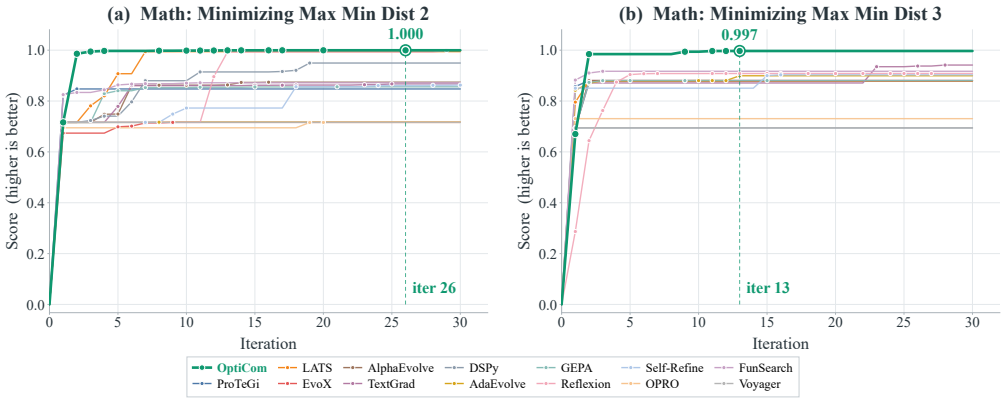

Figure 10: Archived best-so-far scores on **(a)** Minimizing Max Min Dist 2 and **(b)** Minimizing Max Min Dist 3. Large early improvements are followed by smaller refinements, with the final recorded improvements of OptiCom occurring at iterations 26 and 13, respectively. The displayed score of 1.000 is rounded and does not constitute an optimality certificate.

F.3 Cross-Backbone Robustness

Table 16 reproduces the five-run cross-backbone summaries from Table 4. All optimization-related LLM calls use the listed backbone, with the remaining configuration and evaluation procedures unchanged. The experiment therefore replaces the optimization backbone as a whole rather than the candidate generator alone.

To characterize the contribution of adaptation within each backbone, Table 17 reports the difference between the full framework and the variant without the Adapter. The difference is positive in every backbone–task pair. Relative to the corresponding no-Adapter mean, the gains range from approximately 23.1% to 33.7% on Heilbronn Triangle and from 11.1% to 18.1% on HotpotQA.

The effect is not restricted to a backbone with a particularly weak no-Adapter result. On Heilbronn Triangle, the four alternative backbones already exceed the Doubao no-Adapter mean, yet each

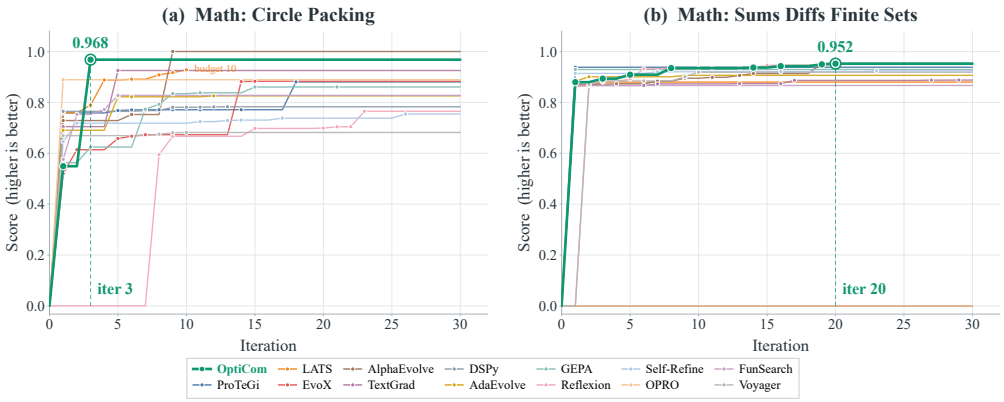

Figure 11: Archived best-so-far scores on **(a)** Circle Packing and **(b)** Sums Diffs Finite Sets. OptiCom makes a large early improvement on Circle Packing, but AlphaEvolve later achieves a higher score. On Sums Diffs Finite Sets, smaller improvements accumulate until iteration 20. The Circle Packing panel marks the shorter recorded budget of LATS.

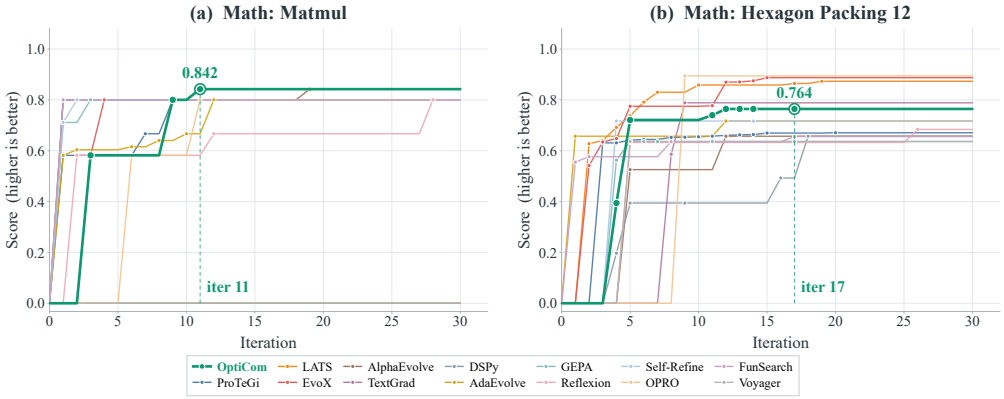

Figure 12: Archived best-so-far scores on **(a)** Matmul and **(b)** Hexagon Packing 12. On Matmul, OptiCom overcomes an initial plateau and reaches 0.842 at iteration 11, matching the strongest final score shown. On Hexagon Packing 12, improvements to 0.764 remain insufficient to match several baselines, illustrating a limitation of the observed optimization trajectory.

still benefits from enabling adaptation. Conversely, the size of the gain does not follow a common ordering across tasks: Doubao has the largest absolute gain on Heilbronn Triangle, whereas Claude Opus 4.6 has the largest on HotpotQA. The evidence therefore favors a benefit that persists across the tested backbones, rather than a monotonic relationship between backbone performance and the value of adaptation.

The full framework also has lower observed standard deviations in all ten comparisons. On HotpotQA, standard deviations range from 0.010 to 0.019 with the Adapter, compared with 0.037 to 0.059 without it. This pattern is consistent with more repeatable outcomes in these runs, although five-run summaries alone do not establish statistical significance or a universal stability guarantee.

Finally, this experiment directly evaluates the cross-backbone contribution of the Strategy Adapter, not every individual Q/O/E/M mechanism. Together with the core ablations, it supports the portability of the proposed design while leaving component-specific transfer effects unresolved. Since total

Table 16: Cross-backbone evaluation over five runs per configuration, with a maximum of 30 iterations. All optimization-related LLM calls use the listed backbone. Bold indicates the higher mean within each backbone–task pair.

	Heilbronn Triangle		HotpotQA	
Backbone	**Full OptiCom**	**w/o Adapter**	**Full OptiCom**	**w/o Adapter**
Doubao-Seed-2.0-pro	$\mathbf{0.953 \pm 0.008}$	0.713 ± 0.027	$\mathbf{0.501 \pm 0.017}$	0.451 ± 0.058
GPT-5.5	$\mathbf{0.987 \pm 0.002}$	0.796 ± 0.011	$\mathbf{0.553 \pm 0.012}$	0.471 ± 0.037
Claude Opus 4.6	$\mathbf{0.973 \pm 0.007}$	0.787 ± 0.012	$\mathbf{0.554 \pm 0.010}$	0.469 ± 0.041
GLM-5.3	$\mathbf{0.969 \pm 0.008}$	0.783 ± 0.014	$\mathbf{0.523 \pm 0.019}$	0.461 ± 0.059
Kimi-K3	$\mathbf{0.974 \pm 0.004}$	0.791 ± 0.011	$\mathbf{0.547 \pm 0.016}$	0.465 ± 0.053

Table 17: Absolute mean-score gains from enabling the Strategy Adapter, computed as Full OptiCom minus w/o Adapter using Table 4. These are within-backbone comparisons under the same 30-iteration cap.

Backbone	**Heilbronn Triangle**	**HotpotQA**
Doubao-Seed-2.0-pro	$+0.240$	$+0.050$
GPT-5.5	$+0.191$	$+0.082$
Claude Opus 4.6	$+0.186$	$+0.085$
GLM-5.3	$+0.186$	$+0.062$
Kimi-K3	$+0.183$	$+0.082$

computational expenditure is not matched across configurations or providers, the results should not be interpreted as a ranking of backbone cost efficiency.

F.4 Additional Ablation Details

How frequently should strategy adaptation occur? Table 18 shows that all three adaptation schedules outperform the no-adaptation variant on Heilbronn Triangle. The default event-triggered policy obtains a mean score of 0.953, compared with 0.954 when adaptation is invoked after every nonterminal iteration. It uses approximately 65.5% fewer Adapter calls, although the reduction in total token consumption is about 1.37%. Thus, the evidence supports obtaining similar observed mean quality without invoking the Adapter at every iteration, rather than a substantial reduction in total computation. Every-iteration adaptation has the smallest observed standard deviation, so the results do not establish that less frequent adaptation is uniformly preferable.

Periodic adaptation every five iterations obtains a lower mean score of 0.939. However, it also makes fewer Adapter calls than the event-triggered variant. This comparison therefore combines differences in timing and frequency and does not isolate the advantage of state-dependent triggering at a matched number of updates.

Table 18: Strategy adaptation frequency on Heilbronn Triangle. Scores report mean $\pm$ standard deviation over five runs. Adapter calls and total API tokens are run averages; K denotes one thousand tokens. Updates are not invoked after the final iteration.

Configuration	**Final score**	**Adapter calls**	**Total tokens**
No adaptation	0.713 ± 0.027	0	853K
Event-triggered adaptation	0.953 ± 0.008	10	865K
Periodic adaptation ($k = 5$)	0.939 ± 0.010	5	861K
Every-iteration adaptation	0.954 ± 0.003	29	877K

Feedback access with and without strategy adaptation. Table 19 shows that rich feedback improves mean scores both with and without the Adapter, by 0.056 and 0.034, respectively. Both

configurations also exhibit lower standard deviations with diagnostic feedback. These observations suggest that information beyond a scalar score can benefit the optimization process even when strategy guidance is not updated; because generation and other downstream decisions also receive this information, the comparison does not isolate immediate Controller selection.

The Adapter remains beneficial under basic feedback: its mean-score advantage is 0.218, compared with 0.240 under rich feedback. Thus, diagnostic information is helpful but is not a prerequisite for the observed adaptation benefit. The difference between these two gains is 0.022; with five runs and substantial variability in the basic-feedback condition, we treat this as descriptive evidence rather than an established interaction effect. Token consumption varies across the feedback and adaptation conditions, so the comparison concerns performance under a common iteration cap rather than equal information-processing cost. It does not establish an upper bound on adaptive performance determined by evaluator quality.

Table 19: Feedback access on Heilbronn Triangle over five runs. Both conditions use the same evaluator and final scoring rule. Scores are mean $\pm$ standard deviation; token counts are mean total API consumption per run.

	Final score		**Total tokens**	
Feedback	**Full OptiCom**	**w/o Adapter**	**Full OptiCom**	**w/o Adapter**
Basic	0.897 ± 0.057	0.679 ± 0.126	809K	799K
Rich	0.953 ± 0.008	0.713 ± 0.027	865K	853K

Observed sensitivity to the initial composition. Table 20 compares fixed and adaptive optimization from three initial configurations. Adaptive optimization improves the mean score in every case, with absolute gains of 0.141, 0.130, and 0.094 for local revision, diagnosis and repair, and exploration and recombination, respectively. Notably, adaptation also improves over the strongest fixed configuration, whose mean score is 0.857. The benefit is therefore not confined to the default fixed reference used in the core ablation.

The range of mean scores across initializations decreases from 0.044 for fixed compositions to 0.005 for adaptive runs. This suggests that subsequent decisions can reduce the influence of the initial mechanism choice on terminal quality. Because only the first iteration is constrained in the adaptive variants, the result concerns sensitivity to initialization rather than recovery from a prolonged unsuitable strategy. Similar terminal means also do not imply identical intermediate decisions, resource use, or statistical equivalence across initializations.

Table 20: Initialization sensitivity on Heilbronn Triangle. Each entry reports the mean and standard deviation over five runs with a maximum of 30 iterations. Adaptive OptiCom begins with the corresponding initial composition and can adjust it after the first iteration.

Initial configuration	**Fixed**	**Adaptive OptiCom**
Local-revision-first	0.813 ± 0.017	$\mathbf{0.954 \pm 0.008}$
Diagnosis-and-repair-first	0.826 ± 0.012	$\mathbf{0.956 \pm 0.008}$
Exploration-and-recombination-first	0.857 ± 0.011	$\mathbf{0.951 \pm 0.008}$

G Limitations and Broader Impact

The profile comparison does not establish equivalence to all original optimizers, and the official-code checks cover only two methods on seven tasks. The motivation study is limited to one benchmark and three constructed situations per state. Complete per-run records are unavailable for the overall matrix, limiting uncertainty analysis. Common iteration caps do not control total computation, and the token-aware case study shows that substantial additional search cost can occur after the final improvement. The experiments do not establish long-horizon scaling, monotonic adaptation, causal effects of individual action changes, or satisfaction of the near-optimal selection assump-

tion. Evaluation quality remains limited by each harness, including model-judge variability and benchmark-specific validity checks.