SPECEVO: SPECULATIVE EVOLUTION WITH LARGE LANGUAGE MODELS FOR COST-EFFICIENT SCIENTIFIC DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-driven evolutionary search is a promising paradigm for scientific discovery, but existing methods remain costly and slow, often applying expensive reasoning uniformly while underusing evidence from failed, redundant, and saturated trials. We introduce *Speculative Evolution* (SPECEVO), a multi-LLM framework that coordinates lightweight and frontier models through continual consolidation of search evidence. Lightweight models power both parallel Speculators, which generate and refine candidate programs, and an Advisor, which distills accumulated successes, failures, and saturated directions into persistent guidance for subsequent exploration. At sparse checkpoints, a frontier-model Navigator inspects the evolving population and intervenes through synthesis, surgical refinement, or reframing according to search stagnation. This *speculate-then-consolidate* workflow delegates routine exploration and evidence consolidation to inexpensive models while reserving frontier reasoning for strategic interventions whose benefits extend across many subsequent low-cost evaluations. Across 176 tasks spanning four scientific and algorithmic discovery domains, SPECEVO matches or outperforms frontier-model baselines while reducing API cost by up to 80%, achieves up to $1.5\times$ greater evaluation efficiency than cost-efficient baselines on equation discovery, and improves over the strongest CO-Bench baseline by 3.8%. These results support trajectory-conditioned model coordination as an effective approach to cost-efficient scientific discovery.

1 INTRODUCTION

Recent advances in large language models have enabled a new paradigm of evolutionary search for automated scientific discovery (Yamada et al., 2025; Liu et al., 2026b). LLMs act as semantic variation operators, proposing meaningful modifications to candidate programs that are refined through execution-based evaluation and selection (Liu et al., 2026a; Wu et al., 2024). Such approaches have achieved strong results across mathematical discovery, combinatorial optimization, and systems engineering (Hieu et al., 2026; Cemri et al., 2026; Novikov et al., 2025; Liu et al., 2024b).

Despite their strong results, many systems nevertheless use a single backbone for candidate generation and higher-level search decisions (Agrawal et al., 2025; Cemri et al., 2026; Liu et al., 2026b). When that backbone is a frontier model, routine proposals repeatedly incur expensive inference. Figure 1 illustrates the resulting performance–cost trade-off: using lightweight backbones lowers API expenditure but can substantially reduce solution quality in the evaluated baselines. Lightweight models offer inexpensive exploration, but without reusable guidance they can repeatedly revisit similar strategies or reproduce implementation errors (Dat et al., 2025; Liu et al., 2024a). Merely generating more affordable proposals therefore does not ensure sustained search progress. Heterogeneous-model approaches address part of this problem by assigning different search operations to different models or transferring a population from lightweight exploration to frontier-model refinement (Tanveer, 2026; Luo et al., 2026a; Huang et al., 2026). These approaches motivate a complementary question: how can accumulated successes, failures, and signs of saturation continually inform both subsequent lightweight proposals and the scope of frontier-model intervention?

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.

To address these gaps, we introduce SPECEVO, a multi-LLM evolutionary framework organized around a workflow we call *speculate-then-consolidate* (StC). StC repeatedly converts accumulated search evidence into guidance and strategic interventions that inform subsequent low-cost exploration; SPECEVO provides its concrete realization through specialized model roles and shared search state. Consolidation occurs periodically while lightweight exploration continues. A central design choice is to distinguish the programs retained for further search from the evidence retained for learning. An adaptive MAP-Elites-style archive maintains one elite per structural–semantic niche, allowing programs outside the global top ranks to remain available when they improve their own niches. A separate history records all evaluated attempts, including rejected programs and execution failures. Together, these structures expose which strategies remain competitive, which directions consume evaluations without progress, and which implementation errors recur. This evidence supports both revisable guidance for lightweight models and search-state-dependent choices about frontier intervention. Across model configurations, SPECEVO improves the performance–cost trade-off over methods built around a single frontier model. Its advantages over cost-efficient allocation strategies persist under matched model-call and evaluation budgets, supporting the value of coordinating model use through accumulated search evidence. Ablations and search-trajectory analyses further support the importance of preserving diverse candidates, adapting intervention scope to stagnation, and revising guidance as evidence accumulates. Our main contributions are:

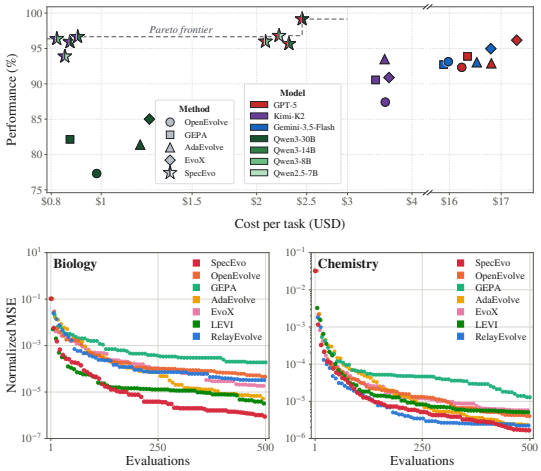

Figure 1: **Top:** Performance vs. API cost on mathematical discovery tasks; SPECEVO forms the Pareto frontier. **Bottom:** Best-so-far NMSE on LSR-Synth Biology and Chemistry over 500 evaluations.

- We introduce SPECEVO, an asynchronous realization of StC with parallel lightweight Speculators, a lightweight Advisor, and a sparsely invoked frontier-model Navigator.
- We develop coordinated feedback mechanisms that combine failure- and niche-progress-aware guidance with a stagnation signal controlling parent sampling and selecting among synthesis, surgical refinement, and reframing at fixed Navigator checkpoints.
- We provide a systematic evaluation on 176 tasks spanning mathematical discovery, systems optimization, combinatorial optimization, and scientific equation discovery, combining comparisons under matched budgets with controlled model-allocation studies, component ablations, and analyses of search dynamics.

2 RELATED WORK

LLM-Guided Evolutionary Search. FunSearch and AlphaEvolve demonstrate how execution-based selection can turn LLM-generated program variations into mathematical and systems discoveries (Romera-Paredes et al., 2024; Novikov et al., 2025). Subsequent work addresses two complementary challenges within this loop. The first is preserving useful alternatives: OpenEvolve combines island evolution with quality-diversity archiving, while SeaEvo organizes search around persistent strategy-level memory (Sharma, 2025; Luo et al., 2026b). These designs retain alternatives that can support exploration beyond the current best program. The second is using evaluation feedback to improve search decisions: GEPA derives reflective updates and maintains complementary candidates through Pareto selection, AdaEvolve adapts exploration and resource allocation to observed progress, and EvoX evolves the search policy itself (Agrawal et al., 2025; Cemri et al., 2026; Liu et al., 2026b). SPECEVO builds on these principles, using archive- and trajectory-level evidence to coordinate lightweight exploration, guidance revision, and frontier intervention.

Cost-Efficient LLM Allocation. Cost-efficient LLM systems commonly coordinate cheap and strong models through cascades or adaptive routing (Chen et al., 2023b; Aggarwal et al., 2024; Ding et al., 2024; Ong et al., 2025; Zhang et al., 2026). SCOPE uses query-level partial evaluations to select a fixed model assignment for each module of a compound system under a quality constraint (Huang et al., 2026). While reducing configuration-search cost, it does not adapt allocation to an evolving search state. Within evolutionary search, LEVI combines diversity preservation and proxy evaluation with role-aware routing, assigning local refinements to cheap models and paradigm shifts to a frontier model (Tanveer, 2026). However, the frontier model remains tied to a predefined role, while its error-oriented advice does not use broader progress and saturation evidence to adapt interventions. RelayEvolve develops multiple cheap-model trajectories and performs a one-time handoff of a quality-diverse population to strong-model refinement (Luo et al., 2026a). Afterward, lightweight exploration ends, and evidence from discarded or exhausted trajectories is not consolidated into reusable guidance. SPECEVO instead continually consolidates accumulated evidence to guide both lightweight exploration and state-dependent frontier-model intervention.

3 METHODOLOGY

Figure 2 summarizes how SPECEVO initializes its population and coordinates the subsequent search, while Appendices D and H provide the complete algorithms and prompt templates, respectively.

3.1 MOTIVATION: FROM SPECULATIVE DECODING TO SPECULATIVE EVOLUTION

Speculative decoding uses lightweight drafts and parallel verification by a stronger model to amortize inference while preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023a). Related evolutionary methods likewise reserve stronger models for difficult mutations or structural changes (Ray et al., 2026; Tanveer, 2026). In SPECEVO, candidate quality is established by an executable evaluator, while accumulated outcomes inform future proposals. The StC workflow therefore transfers the principle of asymmetric computation to search trajectories: inexpensive exploration supplies evidence, and periodic consolidation turns that evidence into guidance and strategic proposals for subsequent exploration.

Figure 2 separates population initialization on the left from the ongoing StC loop on the right. Initialization proceeds from seed generation through lightweight diversification to population embedding. The resulting archive supports continuous Speculator exploration, while the Navigator above and the Advisor below provide strategic proposals and reusable guidance at slower cadences. Once the prescribed budget is exhausted, the search terminates and the highest-scoring program in the final archive is returned. We next describe the shared search state and each functional role.

Initialization. The **Generate seed programs** block in Figure 2 provides diverse, verified starting hypotheses. The Navigator uses the frontier model $\mathcal{M}_{\mathrm{N}}$ to generate $M{=}5$ seed programs sequentially, conditioning each call on the strategies and evaluation outcomes of earlier seeds. The prompt asks the model to identify their algorithmic paradigms and propose a different modeling, decomposition, construction, or search strategy. In the subsequent **Generate variants** block, the Speculator pool uses the lightweight model $\mathcal{M}_{\mathrm{S}}$ to expand each executable seed into $V{=}20$ inexpensive variants in parallel, drawing on $n_{\mathrm{insp}} \in \{1, 2\}$ other seeds as inspiration. We initialize the behavioral archive from these evaluated programs using the representation and admission rule described next.

Behavioral archive. The **Embedded population** block supplies the structural–semantic representation used by the **Adaptive archive**. We maintain $\mathcal{A}_t$ as a MAP-Elites-style archive (Mouret & Clune, 2015) holding at most $K{=}50$ incumbents, one per *niche*: a region of descriptor space within which candidates compete on fitness. This preserves competitive programs across regions rather than retaining only the globally highest scores. At descriptor fit r, each program is represented as

$$\mathbf{z}_r(p) = \left[\mathcal{Z}_r^{\mathrm{ast}}\big(\phi(p)\big) \;\big\|\; \mathcal{Z}_r^{\mathrm{sem}}\big(\mathbf{U}_r^{\top}\,(\mathbf{e}(p) - \bar{\mathbf{e}}_r)\big)\right], \tag{1}$$

where $\phi(p)$ contains eight statistics of the code's abstract syntax tree (AST), such as loop counts and nesting depth, each transformed by $\log(1+x)$ to reduce sensitivity to unusually large programs. The description embedding $\mathbf{e}(p)$ is centered and projected onto eight leading PCA directions to compactly represent the strategy description $d(p)$. The operators $\mathcal{Z}_r^{\mathrm{ast}}$ and $\mathcal{Z}_r^{\mathrm{sem}}$ standardize each coordinate of the AST and projected-embedding blocks, respectively, using current archive means and standard deviations, making their scales comparable in Euclidean distance (Appendix D.1).

Problem setup. Given a task $\mathcal{T}$ with a space $\mathcal{P}$ of executable candidate programs and an evaluator $F : \mathcal{P} \to \mathbb{R}$, we aim to discover $p^\star = \arg\max_{p\in\mathcal{P}} F(p)$ under a prescribed wall-clock, API-cost, or evaluation budget. At evaluation step t, we maintain two shared structures: a behavioral archive $\mathcal{A}_t$ retaining a bounded set of competitive programs, and a search history $\mathcal{H}_t$ recording every evaluated attempt, including rejected candidates and execution failures. Each stored program p is paired with a natural-language description $d(p)$ of its strategy, supplied by the generating model.

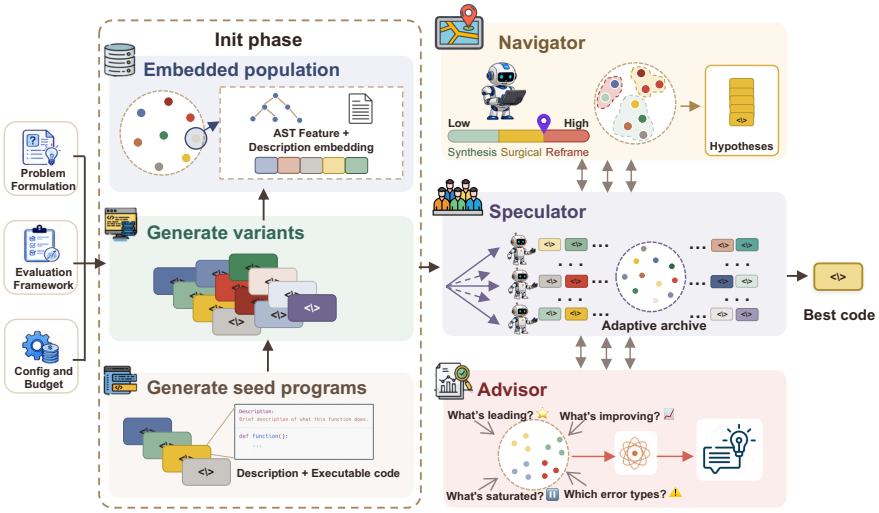

Figure 2: Overview of SPECEVO. Initialization (left) generates M seed programs, expands each executable seed into V lightweight variants, and embeds the evaluated population to initialize the archive. During evolution (right), W lightweight-model Speculators continuously explore the archive. The frontier-model Navigator (top) intervenes every Δ_N completed evaluations, while the lightweight-model Advisor (bottom) updates reusable guidance every Δ_A evaluations.

3.2 THE SPECULATE-THEN-CONSOLIDATE WORKFLOW

While fewer than K slots are occupied, valid candidates are admitted into new slots. At capacity, the first descriptor fit defines sites $\mathbf{c}_i = \mathbf{z}_r(p_i)$ whose Voronoi cells form the niches. A candidate is then assigned to its nearest site and admitted only if it strictly improves that niche's incumbent. Replacement leaves the site and descriptor map fixed until the next refit, stabilizing candidate assignment. After initialization, every Δ_C=50 completed evaluations, we refit PCA and normalization statistics on the current archive, recompute incumbent descriptors, and re-anchor the sites to follow the evolving population. Slot indices support progress tracking, although niche boundaries can change (Appendix D). Table 3 shows the effect of refitting on overall search performance.

Stagnation signal. The low-to-high scale in the Navigator block represents search stagnation: s_t increases toward 1 when either global-best improvements or archive admissions become infrequent. We use this signal to adjust both parent selection and the scope of Navigator intervention:

$$s_t = \max\Big\{ \min\big(1, \tfrac{t-\tau_t^{\text{best}}}{H_g}\big),\ \min\big(1, \tfrac{t-\tau_t^{\text{adm}}}{H_l}\big) \Big\}, \tag{2}$$

where τ_t^{best} and τ_t^{adm} denote the evaluation steps of the most recent global-best improvement and archive admission, respectively, with normalization horizons H_g=100 and H_l=20. Each term measures a normalized waiting time, capped at one. Taking their maximum makes $s_t = 1$ after 100 evaluations without a global-best improvement or 20 without an archive admission.

Speculators: continuous exploration. We use W asynchronous Speculators (W=4 in our experiments), which continuously generate candidates using $\mathcal{M}_{\text{S}}$. Sampling concentrates on leading elites when the search is progressing, while increasing stagnation broadens exploration toward lower-ranked archive regions. Whenever a worker becomes idle, it selects a parent according to its fitness

rank r_i under a Zipf distribution whose exponent adapts to stagnation (Piantadosi, 2014):

$$\Pr(p_i \mid \mathcal{A}_t) = \frac{r_i^{-\beta_t}}{\sum_{j=1}^{|\mathcal{A}_t|} r_j^{-\beta_t}}, \qquad \beta_t = \beta_{\min} + (1 - s_t)(\beta_{\max} - \beta_{\min}), \tag{3}$$

with $(\beta_{\min}, \beta_{\max}) = (0.3, 2.0)$. Thus, β_t decreases from 2.0 to 0.3 as stagnation increases, flattening the distribution while retaining a preference for higher ranks. We use rank-based selection to avoid dependence on task-specific score scales. The worker then inserts the selected parent into one of six uniformly sampled variation prompts: general mutation, focused single-weakness repair, mechanism replacement, analysis-targeted mutation, structural crossover, and component-level crossover. Figure 8 and Table 3 show that using these complementary prompts sustains higher code diversity and yields better performance than a single general mutation prompt. For crossover, the worker draws the second parent from a different niche whenever possible. Every proposal is executed, checked against the archive admission rule, and recorded in $\mathcal{H}_t$ whether or not it is admitted.

Navigator: sparse, state-dependent intervention. At each Navigator checkpoint, the thresholds γ_1 and γ_2 partition the stagnation signal into three levels of increasing intervention scope:

$$m_t = \begin{cases} \textsc{Synthesis}, & s_t \le \gamma_1, \\ \textsc{Surgical}, & \gamma_1 < s_t \le \gamma_2, \\ \textsc{Reframe}, & s_t > \gamma_2, \end{cases} \qquad (\gamma_1, \gamma_2) = (0.4, 0.7). \tag{4}$$

These thresholds correspond to 40% and 70% of either waiting-time horizon. When progress remains steady, the Navigator uses SYNTHESIS to combine complementary mechanisms from competitive niches into a coherent program. Under moderate stagnation, it uses SURGICAL for a targeted structural correction to the current champion, while under stronger stagnation it uses REFRAME to introduce a strategy family absent from the current population (Appendix H.2).

The Navigator receives a two-resolution view rather than the full population: complete code for a few high-ranking niche incumbents and summaries of descriptions and scores from five additional niches. We use three anchors for SYNTHESIS, the champion alone for SURGICAL, and two anchors for REFRAME. This view bounds context while exposing behaviorally distinct alternatives without encouraging verbatim imitation (see Appendix D.3 for details). The Navigator outputs a complete candidate program and its strategy description, and the program is evaluated immediately. If it is valid, the Speculators generate $J{=}5$ nearby variants, allowing a single expensive intervention to seed several low-cost follow-up evaluations.

Advisor: consolidating evidence into guidance. Every $\Delta_A{=}50$ evaluations, the Advisor converts recent trajectory statistics and accumulated failure information into a compact prescriptive note. Using $\mathcal{H}_t$ and the archive, it identifies *working* niches whose incumbents improved during the latest interval and uses *saturated* to denote niches with high evaluation activity but no incumbent improvement over the same interval. It also draws on a persistent memory of recurring execution-failure signatures with representative evaluator messages (Appendix D.4). Based on these signals and its previous note, the Advisor rewrites guidance of at most 500 words specifying which mechanisms to preserve, which directions to de-emphasize, and which recurring errors to avoid. The note thus acts as a rolling, revisable memory rather than an ever-growing transcript. To reduce feedback-induced herding, the note is independently included in each Speculator prompt with probability $\rho_A{=}0.35$, leaving most proposals to explore without explicit Advisor guidance.

4 EXPERIMENTS

We address three research questions: **RQ1:** Does SPECEVO outperform strong LLM-driven evolutionary methods under matched wall-clock, API-cost, and evaluation budgets? **RQ2:** Does *speculate-then-consolidate* outperform cost-efficient strategies exploiting the asymmetry between lightweight and frontier models? **RQ3:** How do the Speculator, Navigator, and Advisor individually contribute to exploration diversity, strategic search progress, and failure reduction?

Benchmarks. We evaluate SPECEVO on 176 real-world scientific and algorithmic discovery tasks spanning four categories: mathematics (7 problems from AlphaEvolve Math (Novikov et al., 2025)), systems optimization (4 problems from ADRS (Cheng et al., 2025)), combinatorial optimization (36

Table 1: Performance under a budget of 500 model calls. SPECEVO achieves the best performance on five of six tasks, while using the frontier model for around 2.4% of calls.

Strategy	Circle Packing		Circle Rect.		Heil. Conv.		Heil. Tri.		MinMax 14-3		Signal		Frontier Model Calls (%)
	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	
RelayEvolve	2.521 ± 0.018	2.531	2.238 ± 0.020	2.249	0.022 ± 0.000	0.022	0.029 ± 0.001	0.030	0.221 ± 0.005	0.224	0.677 ± 0.012	0.684	89.0
All-cheap	2.265 ± 0.164	2.360	2.198 ± 0.018	2.219	0.020 ± 0.002	0.022	0.015 ± 0.007	0.019	0.218 ± 0.006	0.222	0.673 ± 0.005	0.678	0.0
Fixed-switch	2.500 ± 0.000	2.500	2.264 ± 0.142	2.346	0.021 ± 0.001	0.021	0.028 ± 0.004	0.032	0.229 ± 0.016	0.238	**0.725** ± 0.075	**0.769**	50.0
Random	2.500 ± 0.000	2.500	1.762 ± 0.505	2.345	0.024 ± 0.001	0.024	0.023 ± 0.004	0.027	0.224 ± 0.000	0.224	0.659 ± 0.060	0.702	49.9
Bandit	2.471 ± 0.025	2.500	2.201 ± 0.087	2.251	0.024 ± 0.002	0.026	0.027 ± 0.002	0.029	0.233 ± 0.004	0.236	0.658 ± 0.032	0.695	50.2
LEVI	2.544 ± 0.032	2.607	2.174 ± 0.109	2.352	0.023 ± 0.001	0.025	0.031 ± 0.002	0.033	0.231 ± 0.005	0.238	0.562 ± 0.041	0.619	3.2
SPECEVO (Ours)	**2.602** ± 0.034	**2.635**	**2.342** ± 0.011	**2.362**	**0.026** ± 0.001	**0.027**	**0.033** ± 0.000	**0.034**	**0.239** ± 0.001	**0.240**	0.687 ± 0.015	0.725	**2.4**

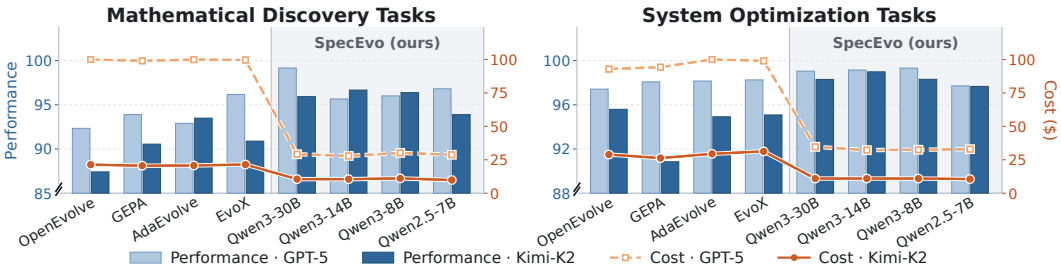

Figure 3: Performance under a fixed three-hour wall-clock budget. SPECEVO attains the highest performance in both families while reducing API cost by roughly 50%–80%, remaining stable across model configurations.

problems from CO-Bench (Sun et al., 2026)), and scientific equation discovery (129 problems from LSR-Synth (Shojaee et al., 2025b)). Detailed benchmark descriptions are provided in Appendix F.

Baselines and setups. Our baselines comprise single-LLM configurations of OpenEvolve (Sharma, 2025), GEPA (Agrawal et al., 2025), AdaEvolve (Cemri et al., 2026), and EvoX (Liu et al., 2026b), alongside the cost-efficient heterogeneous-model approaches LEVI (Tanveer, 2026) and RelayEvolve (Luo et al., 2026a). Following RelayEvolve's evaluation design, we further examine model coordination through four controlled allocation strategies: All-cheap, Fixed-switch, Random, and Bandit. Appendix C provides their definitions and implementation details. We evaluate frontier proprietary and open-weight models: GPT-5 (OpenAI, 2025) and Kimi-K2-Thinking (hereafter Kimi-K2; Team et al., 2025), together with four Qwen models ranging from 7B to 30B parameters (Qwen Team, 2025), all accessed through OpenRouter (see Table 9 for prices). Unless otherwise stated, SPECEVO uses GPT-5 for the Navigator and Qwen3-30B-A3B-Instruct-2507 for the Speculators and Advisor. Experiments run on a dual-socket server with two 16-core Intel Xeon Silver 4314 CPUs, with results averaged over three independent runs. To ensure a fair comparison under the three-hour wall-clock budget, we configure baselines with native support for parallel search to use $W = 4$ workers, while retaining their original sequential execution schedules otherwise. Section 4.2 reports component ablations and hyperparameter sensitivity analyses under the same three-hour wall-clock budget. Full hyperparameters and metric definitions are provided in Appendices D and E, respectively.

4.1 MAIN RESULTS

RQ1: Overall effectiveness. SPECEVO remains consistently strong across domains, budgets, and model configurations, supporting the contribution of its search organization across the tested backbones. Figures 3 and 4 summarize performance under the three-hour wall-clock budget. We find that SPECEVO leads most mathematical settings and all systems settings, while achieving the highest aggregate CO-Bench score, 3.8% higher than the strongest competitor, and ranking first on over 50% of the problems. Across these domains, its API cost is approximately 50%–80% lower than that of frontier-heavy baselines under the API pricing used in our experiments. Figure 1 further shows that SPECEVO is **robust across the tested frontier and lightweight backbones**, whereas competing methods exhibit performance drops of up to 20% with Qwen3-30B. With GPT-5 as the frontier model, all small-model variants attain the best score of 0.240 on MinMax 14-3 and the maximum score of 26.256 on PRISM. Moreover, while several baselines degrade noticeably when the frontier

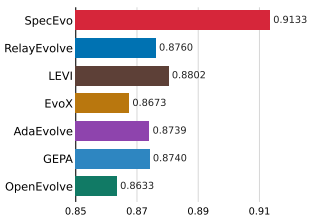

Figure 4: Performance comparison on CO-Bench.

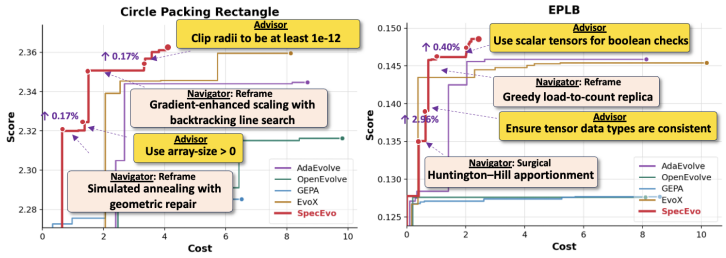

Figure 5: Cost–quality trade-off with GPT-5. Navigator and Advisor accelerate search and help escape local optima.

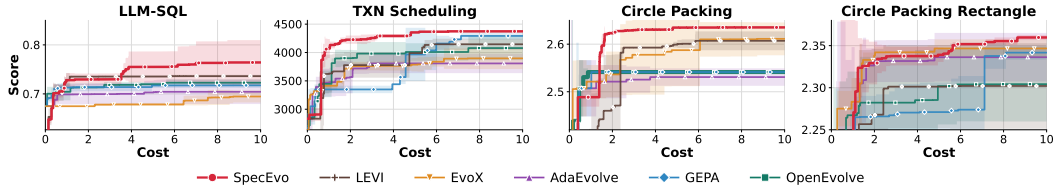

Figure 6: Performance comparison under $10 API cost.

backbone is changed from GPT-5 to Kimi-K2, SPECEVO remains competitive across configurations (Appendix G.2). See Appendices G.1–G.3 for detailed results and cost breakdowns. Figure 5 further shows that SPECEVO reaches stronger solutions earlier and at substantially lower API cost, whereas other baselines improve more slowly and plateau at lower quality despite greater expenditure.

Under the $10 API budget, Figure 6 further shows that SPECEVO outperforms the strongest baseline by shifting routine exploration to inexpensive Speculators and reserving sparse, history-informed frontier calls for strategic intervention, with the largest improvement observed on LLM-SQL (5.12% gain). Importantly, Table 2 and Figure 1 (bottom) show that the advantage persists under an identical budget of 500 evaluations on LSR-Synth. In a separate sample-efficiency analysis, we continue the baseline trajectories beyond 500 evaluations and measure when they first match SPECEVO's performance at evaluation 500. SPECEVO achieves the lowest NMSE across all eight domain–split settings, while competing methods require approximately 43%–65% more evaluations to reach comparable quality. The consistent advantage on OOD data and high symbolic accuracy further suggest that SPECEVO discovers more generalizable functional structure rather than merely fitting observed samples. Further breakdowns and statistical significance analyses are provided in Appendix G.4.

RQ2: Heterogeneous model coordination. SPECEVO's advantage stems not merely from combining inexpensive and frontier models, but from coordinating their roles using accumulated search evidence. Table 1 compares coordination strategies under a common budget of 500 model calls. We find that SPECEVO ranks first on most tasks and never below second, while using the frontier model for only 2.4% of calls, compared with 89.0% for RelayEvolve. All-cheap omits frontier reasoning, while Fixed-switch, Random, and Bandit vary model use without the same trajectory-conditioned coordination. RelayEvolve ends lightweight exploration after a one-way handoff, whereas LEVI assigns the frontier model primarily to predefined paradigm shifts. SPECEVO instead combines fixed Navigator checkpoints with stagnation-dependent intervention modes, inexpensive follow-up variants, and Advisor guidance distilled from search history. Since LEVI also uses a small frontier-call share, these results suggest that effective coordination depends on how scarce frontier reasoning informs subsequent search, beyond simply reducing its frequency.

RQ3: Search dynamics. The search traces on mathematical discovery tasks reveal how the three roles form a complementary feedback loop. Figure 8 shows that SPECEVO sustains higher code diversity than both its single-prompt ablation and LEVI as search progresses. Since CDI depends on archive cardinality, we compare methods at checkpoints with comparable archive sizes to avoid confounding diversity with population size. Speculators combine stagnation-adaptive parent sampling with specialized variation prompts to counter lightweight models' tendency to repeat familiar structures (Dat et al., 2025). The archive retains distinct strategy families, including analytic template-

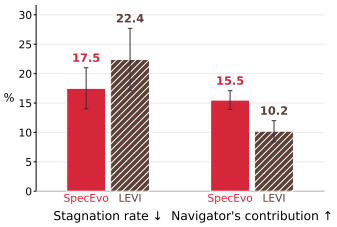

Figure 7: Stagnation rates and frontier-model contributions.

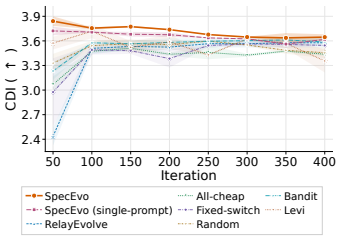

Figure 8: Diversity analysis of population.

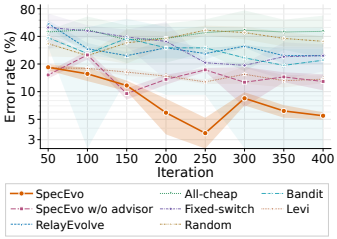

Figure 9: Execution error rates with and without the Advisor.

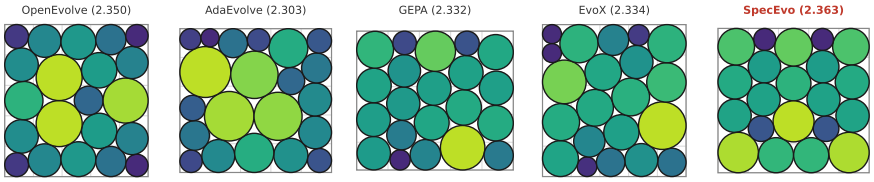

Figure 10: Best solutions generated by different methods for the Circle Packing Rectangle task.

based constructions, penalty-annealed search, and neighbor-query-accelerated local refinement (examples in Appendix I). Figure 7 shows a lower stagnation rate (the longest global-best plateau's share of the total evaluation budget) and a greater frontier-model contribution (the fraction of new bests originating from frontier-model proposals or their lightweight descendants) than LEVI. At fixed checkpoints, the Navigator uses stagnation to select among synthesis, surgical refinement, and reframing, adjusting the scope of intervention as progress slows. Figure 5 provides concrete examples of these interventions redirecting stalled searches toward better solutions. The Advisor closes the feedback loop at the execution level by converting recurring failures, such as invalid indexing, undefined names, dtype mismatches, or constraint violations, into reusable guidance. This reduces execution errors to approximately 5%–8%, compared with 13%–25% for LEVI (Figure 9), while stochastic injection leaves most prompts unguided to preserve independent exploration.

4.2 Ablation Studies

We ablate the speculation and consolidation components of SpecEvo's StC workflow (Table 3). For speculation, we remove AST features or description embeddings from the archive descriptor, disable periodic refits, or replace the six specialized prompts with a single general mutation operator. For consolidation, we remove the Navigator or replace its adaptive mode selection with random selection or a fixed mode (Synthesis, Surgical, or Reframe). We also remove the Advisor or replace its evolving guidance with fixed advice. We further assess sensitivity to the archive capacity K, Navigator and Advisor intervals, and Speculator count (Figures 11, 12).

Component contributions. Table 3 supports two complementary sources of improvement: diverse exploration and consolidation that adapts to the search trajectory. Structural and semantic descriptors preserve distinct implementations and strategies, while periodic refits keep niches aligned with the evolving population. Specialized Speculator prompts sustain complementary search moves; collapsing them into a single operator produces the largest exploration-side degradation, particularly on Heilbronn Triangle. The consolidation ablations highlight the importance of conditioning feedback on search state. Random Navigator modes underperform on both tasks, while the strongest fixed mode differs across tasks, supporting the value of matching intervention scope to stagnation. Fixed guidance likewise underperforms adaptive Advisor updates and even yields lower mean performance than removing the Advisor on Heilbronn Triangle. This suggests that static advice can become counterproductive as the search evolves, motivating continual revision based on productive mechanisms, saturated directions, and recurring failures.

Case studies. Figure 5 illustrates this division of labor, while Figure 10 shows the final packing solutions. On Circle Packing Rectangle, routine refinement initially concentrates on similar geo-

Table 2: Performance on LSR-Synth under 500 evaluations. The final row reports the total evaluations required for each baseline to match SPECEVO@500.

Domain	Split	OE	GEPA	AdaEvo	EvoX	RelayEvo	LEVI	SPECEVO
Chemistry	ID	5.27E-06	1.72E-05	2.48E-06	6.87E-06	2.59E-06	5.80E-06	**4.57E-07**
	OOD	4.99E-02	2.69E-01	6.05E-02	8.68E-02	1.30E-01	1.17E-01	**1.68E-02**
Biology	ID	6.33E-05	8.79E-05	9.92E-06	3.79E-05	1.01E-05	3.87E-06	**3.34E-07**
	OOD	7.48E-01	1.92E+00	1.28E-01	5.83E-01	1.56E-01	1.96E-02	**3.74E-03**
Materials	ID	5.92E-08	1.86E-07	6.32E-07	9.05E-08	5.79E-08	5.73E-07	**4.84E-08**
	OOD	2.44E-06	7.57E-05	1.18E-04	1.63E-05	6.72E-06	8.36E-06	**1.68E-06**
Physics	ID	3.29E-04	1.53E-03	2.79E-04	6.66E-04	1.81E-04	1.38E-05	**2.92E-06**
	OOD	6.93E-03	2.82E-02	1.32E-02	5.32E-02	3.30E-03	2.23E-03	**4.75E-04**
Evals. to Beat SPECEVO		764	824	715	780	768	752	–

Table 3: Ablation study on representative tasks. Relative decreases in mean score from the full SPECEVO configuration.

Ablation setting	Circle Packing Rectangle	Heilbronn Triangle
w/o AST features	↓0.38%	↓7.76%
w/o description embedding	↓0.13%	↓6.37%
w/o refit	↓1.65%	↓4.99%
Single operator prompt	↓2.17%	↓12.74%
w/o Navigator	↓2.25%	↓9.42%
Random mode	↓2.60%	↓7.47%
Synthesis mode only	↓2.71%	↓0.83%
Surgical mode only	↓0.49%	↓12.47%
Reframe mode only	↓1.97%	↓7.76%
w/o Advisor	↓3.82%	↓6.09%
Fixed guidance	↓3.46%	↓8.31%

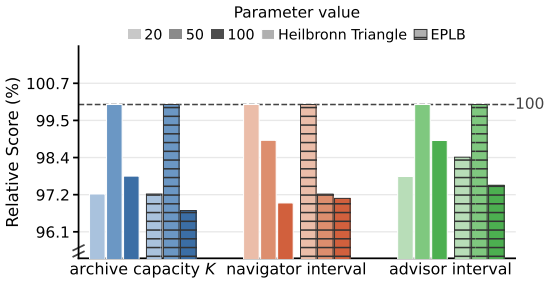

Figure 11: Sensitivity to archive capacity K and Navigator/Advisor intervals.

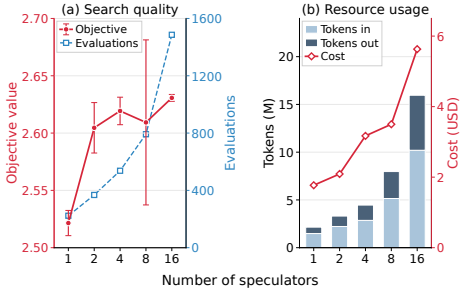

Figure 12: Effect of the number of Speculators on search quality and resource usage.

metric constructions. The Navigator then uses REFRAME to introduce simulated annealing with geometric repair and, later, gradient-enhanced scaling with backtracking line search. The Advisor complements these shifts with safeguards for empty arrays and near-zero radii, allowing subsequent Speculator variants to explore the new neighborhoods without repeatedly failing on numerical edge cases. A similar pattern appears on EPLB: SURGICAL first strengthens the incumbent through Huntington–Hill apportionment and load-aware heap scheduling, while REFRAME later replaces the saturated placement logic with load-to-count allocation and rotated round-robin assignment. Between these interventions, the Advisor helps prevent recurring dtype and scalar-boolean errors. These interventions provide reusable starting points for continued low-cost Speculator refinement.

Hyperparameter sensitivity. In Figure 11, we find that $K = 50$ yields the strongest performance, suggesting a balance between strategy coverage and sufficient refinement within each niche. The preferred Navigator and Advisor intervals are 20 and 50 evaluations, respectively. This difference is consistent with their informational needs: strategic intervention benefits from timely responses to stagnation, whereas reusable guidance requires enough observations to distinguish recurring patterns from isolated outcomes. Figure 12 shows that scaling from one to four Speculators improves both solution quality and evaluation throughput on Circle Packing. Beyond four, throughput continues to increase, but quality gains become marginal while token usage and API cost rise sharply. We therefore use four Speculators to capture most of the benefit of parallel exploration at moderate cost.

5 CONCLUSION

We introduced SPECEVO, a multi-LLM framework that coordinates low-cost exploration and sparse frontier-model reasoning through continual consolidation of search evidence. Its *speculate-then-consolidate* workflow combines parallel lightweight Speculators, persistent Advisor guidance distilled from successes, failures, and saturated directions, and stagnation-dependent Navigator interventions at fixed checkpoints. Across 176 scientific and algorithmic discovery tasks, SPECEVO matches or outperforms strong baselines under matched wall-clock, API-cost, and evaluation budgets, with 50%–80% lower API expenditure under the three-hour wall-clock budget. Its gains across backbone configurations and under matched evaluation budgets support the value of using accumulated search evidence to guide sparse frontier interventions and subsequent lightweight refinement.

AI USE STATEMENT

LLMs in the method and evaluation. LLMs are a core component of the method studied in this work. SPECEVO and all baselines call the language models listed in Table 9 to generate, refine, and analyze candidate programs (Section 4 and Appendix C), and we use `text-embedding-3-small` for SPECEVO's archive descriptors and for the diversity metric (Appendices D and E). To score symbolic accuracy on LSR-Synth, we follow the LLM-SRBench protocol (Shojaee et al., 2025b) and use GPT-4o to judge whether a discovered equation is mathematically equivalent to the ground truth (Appendix G.4). These uses are part of the experimental method and evaluation protocol described in the paper.

AI assistance in research and writing. Beyond the experimental uses above, we did not use generative AI tools for any task that requires disclosure under the ICLR 2027 AI policy. We used generative AI tools to assist with writing, including language editing, improving clarity and readability, and condensing or rephrasing parts of the manuscript. The authors reviewed and revised all AI-assisted text. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034, 2024.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025.

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

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023a.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023b.

Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, et al. Barbarians at the gate: How ai is upending systems research, 2025. *URL https://arxiv.org/abs/2510.06189*, 5(4), 2025.

Pham Vu Tuan Dat, Long Doan, and Huynh Thi Thanh Binh. Hsevo: Elevating automatic heuristic design with diversity-driven harmony search and genetic algorithm using llms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26931–26938, 2025.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks Lakshmanan, and Ahmed H Awadallah. Hybrid llm: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, volume 2024, pp. 41348–41366, 2024.

Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. *Advances in Neural Information Processing Systems*, 37:44678–44709, 2024.

Ha Minh Hieu, Hung Phan, Tung Duy Doan, Tung Dao, Cong Dao Tran, and Huynh Thi Thanh Binh. Pareto-grid-guided large language models for fast and high-quality heuristics design in multi-objective combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 36964–36972, 2026.

Yiqian Huang, Shiqi Zhang, Tianyuan Jin, and Xiaokui Xiao. Scope: Cost-efficient model selection for compound ai systems under quality constraints. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, pp. 1922–1931, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3818067. URL https://doi.org/10.1145/3770855.3818067.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

Robert Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *International Conference on Learning Representations*, volume 2026, pp. 74026–74078, 2026.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366. Springer, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Yuchen Li, Handing Wang, and Yaochu Jin. Evosr-llm: Evolutionary symbolic regression guided by large language models. *IEEE Transactions on Evolutionary Computation*, 2026.

Fang Liu, Yang Liu, Lin Shi, Houkun Huang, Ruifeng Wang, Zhen Yang, Li Zhang, Zhongqi Li, and Yuchi Ma. Exploring and evaluating hallucinations in llm-powered code generation. *arXiv e-prints*, pp. arXiv–2404, 2024a.

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. *arXiv preprint arXiv:2401.02051*, 2024b.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, et al. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 58(8):1–32, 2026a.

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

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J Chan, Jakob Foerster, Mihaela van der Schaar, and Robert T Lange. Discovering preference optimization algorithms with and for large language models. *Advances in Neural Information Processing Systems*, 37:86528–86573, 2024.

Sichun Luo, Yi Huang, Guanzhi Deng, Haibo Wang, Haochen Luo, Lei Li, Zefa Hu, Junlan Feng, and Qi Liu. Relay, don't route: Adaptive population handoff for cost-efficient llm-driven evolution. *arXiv preprint arXiv:2608.05651*, 2026a.

Sichun Luo, Yi Huang, Haochen Luo, Fengyuan Liu, Guanzhi Deng, Lei Li, Qinghua Yao, Zefa Hu, Junlan Feng, and Qi Liu. Seaevo: Advancing algorithm discovery with strategy space evolution. *arXiv preprint arXiv:2604.24372*, 2026b.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. *arXiv preprint arXiv:2405.09783*, 2024a.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Jim Fan, et al. Eureka: Human-level reward design via coding large language models. In *International conference on learning Representations*, volume 2024, pp. 26516–26560, 2024b.

Yoshitomo Matsubara, Naoya Chiba, Ryo Igarashi, and Yoshitaka Ushiku. Rethinking symbolic regression datasets and benchmarks for scientific discovery. *arXiv preprint arXiv:2206.10540*, 2022.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

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

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

OpenAI. GPT-5 System Card. `https://openai.com/index/gpt-5-system-card/`, August 2025.

Steven T Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130, 2014.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Qwen Team. Qwen3: Think deeper, act faster. `https://qwenlm.github.io/blog/qwen3/`, April 2025.

Pretam Ray, Pratik Prabhanjan Brahma, Zicheng Liu, and Emad Barsoum. Adaptevolve: Improving efficiency of evolutionary ai agents through adaptive model selection. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 40625–40633, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan Reddy. Llm-sr: Scientific equation discovery via programming with large language models. In *International Conference on Learning Representations*, volume 2025, pp. 16054–16085, 2025a.

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K Reddy. Llm-srbench: A new benchmark for scientific equation discovery with large language models. *arXiv preprint arXiv:2504.10415*, 2025b.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. Co-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33126–33134, 2026.

Temoor Tanveer. Levi: Stronger search architectures can substitute for larger llms in evolutionary search. *arXiv preprint arXiv:2605.09764*, 2026.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Niki Van Stein and Thomas Bäck. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29 (2):331–345, 2024.

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. *IEEE Transactions on Evolutionary Computation*, 29(2):534–554, 2024.

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

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. *Advances in neural information processing systems*, 37:43571–43608, 2024.

Yansen Zhang, Yilu Liu, Tianyu Liu, Jiamin Chen, Xiaokun Zhang, Kai Xie, Xue Liu, Chen Ma, and Yiyan Qi. Budget-aware llm discovery via cost-calibrated frontier utility. *arXiv preprint arXiv:2607.26828*, 2026.

Appendix

This appendix provides supplementary material supporting the methodology, evaluation, and analysis presented in the main paper. Appendix A discusses the limitations of SPECEVO. Appendices B and C extend the related-work discussion and document the baselines and common evaluation protocol. Appendix D provides the complete algorithmic specification of SPECEVO, including the Speculator loop and the Navigator and Advisor interventions, while Appendix E defines the metrics used to analyze search dynamics. Appendices F and G describe the benchmark suites and provide detailed experimental results, including API cost and token usage breakdowns. Appendix H presents the complete prompt templates for all three roles, and Appendix I analyzes generated programs, recurring failure modes, algorithmic mechanisms, and the guidance consolidated by the Advisor on Circle Packing Rectangle and EPLB.

CONTENTS

A Limitations **15**

B Extended Related Work **15**

C Baseline Details **16**

D Algorithm Details **18**

D.1 Behavioral Descriptor Construction 18

D.2 The complete SPECEVO workflow 19

D.3 Navigator intervention 20

D.4 Advisor intervention 21

E Metric Details **22**

E.1 Aggregate Metrics Used in Figure 3 22

E.2 Code Diversity Index (CDI) 23

E.3 Stagnation rate 23

E.4 New bests traced to the frontier model 24

F Benchmark Details **24**

F.1 Mathematical Optimization Benchmarks 24

F.2 Systems Optimization Benchmarks 24

F.3 CO-Bench 24

F.4 LSR-Synth 25

G Additional Experimental Results **27**

G.1 Breakdown of API Cost and Token Usage 27

G.2 Detailed Results for Mathematical Discovery and Systems Optimization Tasks . 27

G.3 Detailed Results for Each CO-Bench Problem Category 29

G.4 Detailed Results on LSR-Synth 31

H Prompt Templates **36**

H.1 Speculator 36

H.2 Navigator 37

H.3 Advisor 39

I Qualitative Analysis **39**

I.1 Circle Packing Rectangle 40

I.2 EPLB 42

A LIMITATIONS

Dependence on parallel computing infrastructure. The wall-clock efficiency of SPECEVO partly depends on the amount of parallelism available to the Speculator pool. Our experiments use $W = 4$ concurrent Speculators on a dual-socket server with two 16-core Intel Xeon Silver 4314 processors. Consequently, the number of candidates explored within a fixed wall-clock budget may change with the worker count, CPU capacity, memory bandwidth, network latency, and API rate limits. Although our fixed API-cost and fixed evaluation-budget experiments provide complementary controls, they do not make the wall-clock results hardware-independent. The reported cost reductions should therefore be interpreted primarily as savings in LLM API expenditure under the specified computing environment, rather than reductions in total computational cost.

Compute-intensive candidate evaluation. The benchmarks considered in this work use relatively lightweight evaluators: candidate programs can generally be executed and scored on CPUs without repeatedly training large neural networks, running long simulations, or conducting physical experiments. This setting allows multiple Speculators to evaluate proposals concurrently at modest local-compute cost. For tasks whose evaluation requires GPU training, large-scale inference, expensive simulation, or real-world experimentation, increasing the number of parallel workers may instead multiply accelerator usage, memory requirements, and the cost of unsuccessful trials. Moreover, a fixed number of evaluations does not represent a fixed compute budget when candidate evaluations have heterogeneous costs. Evaluating SPECEVO in such settings is an important direction for future work. A more comprehensive study should jointly control API expenditure, CPU- or GPU-hours, peak memory, energy consumption, wall-clock time, and the number of evaluated candidates. It would also be valuable to adapt the worker count and the frequencies of Navigator and Advisor intervention dynamically according to evaluator cost, available hardware, and observed search progress.

Reliability of model-generated descriptions. The semantic component of our archive descriptor relies on descriptions produced by the proposing model, predominantly lightweight Speculators. These descriptions may omit important changes, remain generic across distinct programs, or claim mechanisms absent from the code. Such inaccuracies can distort semantic distances, grouping different strategies together or separating similar implementations, thereby affecting niche assignment, parent selection, and the population summaries provided to the Navigator and Advisor. We have not systematically quantified sensitivity to description quality. Future work could evaluate this sensitivity through controlled description perturbations and explore code-grounded verification or descriptors derived from execution behavior.

B EXTENDED RELATED WORK

LLM-Guided Evolutionary Search. LLM-guided evolutionary search combines language models as semantic variation operators with executable evaluators and population-based optimization. FunSearch established this paradigm by evolving programs for open mathematical problems (Romera-Paredes et al., 2024), while ELM combined pretrained code models with quality-diversity search (Lehman et al., 2023). Subsequent methods specialize the evolutionary loop in different ways: EoH jointly evolves natural-language ideas and executable heuristics (Liu et al., 2024b); ReEvo derives reflective guidance from performance comparisons (Ye et al., 2024); LLaMEA evolves optimization algorithms through iterative evaluation (Van Stein & Bäck, 2024); HSEvo explicitly balances diversity and convergence (Dat et al., 2025); and MPaGE extends heuristic evolution to multiple objectives (Hieu et al., 2026). Related systems apply LLM-guided search to reward design (Ma et al., 2024b), preference optimization (Lu et al., 2024), and scientific equation discovery (Li et al., 2026; Shojaee et al., 2025b;a; Grayeli et al., 2024; Ma et al., 2024a). More general evolutionary coding agents include AlphaEvolve (Novikov et al., 2025), OpenEvolve (Sharma, 2025), and ShinkaEvolve (Lange et al., 2026), while GEPA evolves prompts and natural-language rules through reflection and Pareto selection (Agrawal et al., 2025). Recent work further improves the representation and control of search: SeaEvo organizes candidates into persistent semantic strategy families (Luo et al., 2026b); EvoX evolves the search strategy itself (Liu et al., 2026b); AdaEvolve adapts exploration intensity, resource allocation across islands, and meta-level guidance (Cemri et al., 2026); and CORAL delegates search decisions to autonomous peer agents communicating

through shared memory (Qu et al., 2026). Unlike these systems, SPECEVO assigns routine exploration, experience consolidation, and strategic intervention to models of different cost, analogous to the division of labor in a research laboratory.

Cost-Aware LLM Routing and Evolutionary Search Allocation. Cost-aware LLM systems reduce inference expense by selecting among models with different capabilities. FrugalGPT constructs cascades that progressively escalate difficult queries (Chen et al., 2023b), AutoMix uses self-verification to decide whether a small-model response requires augmentation by a stronger model (Aggarwal et al., 2024), and RouteLLM learns routing policies from human preference data (Ong et al., 2025). These methods focus on independent inference requests, whereas evolutionary search must allocate resources across a sequence of interdependent population updates. ShinkaEvolve addresses this setting with a bandit that selects mutation models according to their observed performance (Lange et al., 2026). AdaptEvolve instead runs a small model first and uses token-level confidence statistics with an online decision-tree router to determine whether the mutation should be regenerated by a larger model (Ray et al., 2026). AdaEvolve allocates computation across concurrent evolutionary processes according to their recent progress and introduces meta-level reasoning to adjust search behavior under stagnation (Cemri et al., 2026). RelayEvolve shifts allocation from individual mutations to entire search phases: a cheap model develops multiple trajectories in short blocks, Relay Gain measures the marginal quality and coverage of their candidate bank, and saturation of this signal triggers a one-time handoff of a curated seed population to strong-model evolution (Luo et al., 2026a). LEVI combines three complementary mechanisms: bootstrapped CVT-MAP-Elites for diversity preservation, role-aware routing that assigns local refinements to smaller models and paradigm shifts to a stronger model, and rank-preserving proxy benchmarks that reduce evaluation overhead (Tanveer, 2026).

These approaches optimize different components of resource allocation but retain important structural restrictions. ShinkaEvolve and AdaptEvolve select among models at the mutation level, leaving them functionally interchangeable; AdaptEvolve additionally requires token-level probabilities that are unavailable from many proprietary APIs. AdaEvolve adapts the amount and direction of search effort but does not organize heterogeneous models into persistent functional roles. RelayEvolve captures population-level state more explicitly, but information flows through a single cheap-to-strong transition: once handoff occurs, lightweight exploration ends, and the strong phase inherits selected candidates but no consolidated knowledge about discarded attempts, recurring failures, or exhausted trajectories. LEVI provides a clearer division between local and structural mutations, but its stronger model is assigned primarily to a predefined paradigm-shift operation, without a dedicated component that continually aggregates trajectory-wide evidence for subsequent small-model search. SPECEVO instead treats model heterogeneity as a persistent division of labor throughout evolution. Its Speculators perform high-throughput variation, its Advisor repeatedly converts successes, failures, and niche-level progress into reusable guidance, and its Navigator examines the population at sparse checkpoints to choose among synthesis, surgical refinement, and reframing. Where prior methods optimize model selection, phase transition, or mutation type, SPECEVO coordinates candidate generation, experience consolidation, and strategic intervention within the same evolutionary process.

C BASELINE DETAILS

Common evaluation protocol. We evaluate all baselines using their native search logic and publicly released implementations. We introduce only the task adapters required to expose the same problem specification, editable program scaffold, execution environment, and evaluator to every method. All three-hour wall-clock experiments use the same hardware and evaluation environment. For baseline implementations supporting native parallel candidate search, we configure four concurrent workers, matching SPECEVO. The wall-clock results therefore characterize end-to-end performance under this protocol, complemented by comparisons under fixed API-cost and evaluation budgets.

All initialization, mutation, repair, reflection, meta-guidance, and paradigm-shift calls are included in API-cost and wall-clock accounting. API expenditure is computed from the actual input and output tokens of each call using the prices in Table 9. Under an evaluation budget, every candidate submitted to the evaluator consumes one evaluation, including programs that fail to parse, violate

constraints, or time out; auxiliary calls that do not produce a candidate do not increment this counter but still contribute to API cost and elapsed time. Under the 500-call RQ2 budget, every LLM invocation counts toward the common limit. Each method returns the highest-scoring valid program discovered before its stopping condition is reached.

OpenEvolve. OpenEvolve (Sharma, 2025) implements asynchronous island-based program evolution with a MAP-Elites-style database. Candidate parents are selected from a mixture of high-performing and behaviorally diverse programs, mutated using evaluator feedback, and periodically exchanged across islands. We retain its native archive, migration, and prompt construction mechanisms, while assigning the selected frontier backbone to all LLM calls.

GEPA. We use the inference-time code-optimization configuration of GEPA (Agrawal et al., 2025), treating the candidate program as the textual artifact being optimized. GEPA converts execution traces and evaluator diagnostics into natural-language reflection, uses the resulting analysis to propose targeted mutations, and maintains complementary candidates through Pareto-aware selection and merging. The same frontier backbone is used for both reflection and candidate generation.

AdaEvolve. AdaEvolve (Cemri et al., 2026) derives an accumulated-improvement signal from the search trajectory and uses it at three levels: adjusting exploration intensity within each island, allocating evaluations across islands through a bandit scheduler, and generating new high-level tactics when global progress stalls. We use the authors' released implementation and default adaptive schedules, with the designated frontier backbone serving both mutation and meta-guidance calls.

EvoX. EvoX (Liu et al., 2026b) jointly evolves candidate programs and the strategy used to generate them. Its inner loop evaluates programs under the current search strategy, while its outer loop periodically revises parent selection and variation behavior according to the progress achieved during the preceding search window. We preserve this two-level evolution and use the same frontier backbone for solution generation and search-strategy evolution.

LEVI. LEVI (Tanveer, 2026) first generates diverse seeds to calibrate a CVT-MAP-Elites archive and then applies role-aware model routing. Qwen3-30B performs ordinary local refinements, whereas the frontier model is reserved for periodically triggered paradigm shifts conditioned on representatives from distinct archive regions. We retain its native archive descriptors, trigger policy, lightweight follow-up variants, and proxy-benchmark construction where applicable. All reported final programs are re-evaluated using the complete task evaluator.

RelayEvolve. RelayEvolve (Luo et al., 2026a) begins with multiple lightweight-model trajectories whose search blocks are allocated by a Grow–Deepen bandit. Relay Gain measures the marginal quality and coverage contributed to an online candidate bank and triggers handoff when further cheap-model exploration saturates. The terminal candidate pool is then curated using the method's quality–diversity objective, and the selected population initializes a shared frontier-model search using the remaining budget. We retain the default strong-model budget reserve and all handoff hyperparameters. Once handoff occurs, lightweight-model exploration does not resume.

Controlled model-allocation strategies. To isolate the effect of model coordination from the mere availability of two differently priced models, we additionally implement four controls on a shared asynchronous evolutionary backend. They use identical parent selection, population updates, mutation prompts, evaluator feedback, and context limits, differing only in how the lightweight and frontier models are assigned. For a total call budget B, these strategies are defined as follows:

- **All-cheap** assigns every call to the lightweight model.
- **Fixed-switch** uses the lightweight model for the first $B/2$ calls and then performs a one-time switch to the frontier model for the remaining calls, carrying the current population forward without additional curation.
- **Random** independently selects the lightweight or frontier model with probability 0.5 for every call.

- **Bandit** treats the two models as arms of a UCB1 scheduler. After initializing both arms, model selection balances their observed mean rewards with an exploration bonus. The reward is the non-negative improvement in the global best score produced by the selected model; invalid or non-improving proposals receive zero reward.

D ALGORITHM DETAILS

We set the temperatures to 0.5, 0.5, and 0.8 for the *synthesis*, *surgical*, and *reframe* modes, respectively, and to 0.7 and 0.4 for the Speculator and Advisor. We set $H_g = 100$, $H_l = 20$, $\beta_{\min} = 0.3$, $\beta_{\max} = 2.0$, $\gamma_1 = 0.4$, and $\gamma_2 = 0.7$. We refit the archive descriptor space every $\Delta_C = 50$ completed evaluations, setting $\Delta_C = \Delta_A$ for consistency with the Advisor update interval. After each Navigator call generates a set of hypotheses, the Speculators produce $J = 5$ variants for each hypothesis.

D.1 BEHAVIORAL DESCRIPTOR CONSTRUCTION

We provide the implementation details of the structural–semantic descriptor introduced in Eq. 1. For a program p, we first parse its source code into a Python abstract syntax tree and extract the following eight statistics, in fixed order:

$$a(p) = [\, a_{\text{depth}}, a_{\text{cyc}}, a_{\text{loop}}, a_{\text{nest}}, a_{\text{branch}}, a_{\text{func}}, a_{\text{comp}}, a_{\text{call}} \,].$$

These denote the maximum AST depth; cyclomatic complexity; the number of `For` and `While` nodes; the maximum nesting depth of such loops; and the numbers of `If`, `FunctionDef`, comprehension, and function-call nodes, respectively. Cyclomatic complexity is initialized to one, incremented for each `If`, `While`, `For`, and `ExceptHandler`, and additionally incremented by the Boolean fan-out of each `BoolOp`. To reduce the influence of unusually large programs, we apply an element-wise logarithmic transform,

$$\phi(p) = \log\bigl(1 + a(p)\bigr) \in \mathbb{R}^8.$$

As a defensive fallback, an unparsable program is assigned an all-zero structural vector, although such programs are normally rejected before archive admission.

For the semantic block, we embed the model-generated strategy description $d(p)$ using `text-embedding-3-small`. The resulting 1536-dimensional vector is L_2-normalized:

$$e(p) = \frac{f_{\text{emb}}(d(p))}{\max\{\|f_{\text{emb}}(d(p))\|_2, \varepsilon\}} \in \mathbb{R}^{1536}.$$

The raw embedding is computed once for each distinct description and cached; refits therefore do not require additional embedding calls.

At refit round r, let A_r be the current archive and let $\bar{e}_r$ be the mean of its description embeddings. We fit PCA to the centered embedding matrix and retain the leading eight principal directions, $U_r \in \mathbb{R}^{1536\times 8}$. The semantic representation is

$$q_r(p) = U_r^\top \bigl(e(p) - \bar{e}_r\bigr) \in \mathbb{R}^8.$$

The structural and semantic blocks are standardized separately, coordinate by coordinate, using their respective means and standard deviations over the current archive. A coordinate with zero empirical variance is mapped to zero. The final descriptor is the direct concatenation

$$z_r(p) = \bigl[\mathcal{Z}_r^{\text{ast}}(\phi(p)) \,\big\|\, \mathcal{Z}_r^{\text{sem}}(q_r(p))\bigr] \in \mathbb{R}^{16},$$

where $\|$ denotes concatenation. We apply no additional block weighting or normalization after concatenation; separate standardization prevents either block from dominating the Euclidean geometry used for niche assignment. The archive holds at most $K = 50$ incumbents. Rather than partitioning the population with a separate clustering step, we anchor each niche at its own incumbent: niche i is the Voronoi cell of the site $c_i = z_r(p_i) \in \mathbb{R}^{16}$, so the tessellation is determined entirely by the archive contents under the current descriptor map z_r.

Before the first refit, no descriptor map has been fitted and niche assignment does not use one: each admitted program simply opens a new niche, in admission order. This is the regime in which the

initialization seeds and their lightweight variants enter the archive, so initialization requires neither a PCA basis nor a set of sites. The first refit is triggered as soon as the archive first holds K incumbents, which, since $M \cdot V \gg K$, occurs during the lightweight diversification stage, and establishes z_1 together with the initial sites $\{c_i\}_{i=1}^{K}$.

Between refit events, z_r and the sites $\{c_i\}$ are held fixed. An incoming candidate q is projected using the most recent PCA basis and normalization statistics and assigned to $i^\star = \arg\min_i \|z_r(q) - c_i\|_2$. It is admitted if and only if $F(q) > F(p_{i^\star})$, in which case it replaces $p_{i^\star}$ but does not displace $c_{i^\star}$; the site moves only at the next refit. The archive therefore remains saturated at K occupied niches for the remainder of the run.

After initialization, every $\Delta_C = 50$ completed evaluations we refit PCA on the current archive, refresh the per-coordinate normalization statistics, rebuild the descriptors $z_r(p_i)$ of all incumbents, and re-anchor every site to its incumbent, $c_i \leftarrow z_r(p_i)$. Because the sites are the incumbents, a refit changes only the geometry used to route future candidates: it never merges two incumbents into a single niche, except in the degenerate case of two identical descriptors, where the higher-scoring program is retained, and niche index i continues to denote the same archive slot across refits. This preserves the per-niche statistics the Advisor uses to identify working and saturated niches, and it removes any need to seed the new geometry with the previous round's sites, whose coordinates are in any case no longer comparable once the PCA basis and normalization statistics have been refit.

D.2 The complete SpecEvo workflow

Algorithm 1 illustrates the overall workflow of SpecEvo. Initialization first uses a small number of frontier-model calls to obtain a diverse set of executable strategy families. Lightweight variants then expand these seeds into a sufficiently broad population for constructing the behavioral archive. After initialization, W asynchronous Speculators perform nearly all candidate generation, while the two consolidation routines are triggered on independent evaluation schedules. Importantly, Navigator interventions are non-blocking in our implementation: once a Navigator checkpoint is triggered, the intervention is dispatched as a background task, while the Speculator workers continue generating and evaluating candidates. Navigator-generated candidates and their lightweight follow-up variants are incorporated into the shared search state as they complete. Thus, the Navigator and Advisor calls in Algorithm 1 are written as single operations for notational compactness and should not be interpreted as synchronization barriers for the Speculator workers.

Speculator step. Let the archive programs be ordered by decreasing score, with r_i denoting the rank of p_i. At each worker step, the parent distribution and its stagnation-dependent exponent are:

$$\Pr(p_i \mid \mathcal{A}_t) = \frac{r_i^{-\beta_t}}{\sum_{j=1}^{|\mathcal{A}_t|} r_j^{-\beta_t}}, \qquad \beta_t = \beta_{\min} + (1 - s_t)(\beta_{\max} - \beta_{\min}). \tag{5}$$

For crossover, the second parent is drawn from a different niche whenever possible. Additional inspirations are sampled without replacement from the same rank distribution after excluding the parent(s), but only their descriptions and scores are shown. This distinction is intentional: parent code supplies a concrete object to modify, whereas description-only inspirations broaden the idea set without encouraging verbatim copying or inflating the prompt.

The six variation prompts are general mutation, focused single-weakness repair, mechanism replacement, analysis-targeted mutation, structural crossover, and component-level crossover. They are sampled uniformly, so each prompt is used with equal probability during Speculator generation.

Asynchrony and checkpoint cadence. The evaluation index t counts completed candidate evaluations rather than optimizer "generations." Because Speculators run asynchronously, multiple evaluations may complete between periodic checks, so a trigger can occur slightly after its nominal interval. After initialization, the Navigator, Advisor, and archive periodic refits maintain independent clocks t_N, t_A, and t_C, all reset to the current t. They are subsequently invoked whenever $t - t_N \geq \Delta_N$, $t - t_A \geq \Delta_A$, and $t - t_C \geq \Delta_C$, respectively. Thus, initialization does not consume any of these intervals, and each periodic operation proceeds independently according to its own evaluation cadence.

Algorithm 1 SpecEvo: asynchronous speculate-then-consolidate search

Require: Task specification $\mathcal{T}$, evaluator F, budget $\mathcal{B}$, number of Speculator workers W, Speculator $\mathcal{M}_{\mathrm{S}}$, Navigator $\mathcal{M}_{\mathrm{N}}$, Advisor $\mathcal{M}_{\mathrm{A}}$, number of initialization seeds M, number of variants per seed V, archive capacity K, Navigator interval Δ_N, Advisor interval Δ_A, periodic refit interval Δ_C, Advisor-guidance probability ρ_A
Ensure: $\arg\max_{p \in \mathcal{A}_t} F(p)$

$t \leftarrow 0$ ▷ completed evaluations
$\mathcal{A}_t, \mathcal{H}_t, \mathcal{S}_0 \leftarrow \varnothing$; $\ell \leftarrow \varnothing$
▷ Frontier-seeded initialization
for $i = 1, \ldots, M$ **do**
  $(p_i^0, d_i^0) \sim \mathcal{M}_{\mathrm{N}}\big(\cdot \mid \mathcal{T}, \{(p_j^0, d_j^0, y_j^0)\}_{j<i}\big)$
  $(y_i^0, \mathcal{A}_t, \mathcal{H}_t, t) \leftarrow \textsc{EvaluateUpdate}(p_i^0, d_i^0, F, \mathcal{A}_t, \mathcal{H}_t, t)$
  if p_i^0 is executable **then**
    $\mathcal{S}_0 \leftarrow \mathcal{S}_0 \cup \{(p_i^0, d_i^0, y_i^0)\}$
▷ Lightweight diversification
for all $(p_i^0, d_i^0, y_i^0) \in \mathcal{S}_0$ **in parallel over up to** W **workers do**
  for $j = 1, \ldots, V$ **do**
    $I_{i,j} \leftarrow \textsc{Inspirations}(\mathcal{S}_0, i)$
    $(p_{i,j}^0, d_{i,j}^0) \sim \mathcal{M}_{\mathrm{S}}(\cdot \mid \mathcal{T}, p_i^0, I_{i,j})$
    $(\cdot, \mathcal{A}_t, \mathcal{H}_t, t) \leftarrow \textsc{EvaluateUpdate}(p_{i,j}^0, d_{i,j}^0, F, \mathcal{A}_t, \mathcal{H}_t, t)$
$\mathcal{A}_t \leftarrow \textsc{RefitArchive}(\mathcal{A}_t, K)$
$t_N \leftarrow t$; $t_A \leftarrow t$; $t_C \leftarrow t$
▷ Asynchronous speculate-then-consolidate search
while no stopping condition in $\mathcal{B}$ is met **do**
  $s_t \leftarrow \textsc{Stagnation}(\mathcal{H}_t)$
  for all idle Speculator workers **do**
    $(P_t, I_t, o_t) \leftarrow \textsc{SpeculationContext}(\mathcal{A}_t, s_t)$
    $\widetilde{\ell} \leftarrow \ell$ with probability ρ_A; otherwise $\widetilde{\ell} \leftarrow \varnothing$
    Dispatch $(p, d) \sim \mathcal{M}_{\mathrm{S}}(\cdot \mid \mathcal{T}, P_t, I_t, o_t, \widetilde{\ell})$
  $(p, d) \leftarrow \textsc{AwaitNext}$
  $(\cdot, \mathcal{A}_t, \mathcal{H}_t, t) \leftarrow \textsc{EvaluateUpdate}(p, d, F, \mathcal{A}_t, \mathcal{H}_t, t)$
  if $t - t_N \geq \Delta_N$ **then**
    $(\mathcal{A}_t, \mathcal{H}_t, t) \leftarrow \textsc{Navigator}(\mathcal{T}, F, \mathcal{A}_t, \mathcal{H}_t, t)$
    $t_N \leftarrow t$
  if $t - t_A \geq \Delta_A$ **then**
    $\ell \leftarrow \textsc{Advisor}(\mathcal{T}, \mathcal{A}_t, \mathcal{H}_t, \ell, t_A, t)$
    $t_A \leftarrow t$
  if $t - t_C \geq \Delta_C$ **then**
    $\mathcal{A}_t \leftarrow \textsc{RefitArchive}(\mathcal{A}_t, K)$
    $t_C \leftarrow t$
return $\arg\max_{p \in \mathcal{A}_t} F(p)$

D.3 Navigator intervention

Algorithm 2 illustrates the overall workflow of a Navigator invocation.

Selecting anchors and inspirations. The Navigator receives a two-resolution view of the behavioral archive. We first collect the incumbent from each occupied niche and rank these niche elites by score. The highest-ranked elites are selected as *anchors*, for which the Navigator observes the complete program, strategy description, and score. The subsequent elites are used as *inspirations*, exposing only their descriptions and scores. Anchors and inspirations are disjoint within each invocation.

Full-code anchors support precise reasoning and modification, whereas description-only inspirations expose a broader set of mechanisms without overwhelming the prompt; ranking niche elites rather than the entire population keeps this context behaviorally diverse.

Table 4 summarizes the mode-specific context: SYNTHESIS needs several anchors' implementations to combine complementary mechanisms, SURGICAL needs only the champion it repairs, and

Algorithm 2 Navigator consolidation

Require: Task specification $\mathcal{T}$, evaluator F, archive $\mathcal{A}_t$, history $\mathcal{H}_t$, evaluation index t, Navigator $\mathcal{M}_{\mathrm{N}}$, Speculator $\mathcal{M}_{\mathrm{S}}$, stagnation thresholds γ_1, γ_2, number of fan-out variants J
Ensure: $\mathcal{A}_t, \mathcal{H}_t, t$

$(q_1, \ldots, q_{K_t}) \leftarrow$ RANKEDINCUMBENTS$(\mathcal{A}_t)$ ▷ one elite per occupied niche
$s_t \leftarrow$ STAGNATION$(\mathcal{H}_t)$
if $s_t \leq \gamma_1$ **then**
  $m_t \leftarrow$ SYNTHESIS; $a \leftarrow 3$
else if $s_t \leq \gamma_2$ **then**
  $m_t \leftarrow$ SURGICAL; $a \leftarrow 1$
else
  $m_t \leftarrow$ REFRAME; $a \leftarrow 2$
$k \leftarrow 5$
▷ Construct a two-resolution population view
$A_t^{(m)} \leftarrow \{(q_i, d(q_i), F(q_i)) : 1 \leq i \leq \min(a, K_t)\}$
$I_t^{(m)} \leftarrow \{(d(q_i), F(q_i)) : a < i \leq \min(a + k, K_t)\}$
$V_t^{(m)} \leftarrow$ POPULATIONVIEW$(A_t^{(m)}, I_t^{(m)}, s_t, \mathcal{H}_t)$
▷ Frontier-model intervention
$(p_N, d_N) \sim \mathcal{M}_{\mathrm{N}}(\cdot \mid \mathcal{T}, V_t^{(m)}, m_t)$
$y_N \leftarrow F(p_N)$; $t \leftarrow t + 1$
if y_N is invalid **then**
  $\mathcal{H}_t \leftarrow$ RECORDFAILURE$(\mathcal{H}_t, p_N, d_N)$
  return $(\mathcal{A}_t, \mathcal{H}_t, t)$
$(\mathcal{A}_t, \mathcal{H}_t) \leftarrow$ UPDATE$(\mathcal{A}_t, \mathcal{H}_t, p_N, d_N, y_N)$
▷ Lightweight refinement of the Navigator proposal
for $j = 1, \ldots, J$ **in parallel do**
  $(p_{N,j}, d_{N,j}) \sim \mathcal{M}_{\mathrm{S}}(\cdot \mid \mathcal{T}, p_N, d_N)$
  $(\cdot, \mathcal{A}_t, \mathcal{H}_t, t) \leftarrow$ EVALUATEUPDATE$(p_{N,j}, d_{N,j}, F, \mathcal{A}_t, \mathcal{H}_t, t)$
return $(\mathcal{A}_t, \mathcal{H}_t, t)$

Mode	Full-code anchors	Description-only inspirations	Requested move
SYNTHESIS	top 3 niche elites	next 5 niche elites	combine mechanisms across anchors
SURGICAL	global champion only	next 5 niche elites	one local structural correction
REFRAME	top 2 niche elites	next 5 niche elites	introduce an absent strategy family

Table 4: Mode-specific context provided to the Navigator.

REFRAME uses two anchors to identify shared assumptions while relying on descriptions to seek a qualitatively different strategy family.

D.4 ADVISOR INTERVENTION

Algorithm 3 summarizes the overall workflow of an Advisor intervention, from extracting niche-level search signals and recurrent failures to distilling guidance for subsequent Speculator generations.

Tracking niche activity and progress. At each Advisor checkpoint, we inspect the evaluations completed since the previous checkpoint. For every occupied niche k, we maintain two signals: an *activity count* a_k, equal to the number of evaluated candidates assigned to that niche during the interval, and a *movement indicator* m_k, which records whether the niche incumbent was replaced during the same interval. Importantly, the activity count includes unsuccessful candidates. Hence, a niche can receive substantial search effort without producing a better incumbent.

These signals distinguish productive search from saturation without requiring a task-specific improvement threshold. A niche with recent incumbent movement is treated as actively improving,

Algorithm 3 Advisor checkpoint

Require: Task specification $\mathcal{T}$, archive $\mathcal{A}_t$, history $\mathcal{H}_t$, evaluation index t, previous Advisor checkpoint t_A, previous advice ℓ, Advisor $\mathcal{M}_{\mathrm{A}}$
Ensure: Updated advice ℓ

$(a_k, m_k)_k \leftarrow \textsc{RegionSignals}(\mathcal{A}_t, \mathcal{H}_t, t_A, t)$ ▷ activity and movement
$D_t^{\mathrm{work}} \leftarrow \textsc{TopScore}(\{\mathcal{A}_t(k) : m_k = 1\}, 4)$
$D_t^{\mathrm{sat}} \leftarrow \textsc{TopActivity}(\{k : m_k = 0 \wedge a_k > 0\}, 3)$

$\mathcal{R}_t \leftarrow \textsc{AggregateFailures}(\mathcal{H}_t)$ ▷ cumulative counts; no eviction
$E_t \leftarrow \textsc{MostFrequent}(\mathcal{R}_t, 8)$ ▷ break ties by recency

$X_t \leftarrow (D_t^{\mathrm{work}}, D_t^{\mathrm{sat}}, E_t, \ell, y_t^{\mathrm{best}}, s_t)$
$\ell \sim \mathcal{M}_{\mathrm{A}}(\cdot \mid \mathcal{T}, X_t)$ ▷ distill structured guidance
return ℓ

whereas a niche with high activity but no movement is considered saturated. This distinction is particularly useful across our heterogeneous benchmarks, where the scale of score improvements varies substantially between tasks.

At each checkpoint, the Advisor constructs two complementary population views:

- D_t^{work}: up to four highest-scoring niches whose incumbents moved during the current interval;
- D_t^{sat}: up to three most active niches that received evaluations but showed no incumbent movement.

Together, these views separate current solution quality from recent search progress and exploration coverage. In particular, a high-scoring niche need not be improving, while a lower-scoring niche may reveal a productive direction that has only recently emerged.

Failure memory. We aggregate evaluator failures from the complete search history $\mathcal{H}_t$, including initialization, using normalized error signatures. For each signature, $\mathcal{R}_t$ records its cumulative occurrence count, most recent occurrence, and a representative message. No signatures are evicted during a run. At each Advisor checkpoint, we select the eight most frequent signatures, breaking ties by recency. This bounds the failure evidence included in the prompt while retaining earlier failures in the underlying history. The Advisor combines this evidence with recent niche activity and its previous guidance to revise the next note.

Advisor output and guidance injection. The Advisor converts the population views, recurrent failures, and previous advice into a compact note with three fields. WORKING summarizes mechanisms associated with leading or recently improving niches; SATURATED identifies directions that continue to consume evaluations without moving their incumbents; and AVOID highlights recurrent anti-patterns supported by the failure memory.

We use temperature 0.4 and a 1000-token generation limit, with the resulting note constrained to fewer than 500 words. The note is not imposed on every worker. Instead, each subsequent Speculator receives it independently with probability $\rho_A = 0.35$. This stochastic injection allows useful population-level lessons to propagate through the search while preserving a substantial fraction of independent exploration.

E METRIC DETAILS

E.1 AGGREGATE METRICS USED IN FIGURE 3

Figure 3 summarizes performance and API expenditure across tasks whose raw objective scales differ substantially, so we normalize both quantities within each benchmark family f: the seven mathematical discovery tasks and the four systems optimization tasks (all evaluators are higher-is-better). Let $\mathcal{T}_f$ be the tasks in f, $\mathcal{C}_f$ the method–backbone configurations shown in the corresponding panel, and $\bar{s}_{c,t}$ and $\bar{u}_{c,t}$ the final best score and the API cost (USD) of configuration c on task t, each

averaged over the three runs. We report

$$\mathrm{Perf}(c; f) = \frac{100}{|\mathcal{T}_f|} \sum_{t \in \mathcal{T}_f} \frac{\bar{s}_{c,t}}{\max_{c' \in \mathcal{C}_f} \bar{s}_{c',t}}, \qquad \mathrm{Cost}(c; f) = 100 \, \frac{\bar{u}_{c;f}}{\max_{c' \in \mathcal{C}_f} \bar{u}_{c';f}},$$

where $\bar{u}_{c;f}$ is the mean of $\bar{u}_{c,t}$ over $t \in \mathcal{T}_f$. Every task thus contributes equally regardless of its native scale, and 100% performance corresponds to matching the strongest displayed mean on every task. Higher normalized performance and lower normalized cost are preferable; the ***Best*** scores in the detailed tables are complementary statistics that are not used in this aggregate.

E.2 Code Diversity Index (CDI)

CDI (Dat et al., 2025) measures how evenly the population's code representations are separated along a minimum spanning tree (MST). At checkpoint t, let $\mathcal{Q}_t = \{p_1, \ldots, p_n\}$ be the complete archive snapshot. We first canonicalize each source string. For each Python program, we parse its abstract syntax tree, remove comments and module/function/class docstrings, and emit a deterministically formatted program. If parsing fails, we fall back to whitespace-normalized source. Let

$$x_i = f_{\mathrm{emb}}(\mathrm{canon}(\mathrm{code}(p_i))) \in \mathbb{R}^d \tag{6}$$

be the embedding of the resulting code. We use `text-embedding-3-small` for all methods and checkpoints.

Construct the complete undirected graph on the n embeddings with Euclidean edge weight

$$d_{ij} = \|x_i - x_j\|_2. \tag{7}$$

Let $\mathcal{E}_t^{\mathrm{MST}}$ be the edge set of a minimum spanning tree, and let $\mathcal{E}_t^{+} = \{e \in \mathcal{E}_t^{\mathrm{MST}} : d_e > 0\}$ remove zero-length duplicate edges. Normalize its positive edge lengths as

$$\pi_e = \frac{d_e}{\sum_{e' \in \mathcal{E}_t^{+}} d_{e'}}. \tag{8}$$

The reported Code Diversity Index is the Shannon entropy

$$\mathrm{CDI}(\mathcal{Q}_t) = -\sum_{e \in \mathcal{E}_t^{+}} \pi_e \log \pi_e, \tag{9}$$

where $\log$ is the natural logarithm. If there is no positive MST edge (e.g., $n \le 1$ or all embeddings coincide), CDI is defined as zero.

CDI is high when separation is distributed across many MST edges instead of being dominated by one bridge between otherwise collapsed groups. Since the edge lengths are normalized, uniform rescaling of every embedding distance does not change CDI; the metric captures the distribution of population separations, not an absolute diameter. With $m = |\mathcal{E}_t^{+}| \ge 1$ positive edges, $0 \le \mathrm{CDI} \le \log m \le \log(n-1)$, with the upper bound attained when the positive MST edges have equal length. Because this upper bound depends on population size, comparisons should use checkpoints with comparable archive sizes. Our computation keeps duplicate programs by default and uses the archive exactly as stored at the checkpoint.

E.3 Stagnation rate

The reported *stagnation rate* is a post-hoc, run-level statistic that asks what fraction of the full evaluation budget was consumed by the single longest global-best plateau.

For a run of T evaluations, define the new-best index set

$$\mathcal{I}_{\mathrm{best}} = \left\{ t \in \{1, \ldots, T\} : y_t > \max_{1 \le j < t} y_j \right\} = \{b_1 < \cdots < b_m\}. \tag{10}$$

Set the sentinels $b_0 = 0$ and $b_{m+1} = T + 1$. The number of consecutive non-new-best evaluations between two adjacent events is $g_i = b_{i+1} - b_i - 1$. This convention includes the prefix before the first new best and the suffix after the last new best. The run-level metric is

$$\mathrm{StagRate} = 100 \times \frac{\max_{0 \le i \le m} g_i}{T} \quad (\%). \tag{11}$$

E.4 NEW BESTS TRACED TO THE FRONTIER MODEL

This metric measures how often observed progress can be attributed to the frontier model. In LEVI, the frontier model is invoked through `paradigm_shift`, while in SPECEVO it serves as the Navigator. For each new-best event b_i in Eq. 10, define

$$z(b_i) = \begin{cases} 1, & \text{if } b_i \text{ is generated by the frontier model, or by a small model} \\ & \text{with at least one frontier-generated parent,} \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The run-level rate is

$$\mathrm{FrontierTrace} = 100 \times \frac{\sum_{i=1}^{m} z(b_i)}{m} \quad (\%), \tag{13}$$

where m is the number of new-best events. Thus, $\mathrm{FrontierTrace}$ measures the fraction of improvements that are either directly produced by the frontier model or arise from lightweight refinement of its programs. Initialization is excluded, and the metric is reported as missing when $m = 0$. Since it counts improvement events rather than their magnitude, it should be interpreted together with final performance and API cost.

F BENCHMARK DETAILS

We provide additional details about the benchmarks used in our evaluation. Unless otherwise stated, a candidate solution is represented as an executable program and evaluated by a task-specific evaluator. The evolutionary methods only observe the resulting scalar score and diagnostic feedback, without access to hidden test instances or ground-truth solutions.

F.1 MATHEMATICAL OPTIMIZATION BENCHMARKS

We consider seven executable mathematical optimization tasks covering geometric packing, extremal point configurations, and signal processing. The geometric tasks follow the formulations used in AlphaEvolve (Novikov et al., 2025). Each candidate program constructs or refines a feasible solution, and the evaluator returns a scalar higher-is-better objective value. The exact configurations used in our experiments are summarized in Table 5.

F.2 SYSTEMS OPTIMIZATION BENCHMARKS

We use four real-world systems optimization tasks from the AI-Driven Research for Systems benchmark suite (Cheng et al., 2025). Although some underlying objectives are naturally formulated as minimization problems, the task evaluators convert them into scalar scores for which a larger value indicates a better candidate in all reported experiments. Table 6 summarizes the four tasks included in our evaluation.

F.3 CO-BENCH

CO-Bench (Sun et al., 2026) evaluates the ability of language-model agents to design complete algorithms for combinatorial optimization rather than directly predicting solutions to individual instances. It contains 36 real-world optimization problems spanning eight categories and a broad range of problem sizes and constraint structures. Most instances are curated from OR-Library and related operations-research collections.

For each problem, the benchmark provides the agent with a natural language problem specification, a starter `solve` interface, and a development set. A hidden evaluation function checks feasibility and measures solution quality. The test instances and evaluator implementation are unavailable during search, reducing direct overfitting to the final evaluation data.

Let $h(x; p)$ denote the raw objective value of a feasible solution x on instance p, and let h_p^* denote the reference value recorded in the benchmark. We use the task-specific normalization implemented

Table 5: Mathematical optimization benchmarks used in our evaluation.

Problem	Configuration	Objective and task definition
Circle Packing (Square)	$N = 26$	Pack N pairwise-disjoint circles inside a unit square and maximize the sum of their radii.
Circle Packing (Rectangle)	$N = 21$	Pack N pairwise-disjoint circles inside a rectangle with perimeter 4 and maximize the sum of their radii.
Heilbronn Problem (Triangle)	$N = 11$	Place N points on or inside a triangle of unit area so as to maximize the minimum area of any triangle formed by three selected points.
Heilbronn Problem (Convex Region)	$N = 13$	Place N points on or inside a convex region of unit area so as to maximize the minimum area of any triangle formed by three selected points.
MinMax Distance	$(N, d) = (16, 2)$	Find N points in $\mathbb{R}^d$ that maximize the ratio between the minimum and maximum pairwise Euclidean distances.
MinMax Distance	$(N, d) = (14, 3)$	Optimize the same MinMax-distance objective for 14 points in three-dimensional Euclidean space.
Signal Processing		Optimize a continuous signal processing objective under noisy evaluation, where candidate programs transform an input signal and are scored by an external evaluator.

in CO-Bench, denoted by $\mathcal{N}_p$, to obtain a higher-is-better score:

$$s(x,p) = \begin{cases} \mathcal{N}_p\big(h(x;p), h_p^*\big), & \text{if execution succeeds and } x \text{ is feasible,} \\ 0, & \text{if execution fails, times out, or } x \text{ is infeasible.} \end{cases} \tag{14}$$

For standard positive-reference ratio normalization, $\mathcal{N}_p(h, h_p^*) = h/h_p^*$ for maximization and $\mathcal{N}_p(h, h_p^*) = h_p^*/h$ for minimization, with nonzero denominators. We retain the benchmark's special-case handling, including shifted ratios for zero-reference instances.

The normalized score is always interpreted as higher-is-better, regardless of the original optimization direction. Failed evaluations receive a final normalized score of 0; this value is not substituted into the raw objective before normalization. A score of 1 corresponds to matching the reference, while scores above 1 can occur when a feasible solution improves upon a recorded best-known reference under the benchmark's uncapped ratio normalization.

The score for each problem is averaged over its test instances, and the overall CO-Bench score is the unweighted mean across all 36 problems:

$$S(A) = \frac{1}{36} \sum_{c=1}^{36} \frac{1}{|\mathcal{P}_c|} \sum_{p \in \mathcal{P}_c} s\big(A(p), p\big), \tag{15}$$

where A is a candidate algorithm and $\mathcal{P}_c$ is the test set of problem c. Due to the three-hour wall-clock budget, we sample at most 50 instances for each problem and split them into training and test sets using a 7:3 ratio, enabling a comprehensive comparison between SPECEVO and the baselines under a consistent computational budget.

Table 7 lists all problem families included in our evaluation.

F.4 LSR-SYNTH

We use LSR-Synth from LLM-SRBench (Shojaee et al., 2025b), a benchmark designed to evaluate LLM-based scientific equation discovery. Unlike traditional symbolic regression benchmarks that often contain well-established canonical equations (La Cava et al., 2021; Matsubara et al., 2022), LSR-Synth combines known functional terms with novel synthetic terms. This construction reduces the likelihood that models can solve the tasks through memorization or direct recall of equations encountered during pre-training.

Table 6: Systems optimization benchmarks used in our evaluation.

Task	Objective	Problem description
Expert Parallelism Load Balancer (EPLB)	Balance expert-parallel workloads across GPUs	Given expert load statistics for a mixture-of-experts model, determine the number and placement of expert replicas across physical GPU slots. The objective is to reduce computational skew and stragglers while satisfying placement constraints and preserving inference throughput.
Model Placement (PRISM)	Optimize model-to-GPU placement cost	Assign multiple models or model components to a collection of GPUs under memory and capacity constraints. The placement should minimize global execution cost or peak resource pressure while maintaining feasibility and service-level requirements.
LLM-SQL	Maximize prefix-cache efficiency	Reorder table rows, columns, or fields so that serialized SQL inputs share longer prefixes and obtain higher prefix-cache reuse. The transformation must preserve the original table semantics and query results.
Transaction Scheduling (TXN)	Minimize transaction-execution makespan	Determine an execution order for dependent or conflicting database transactions. The scheduler seeks to reduce total completion time while respecting dependencies and avoiding excessive conflicts, aborts, or waiting time.

Table 7: The 36 combinatorial optimization problems in CO-Bench, grouped into eight categories.

Category	#	Included problems
Packing	9	Bin packing; Multi-Demand Multidimensional Knapsack problem; Multidimensional knapsack problem; Container loading; Container loading with weight restrictions; Packing unequal circles; Packing unequal rectangles and squares number / area.
Cutting	4	Assortment problem; Constrained / unconstrained guillotine cutting; Constrained non-guillotine cutting.
Facility location	4	Capacitated / Uncapacitated warehouse location; Capacitated / Uncapacitated p-median problem.
Scheduling	7	Aircraft landing; Crew scheduling; Common due date scheduling; Flow shop scheduling; Hybrid Reentrant Shop Scheduling; Job shop scheduling; Open shop scheduling.
Routing	3	Traveling salesman problem; Period vehicle routing problem; Resource constrained shortest path.
Assignment	2	Constrained / unconstrained assignment.
Tree	2	Euclidean Steiner; Corporate structuring.
Graph and set	5	Maximal Independent Set; Graph coloring; Equitable partitioning; Set partitioning; Set covering.
Total	**36**	

Each problem provides a scientific context, descriptions of the relevant variables, and an observational dataset. The objective is to discover a mathematical expression that accurately captures the underlying relationship between the input variables and the target quantity. LSR-Synth covers four scientific domains: chemistry, biology, physics, and material science.

Table 8: Statistics of the LSR-Synth problems used in our evaluation.

	Chemistry	Biology	Physics	Material	Overall
# Problems	36	24	44	25	129
Avg. Datapoints (k)	5.0	5.0	5.0	5.0	5.0
Avg. Variants	3.0	3.0	3.9	3.0	3.3

We use all 129 problems from LSR-Synth, comprising 36 chemistry problems, 24 biology problems, 44 physics problems, and 25 material science problems.

During the search process, candidate equations are ranked using

$$\text{Score} = \log_{10}\left(\frac{1}{1+\text{NMSE}_{\text{train}}}\right),$$

where $\text{NMSE}_{\text{train}}$ is computed on the training data. For the final evaluation, we directly report NMSE on both the in-distribution (ID) test set, whose variable ranges follow those seen during training, and the out-of-distribution (OOD) test set, which contains values outside the training ranges to assess extrapolation ability.

Detailed statistics are reported in Table 8.

G Additional Experimental Results

G.1 Breakdown of API Cost and Token Usage

Table 9 reports the OpenRouter list prices of all models used in our experiments, and Figure 13 shows the corresponding input and output token usage on six representative tasks with GPT-5 as the frontier-model backbone. SpecEvo often uses more tokens in total than several baselines, and in some cases more than LEVI, because it hands a large volume of routine search, mutation, and refinement to lightweight Speculators. Higher token usage does not mean higher cost: most of the additional tokens come from inexpensive small models, and costly frontier-model calls are reserved for infrequent Navigator interventions. With GPT-5 output tokens at $10.00 per million versus $0.30 for Qwen3-30B-A3B-Instruct-2507, moving most of the search workload to lightweight models yields substantial savings even when the total token count is larger.

Tables 10 and 11 provide the task-level API expenditure under the fixed three-hour wall-clock budget. Excluding LEVI, SpecEvo reduces API cost by approximately 49.9%–70.7% across the four benchmark–backbone settings. We treat LEVI separately because it also mixes models: it uses a lightweight model for routine search and calls the frontier model only for occasional `paradigm_shift` operations, so it already benefits from the same price gap between lightweight and frontier models. The comparison with LEVI therefore more directly reflects how each method allocates scarce frontier-model reasoning.

G.2 Detailed Results for Mathematical Discovery and Systems Optimization Tasks

The results below break down the aggregate trends in Figure 3 by task and backbone configuration. Beyond final performance, we check whether the gains persist when either the frontier model or the lightweight Speculator is changed, which separates the contribution of the search architecture from that of any particular model.

Mathematical discovery. Across both frontier backbones, SpecEvo attains the strongest or tied-best single-run result on six of the seven tasks and the leading or tied mean performance on five. Its advantage is most consistent on the Heilbronn and MinMax problems, whose landscapes contain multiple structurally distinct solution families and densely populated local optima. The adaptive archive preserves these alternatives, and the Navigator can synthesize promising mechanisms or reframe the search once lightweight refinement saturates. Even smaller Speculators therefore often match the strongest configurations on these tasks. Signal Processing is the main exception: its

Table 9: OpenRouter API pricing for the models used in our experiments, reported in USD per one million tokens (accessed August 1, 2026). Kimi-K2 in the main text, figures, and tables refers to Kimi-K2-Thinking.

Model	Price per 1M Tokens (USD)	
	Input	Output
Frontier Models		
GPT-5	1.25	10.00
Kimi-K2-Thinking	0.60	2.50
Small Models		
Qwen3-30B-A3B-Instruct-2507	0.10	0.30
Qwen3-14B	0.10	0.22
Qwen3-8B	0.11	0.45
Qwen2.5-7B-Instruct	0.10	0.20
Embedding Model		
Text Embedding 3 Small	0.02	

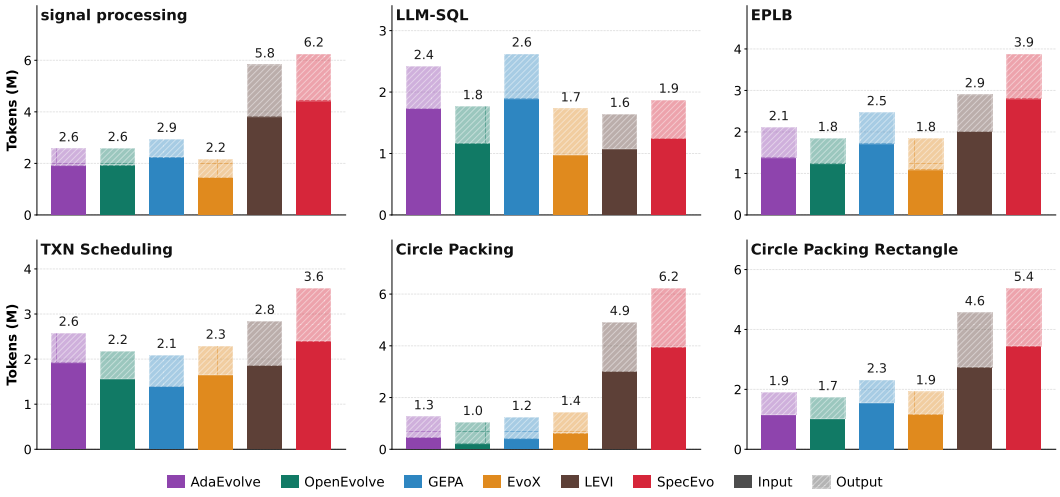

Figure 13: Input and output token usage across six representative tasks using GPT-5 as the frontier-model backbone. Total token consumption is reported in millions (M) for each method.

noisy evaluation makes short-term progress less reliable, so results vary more across lightweight backbones and conventional baselines remain competitive.

Systems optimization. The gains are more uniform on systems tasks, where at least one SPECEVO configuration achieves the best or tied-best mean performance on every problem under both GPT-5 and Kimi-K2. No single lightweight backbone dominates: Qwen3-30B performs strongest on EPLB and LLM-SQL with GPT-5, Qwen3-14B leads Transaction Scheduling, and Qwen3-8B becomes strongest on Transaction Scheduling with Kimi-K2. Even Qwen2.5-7B reaches the performance ceiling on PRISM. This non-monotonic relationship with model size suggests that the gains do not come simply from a more capable Speculator. In the closed-loop workflow, inexpensive models with different strengths carry out high-volume refinement, and population-level evidence determines when strategic redirection is needed.

Robustness and cost efficiency. This division of labor also reduces sensitivity to the frontier backbone. Several frontier-only baselines degrade substantially when GPT-5 is replaced by Kimi-K2, particularly on PRISM, whereas SPECEVO stays close to the task ceiling because the frontier model acts through sparse, high-impact interventions instead of shaping every mutation. The same organi-

Table 10: API cost in USD under the fixed three-hour wall-clock budget on mathematical discovery tasks. Each entry reports the mean and standard deviation over three independent runs. Lower is better.

Strategy	Circle Packing	Circle Rect.	Heil. Conv.	Heil. Tri.	MinMax 16-2	MinMax 14-3	Signal
Backbone: GPT-5							
OpenEvolve	7.65 ± 0.40	8.65 ± 1.01	8.76 ± 2.48	8.53 ± 0.02	8.31 ± 0.28	7.12 ± 0.86	8.53 ± 0.64
GEPA	8.41 ± 0.10	5.20 ± 1.14	8.37 ± 1.04	9.36 ± 1.71	7.82 ± 1.41	8.09 ± 0.55	9.74 ± 1.75
AdaEvolve	8.83 ± 0.31	6.68 ± 1.29	8.62 ± 0.06	8.93 ± 0.86	7.96 ± 0.26	8.58 ± 0.01	7.91 ± 0.91
EvoX	7.66 ± 0.50	7.33 ± 0.68	9.18 ± 0.22	9.12 ± 1.58	8.15 ± 0.56	7.17 ± 0.67	8.72 ± 2.05
LEVI	3.40 ± 1.23	2.62 ± 1.05	**1.60** ± 0.25	1.33 ± 0.19	1.80 ± 0.17	2.45 ± 0.75	**3.64** ± 0.12
SpecEvo	**3.37** ± 1.61	**2.61** ± 1.29	1.62 ± 0.35	**1.32** ± 0.18	**1.65** ± 0.06	**2.33** ± 0.86	3.88 ± 0.01
Backbone: Kimi-K2							
OpenEvolve	1.98 ± 0.20	1.63 ± 0.19	1.64 ± 0.16	1.63 ± 0.36	1.61 ± 0.21	1.52 ± 0.19	2.28 ± 0.28
GEPA	1.95 ± 0.10	1.69 ± 0.05	1.45 ± 0.16	1.45 ± 0.11	1.62 ± 0.04	1.49 ± 0.23	2.18 ± 0.26
AdaEvolve	1.67 ± 0.16	1.79 ± 0.31	1.59 ± 0.12	1.49 ± 0.24	1.63 ± 0.31	1.52 ± 0.09	2.23 ± 0.06
EvoX	1.76 ± 0.15	1.67 ± 0.22	1.62 ± 0.16	1.75 ± 0.22	1.55 ± 0.14	1.66 ± 0.39	2.33 ± 0.05
LEVI	**0.79** ± 0.12	**0.77** ± 0.21	0.45 ± 0.15	0.65 ± 0.30	1.12 ± 0.49	1.04 ± 0.18	**1.39** ± 0.20
SpecEvo	0.83 ± 0.33	0.83 ± 0.22	**0.42** ± 0.21	**0.50** ± 0.27	**1.05** ± 0.27	**1.02** ± 0.26	1.41 ± 0.14

Table 11: API cost in USD under the fixed three-hour wall-clock budget on systems optimization tasks. Each entry reports the mean and standard deviation over three independent runs. Lower is better.

Strategy	EPLB (↑)	LLM-SQL (↑)	Transaction (↑)	PRISM (↑)
Backbone: GPT-5				
OpenEvolve	7.69 ± 0.36	5.16 ± 0.95	2.48 ± 0.25	6.38 ± 0.25
GEPA	10.04 ± 1.26	5.59 ± 0.43	2.38 ± 0.52	4.03 ± 0.37
AdaEvolve	8.26 ± 0.11	6.44 ± 0.40	2.46 ± 0.39	6.22 ± 0.19
EvoX	9.52 ± 0.57	5.19 ± 0.23	2.47 ± 0.31	5.97 ± 0.22
LEVI	2.90 ± 0.25	1.65 ± 0.12	**1.20** ± 0.21	2.42 ± 0.98
SpecEvo	**2.87** ± 0.38	**1.62** ± 0.07	1.21 ± 0.27	**2.41** ± 1.00
Backbone: Kimi-K2				
OpenEvolve	2.25 ± 0.32	1.64 ± 0.05	0.90 ± 0.15	1.97 ± 0.27
GEPA	1.88 ± 0.29	1.55 ± 0.04	1.02 ± 0.28	1.69 ± 0.12
AdaEvolve	2.04 ± 0.07	1.73 ± 0.27	1.15 ± 0.23	1.96 ± 0.14
EvoX	2.24 ± 0.05	1.77 ± 0.13	1.14 ± 0.31	2.16 ± 0.10
LEVI	1.01 ± 0.17	**0.51** ± 0.03	0.40 ± 0.18	**0.62** ± 0.00
SpecEvo	**0.99** ± 0.20	0.53 ± 0.02	**0.39** ± 0.19	0.65 ± 0.00

zation reduces API expenditure by 34%–85% on mathematical discovery and 51%–71% on systems optimization relative to the four frontier-heavy baselines. SPECEVO also generally improves upon LEVI at comparable cost, which indicates that the remaining quality difference comes mainly from how scarce frontier reasoning and accumulated evidence are integrated into the search, not from the model price gap alone.

G.3 DETAILED RESULTS FOR EACH CO-BENCH PROBLEM CATEGORY

Figure 14 summarizes performance across the eight CO-Bench categories, while Table 14 reports the complete results for all 36 problems. SPECEVO achieves the highest category-level score in five categories—Packing, Cutting, Scheduling, Routing, and Tree—with the largest advantages observed on Cutting and Tree problems. At the problem level, SPECEVO ranks first on 20/36 problems, with strong results on assortment, Euclidean Steiner, flow-shop scheduling, and periodic vehicle-routing problems. Assignment is nearly saturated across methods, and results on Facility Location are more mixed.

Figure 15 and Table 15 further examine whether these gains are consistent across individual problems. The mean paired score difference is positive against every baseline, ranging from $+0.0332$ against LEVI to $+0.0500$ against OpenEvolve, and SPECEVO wins more problems than it loses in

Table 12: Performance comparison on mathematical discovery tasks. Red arrows indicate SPECEVO's average API cost reduction relative to the four baselines, excluding LEVI.

Strategy	Circle Packing		Circle Rect.		Heil. Conv.		Heil. Tri.		MinMax 16-2		MinMax 14-3		Signal	
	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best
Backbone: GPT-5														
OpenEvolve	2.541 ± 0.000	2.541	2.341 ± 0.022	2.354	0.022 ± 0.000	0.022	0.033 ± 0.002	**0.036**	0.077 ± 0.000	0.077	0.229 ± 0.008	0.238	0.595 ± 0.005	0.598
GEPA	2.536 ± 0.010	2.541	2.313 ± 0.024	2.327	0.024 ± 0.004	0.029	0.032 ± 0.000	0.032	0.077 ± 0.000	0.077	0.226 ± 0.002	0.227	0.593 ± 0.038	0.636
AdaEvolve	2.563 ± 0.037	2.606	2.296 ± 0.042	2.345	0.022 ± 0.001	0.023	0.029 ± 0.000	0.029	0.077 ± 0.000	0.077	0.235 ± 0.003	0.237	**0.733** ± 0.001	**0.734**
EvoX	2.521 ± 0.035	2.541	**2.352** ± 0.007	2.360	0.027 ± 0.000	0.027	0.033 ± 0.001	0.034	**0.078** ± 0.000	**0.078**	0.239 ± 0.000	0.239	0.688 ± 0.013	0.703
LEVI	2.593 ± 0.001	2.595	2.301 ± 0.040	2.358	0.028 ± 0.001	0.029	0.034 ± 0.000	0.035	0.077 ± 0.000	0.077	0.239 ± 0.000	0.239	0.593 ± 0.068	0.691
SPECEVO	↓59% ($)		↓63% ($)		↓81% ($)		↓85% ($)		↓80% ($)		↓70% ($)		↓56% ($)	
Qwen3-30b	2.580 ± 0.041	2.627	2.338 ± 0.021	**2.363**	**0.030** ± 0.000	**0.030**	0.034 ± 0.002	**0.036**	0.077 ± 0.000	**0.078**	**0.240** ± 0.000	**0.240**	0.711 ± 0.012	0.717
Qwen3-14b	**2.629** ± 0.001	2.630	2.306 ± 0.012	2.342	0.029 ± 0.001	**0.030**	**0.036** ± 0.000	**0.036**	0.077 ± 0.000	**0.078**	**0.240** ± 0.000	**0.240**	0.545 ± 0.012	0.552
Qwen3-8b	2.572 ± 0.055	**2.636**	2.351 ± 0.008	2.361	**0.030** ± 0.000	**0.030**	0.035 ± 0.001	**0.036**	0.077 ± 0.000	**0.078**	**0.240** ± 0.000	**0.240**	0.550 ± 0.004	0.553
Qwen2.5-7b	2.517 ± 0.054	2.548	2.333 ± 0.052	**2.363**	0.025 ± 0.001	0.026	0.034 ± 0.002	**0.036**	**0.078** ± 0.000	**0.078**	**0.240** ± 0.000	**0.240**	0.708 ± 0.009	0.713
Backbone: Kimi-K2														
OpenEvolve	2.379 ± 0.109	2.500	2.184 ± 0.171	2.296	0.018 ± 0.001	0.019	0.026 ± 0.001	0.027	0.068 ± 0.011	0.075	0.213 ± 0.004	0.218	**0.671** ± 0.045	**0.709**
GEPA	2.429 ± 0.062	2.481	2.167 ± 0.145	2.251	0.019 ± 0.002	0.022	0.021 ± 0.014	0.035	0.068 ± 0.011	0.075	0.211 ± 0.000	0.211	0.643 ± 0.020	0.658
AdaEvolve	**2.595** ± 0.037	2.630	2.256 ± 0.005	2.261	0.025 ± 0.002	0.026	0.028 ± 0.005	0.031	0.074 ± 0.004	**0.078**	0.220 ± 0.008	0.227	0.614 ± 0.090	0.693
EvoX	2.561 ± 0.083	2.617	2.266 ± 0.063	2.336	0.024 ± 0.003	0.026	0.025 ± 0.003	0.027	0.073 ± 0.002	0.075	0.213 ± 0.004	0.218	0.655 ± 0.031	0.677
LEVI	2.553 ± 0.045	2.618	2.185 ± 0.121	2.357	0.023 ± 0.001	0.025	0.033 ± 0.000	0.034	0.076 ± 0.001	0.077	0.237 ± 0.000	0.238	0.597 ± 0.036	0.623
SPECEVO	↓55% ($)		↓51% ($)		↓73% ($)		↓68% ($)		↓34% ($)		↓34% ($)		↓37% ($)	
Qwen3-30b	2.593 ± 0.037	**2.635**	**2.334** ± 0.029	**2.361**	0.026 ± 0.001	0.027	0.031 ± 0.002	0.033	**0.078** ± 0.000	**0.078**	0.223 ± 0.012	0.236	0.629 ± 0.065	0.703
Qwen3-14b	2.539 ± 0.035	2.559	2.278 ± 0.007	2.282	**0.030** ± 0.001	**0.031**	0.030 ± 0.007	0.034	**0.078** ± 0.000	**0.078**	**0.239** ± 0.001	**0.240**	0.624 ± 0.045	0.650
Qwen3-8b	2.584 ± 0.080	2.631	2.269 ± 0.146	2.353	0.027 ± 0.000	0.027	**0.034** ± 0.003	**0.036**	**0.078** ± 0.000	**0.078**	0.229 ± 0.009	0.239	0.613 ± 0.065	0.650
Qwen2.5-7b	2.489 ± 0.013	2.497	2.315 ± 0.055	2.347	0.023 ± 0.003	0.025	0.033 ± 0.003	0.035	0.073 ± 0.005	0.076	0.238 ± 0.003	**0.240**	0.615 ± 0.020	0.645

Table 13: Performance comparison on systems optimization tasks.

Strategy	EPLB (↑)		LLM-SQL (↑)		Transaction (↑)		PRISM (↑)	
	Avg	Best	Avg	Best	Avg	Best	Avg	Best
Backbone: GPT-5								
OpenEvolve	0.1398 ± 0.011	0.1459	0.7152 ± 0.024	0.7289	4006.0 ± 65.47	4081.6	26.242 ± 0.008	26.247
GEPA	0.1394 ± 0.010	0.1453	0.7242 ± 0.006	0.7315	4078.2 ± 213.73	4201.6	26.235 ± 0.002	26.236
AdaEvolve	0.1462 ± 0.000	0.1464	0.7287 ± 0.004	0.7307	3958.7 ± 195.14	4184.1	**26.256** ± 0.000	**26.256**
EvoX	0.1448 ± 0.001	0.1454	0.6936 ± 0.031	0.7298	3827.8 ± 354.60	4237.3	**26.256** ± 0.000	**26.256**
LEVI	0.1446 ± 0.0001	0.1449	0.7228 ± 0.0141	0.7328	3777.1 ± 180.37	4032.2	**26.256** ± 0.000	**26.256**
SPECEVO	↓68% ($)		↓71% ($)		↓51% ($)		↓57% ($)	
Qwen3-30b	**0.1472** ± 0.001	**0.1485**	**0.7327** ± 0.003	**0.7347**	4173.3 ± 71.01	4255.3	**26.256** ± 0.000	**26.256**
Qwen3-14b	0.1458 ± 0.0005	0.1461	0.7226 ± 0.0147	0.7311	**4366.8** ± 0.0000	**4366.8**	**26.256** ± 0.000	**26.256**
Qwen3-8b	0.1468 ± 0.0008	0.1472	0.7268 ± 0.0051	0.7307	4132.2 ± 406.34	**4366.8**	**26.256** ± 0.000	**26.256**
Qwen2.5-7b	0.1427 ± 0.0021	0.1438	0.7210 ± 0.0130	0.7307	4080.4 ± 65.04	4184.1	**26.256** ± 0.000	**26.256**
Backbone: Kimi-K2								
OpenEvolve	0.1339 ± 0.010	0.1453	0.6980 ± 0.005	0.7012	3966.1 ± 368.9	4237.2	23.443 ± 0.922	24.498
GEPA	0.1287 ± 0.001	0.1297	0.7043 ± 0.003	0.7071	3748.1 ± 126.7	3891.0	23.049 ± 0.932	24.120
AdaEvolve	0.1312 ± 0.005	0.1373	0.6996 ± 0.021	0.7139	3761.1 ± 193.5	3984.0	26.242 ± 0.019	**26.256**
EvoX	0.1337 ± 0.010	0.1451	0.7038 ± 0.015	0.7181	3613.9 ± 143.6	3759.3	25.436 ± 1.229	26.233
LEVI	0.1446 ± 0.000	0.1449	0.6654 ± 0.013	0.6843	4201.6 ± 0.000	4201.6	24.342 ± 0.034	24.498
SPECEVO	↓53% ($)		↓68% ($)		↓63% ($)		↓67% ($)	
Qwen3-30b	**0.1464** ± 0.001	**0.1470**	0.7034 ± 0.016	0.7201	4143.2 ± 178.8	4255.3	26.212 ± 0.076	**26.256**
Qwen3-14b	0.1457 ± 0.0006	0.1464	**0.7080** ± 0.0237	**0.7217**	4183.0 ± 175.7	4385.9	**26.256** ± 0.000	**26.256**
Qwen3-8b	0.1438 ± 0.0002	0.1440	0.6909 ± 0.0282	0.7072	**4398.9** ± 22.40	**4424.8**	26.240 ± 0.013	**26.256**
Qwen2.5-7b	0.1448 ± 0.0008	0.1453	0.6996 ± 0.0234	0.7130	4032.5 ± 354.4	4237.2	**26.256** ± 0.000	**26.256**

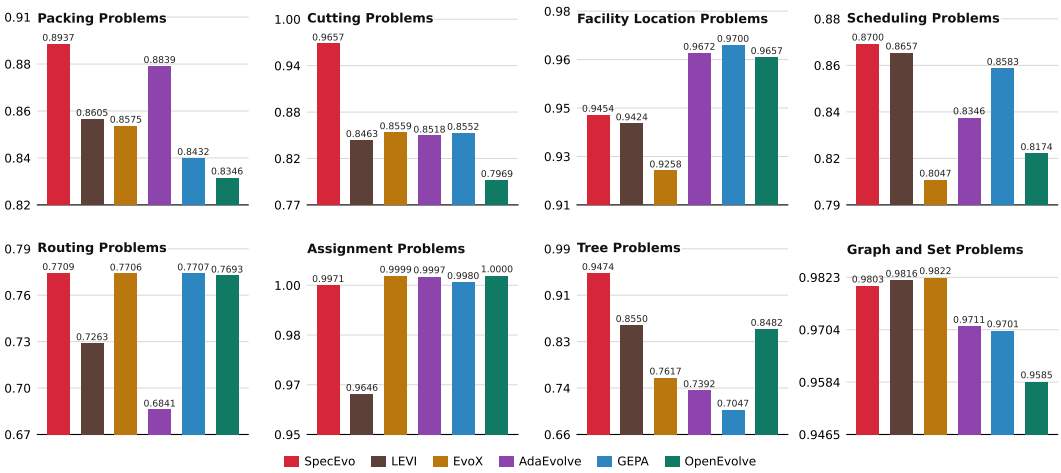

Figure 14: Category-level performance on CO-Bench. Each bar represents the average normalized score over all problems within the corresponding category. Higher is better.

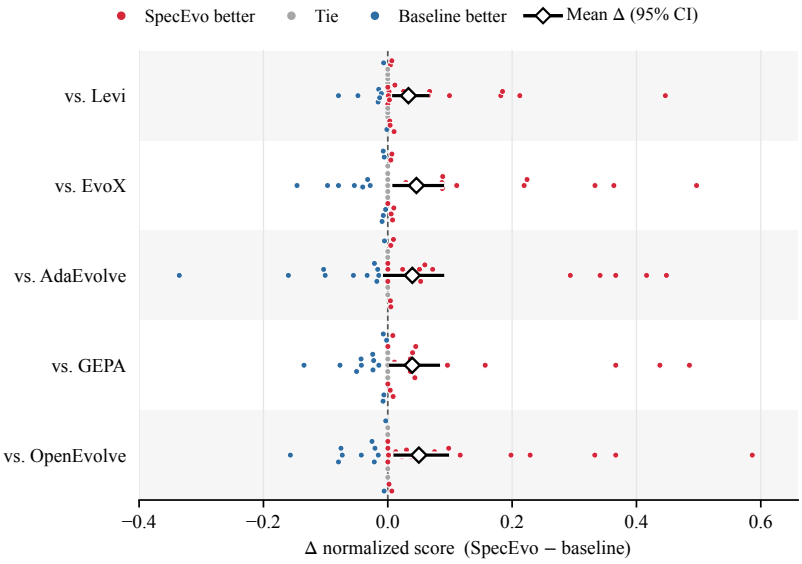

Figure 15: Per-task score differences on the 36 CO-Bench tasks.

all five pairwise comparisons. The bootstrap confidence intervals for the mean difference are also positive for four of the five baselines, with AdaEvolve being the exception.

G.4 Detailed Results on LSR-Synth

We extend the comparison on LSR-Synth to three additional LLM-based equation discovery methods: LLM-SR (Shojaee et al., 2025a), LaSR (Grayeli et al., 2024), and SGA (Ma et al., 2024a). In addition to normalized mean squared error (NMSE), we report numeric precision ($\mathrm{Acc}_{0.1}$) and symbolic accuracy (SA), following the evaluation protocol of LLM-SRBench (Shojaee et al., 2025b).

For a discovered equation, accuracy to tolerance τ is defined as

$$\mathrm{Acc}_\tau = \mathbb{I}\left[\max_{1\le i\le N_{\text{test}}} \left|\frac{\hat{y}_i - y_i}{y_i}\right| \le \tau\right], \tag{16}$$

Table 14: Per-task normalized scores on the 36 CO-Bench tasks. Higher scores indicate better performance.

Task	OpenEvolve	GEPA	AdaEvolve	EvoX	LEVI	SPECEVO
Packing Problems						
Bin packing – one-dimensional	0.8963	0.8903	0.9213	0.8979	0.8870	**0.9863**
Multi-Demand Multidimensional Knapsack	0.4473	0.5196	0.6234	0.5893	0.6090	**0.6763**
Multidimensional knapsack problem	0.9867	0.9846	0.9890	0.9866	0.9890	**0.9932**
Container loading	0.8362	0.8392	0.8281	0.8156	**0.8396**	0.8329
Container loading with weight restrictions	0.3540	0.3577	**0.6162**	0.3777	0.2808	0.2808
Packing unequal circles	0.9666	0.9666	0.9666	1.0000	1.1209	**1.3333**
Packing unequal circles area	1.0293	**1.0913**	1.0847	1.0574	1.0686	1.0671
Packing unequal rectangles and squares	0.9616	0.9061	**0.9855**	0.9616	0.9616	0.8823
Packing unequal rectangles and squares area	**1.0338**	1.0337	0.9402	1.0315	0.9880	0.9912
Cutting Problems						
Assortment problem	0.4164	0.5650	0.5545	0.5060	0.5562	**1.0027**
Constrained guillotine cutting	0.8709	0.9247	0.9569	**0.9892**	0.9176	0.9354
Unconstrained guillotine cutting	0.9460	**0.9829**	0.9636	0.9777	0.9571	0.9685
Constrained non-guillotine cutting	0.9542	0.9481	0.9323	0.9508	0.9542	**0.9562**
Facility Location Problems						
Capacitated warehouse location	0.8732	0.9122	0.8763	0.7245	0.9232	**0.9485**
Uncapacitated warehouse location	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Capacitated p-median	0.9896	0.9677	**0.9928**	0.9789	0.8475	0.8329
Uncapacitated p-median	0.9998	**1.0000**	0.9998	**1.0000**	0.9990	**1.0000**
Scheduling Problems						
Aircraft landing	0.9026	0.9047	**0.9827**	0.9144	0.8976	0.8820
Crew scheduling	0.5285	0.4682	**0.5294**	0.2939	0.5132	0.5132
Common due date scheduling	0.8887	0.9468	0.9271	**0.9902**	0.9802	0.9867
Flow shop scheduling	0.6364	0.9356	0.6279	0.8585	0.9300	**0.9694**
Hybrid Reentrant Shop Scheduling	0.9917	0.9917	0.9917	0.9917	0.9917	**0.9918**
Job shop scheduling	0.8116	0.8238	0.8416	0.7849	0.8501	**0.8602**
Open shop scheduling	**0.9621**	0.9370	0.9421	0.7991	0.8971	0.8868
Routing Problems						
Traveling salesman problem	0.9465	0.9641	0.9543	0.9494	**0.9691**	0.9211
Period vehicle routing problem	0.7527	0.7392	0.4892	0.7538	0.6010	**0.7828**
Resource-constrained shortest path	**0.6087**	**0.6087**	**0.6087**	**0.6087**	**0.6087**	**0.6087**
Assignment Problems						
Assignment problem	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Generalized assignment problem	**1.0000**	0.9960	0.9994	0.9998	0.9292	0.9941
Tree Problems						
Euclidean Steiner problem	0.6965	0.4095	0.4784	0.5310	0.7101	**0.8947**
Corporate structuring	0.9999	0.9999	0.9999	0.9923	0.9999	**1.0000**
Graph and Set Problems						
Maximal Independent Set	**0.9347**	0.9202	0.9275	0.9202	0.9261	0.9130
Graph coloring	0.8835	0.9382	0.9500	**1.0000**	**1.0000**	**1.0000**
Equitable partitioning problem	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Set partitioning	0.9912	**1.0000**	0.9912	0.9997	0.9898	0.9924
Set covering	0.9831	0.9921	0.9869	0.9911	0.9923	**0.9960**

Table 15: Paired statistical comparison of SPECEVO against baselines on the 36 CO-Bench tasks.

Baseline	W / T / L	Mean Δ	95% CI
LEVI	21 / 7 / 8	$+0.0332$	$[+0.0070, +0.0668]$
EvoX	18 / 6 / 12	$+0.0460$	$[+0.0073, +0.0906]$
AdaEvolve	21 / 4 / 11	$+0.0395$	$[-0.0076, +0.0908]$
GEPA	18 / 5 / 13	$+0.0393$	$[+0.0020, +0.0841]$
OpenEvolve	21 / 4 / 11	$+0.0500$	$[+0.0091, +0.0985]$

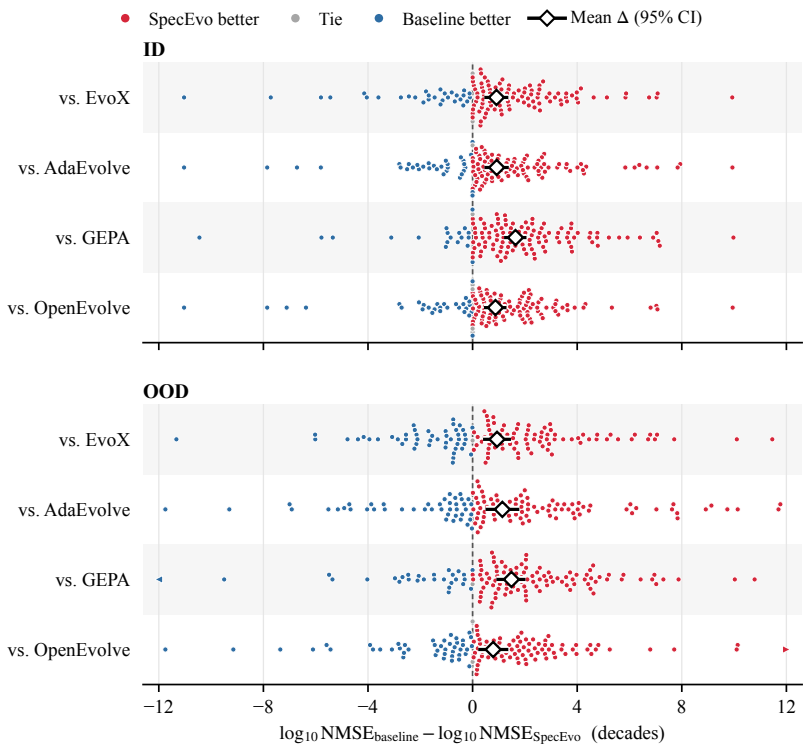

Figure 16: Per-task score differences on the 129 symbolic regression tasks.

where $\hat{y}_i$ and y_i denote the predicted and ground-truth values, respectively. We use $\tau = 0.1$ and report the percentage of problems satisfying this criterion. While NMSE measures aggregate prediction error, $\mathrm{Acc}_{0.1}$ requires relative error to remain within the specified tolerance at every test point. We compute both metrics separately on the in-distribution (ID) and out-of-distribution (OOD) test sets to assess interpolation and extrapolation performance.

Symbolic accuracy complements these numerical metrics by assessing whether the discovered hypothesis recovers the ground-truth symbolic form. Following LLM-SRBench (Shojaee et al., 2025b), we use GPT-4o to judge mathematical equivalence between the predicted and ground-truth expressions after removing parameters and constants. This model-based assessment accommodates different representations of equivalent expressions without requiring an exact syntactic match. We report SA as the percentage of problems judged symbolically equivalent. Unlike the numerical metrics, SA compares the equations themselves and is therefore reported once per domain, rather than separately for the ID and OOD splits.

Table 16 summarizes the domain-level results under a common budget of 500 candidate evaluations. We find that SPECEVO achieves the lowest NMSE in six of the eight domain–split settings, including three of the four OOD settings. Biology is the exception, where LLM-SR obtains lower NMSE on both splits. For numeric precision, SPECEVO achieves the highest values on both splits in chemistry, biology, and physics. In material science, it ties for the highest ID precision, while several baselines achieve higher OOD precision. The table further shows that SPECEVO achieves the highest SA in biology (41.66%) and physics (11.36%), while tying for the highest SA in chemistry (13.88%) and material science (24.0%). These results point to improvements in both numerical prediction and symbolic-form recovery, and the consistently low OOD NMSE is further evidence of generalization beyond the training ranges.

Figure 16 and Table 17 provide a task-level comparison on the 129 LSR-Synth problems under both ID and OOD evaluation. For each problem, we measure the paired difference as $\Delta = \log_{10} \mathrm{NMSE}_{\mathrm{baseline}} - \log_{10} \mathrm{NMSE}_{\mathrm{SPECEVO}}$, such that positive values indicate lower NMSE for SPECEVO. Across all four baselines, SPECEVO wins substantially more problems than it loses

Table 16: Performance comparison on LSR-Synth under a fixed budget of 500 candidate evaluations. Metrics include normalized mean squared error (NMSE), numeric precision ($\mathrm{Acc}_{0.1}$), and symbolic accuracy (SA).

Domain	Method	NMSE (↓)		$\mathrm{Acc}_{0.1}$ (%) (↑)		SA (%) (↑)
		ID	OOD	ID	OOD	
Chemistry	LLM-SR	5.69e-07	1.98e-02	36.1	44.3	**13.88**
	LaSR	6.43e-05	5.46e-01	28.9	30.5	8.33
	SGA	8.12e-06	5.25e-02	31.5	41.4	0.0
	OpenEvolve	5.27e-06	4.99e-02	33.3	41.7	5.55
	GEPA	1.72e-05	2.69e-01	19.4	38.9	11.11
	AdaEvolve	2.48e-06	6.05e-02	33.3	41.7	13.88
	EvoX	6.87e-06	8.68e-02	27.8	44.4	11.11
	RelayEvolve	2.59e-06	1.30e-01	36.1	38.9	5.55
	LEVI	5.80e-06	1.17e-01	27.8	42.5	8.33
	SpecEvo	**4.57e-07**	**1.68e-02**	**41.7**	**55.6**	**13.88**
Biology	LLM-SR	**3.30e-07**	1.08e-03	19.2	23.8	33.33
	LaSR	2.98e-06	3.14e-02	18.5	21.7	4.16
	SGA	3.47e-05	1.08e-02	8.3	12.5	4.16
	OpenEvolve	6.33e-05	7.48e-01	8.3	12.5	29.16
	GEPA	8.79e-05	1.92e+00	8.3	12.5	12.5
	AdaEvolve	9.92e-06	1.28e-01	12.5	16.7	29.16
	EvoX	3.79e-05	5.83e-01	8.3	16.7	29.16
	RelayEvolve	1.01e-05	1.56e-01	8.3	12.5	33.33
	LEVI	3.87e-06	1.96e-02	8.3	12.5	12.5
	SpecEvo	**3.34e-07**	**3.74e-03**	**20.8**	**29.2**	**41.66**
Physics	LLM-SR	1.75e-05	2.63e-03	10.0	12.3	9.09
	LaSR	2.46e-05	2.33e-03	6.8	11.4	0.00
	SGA	5.21e-06	1.35e-02	11.5	16.4	0.00
	OpenEvolve	3.29e-04	6.93e-03	4.5	11.4	0.00
	GEPA	1.53e-03	2.82e-02	0.0	6.8	6.81
	AdaEvolve	2.79e-04	1.32e-02	6.8	11.4	9.09
	EvoX	6.66e-04	5.32e-02	6.8	6.8	6.81
	RelayEvolve	1.81e-04	3.30e-03	2.3	9.1	0.00
	LEVI	1.38e-05	2.23e-03	5.6	12.8	9.09
	SpecEvo	**2.92e-06**	**4.75e-04**	**20.5**	**20.5**	**11.36**
Material Science	LLM-SR	5.24e-08	1.92e-06	**52.0**	88.0	12
	LaSR	6.25e-07	4.75e-5	36.0	72.0	8
	SGA	4.26e-07	1.88e-06	48.0	**96.0**	8
	OpenEvolve	5.92e-08	2.44e-06	48.0	**96.0**	**24**
	GEPA	1.86e-07	7.57e-05	28.0	**96.0**	**24**
	AdaEvolve	6.32e-07	1.18e-04	44.0	72.0	12
	EvoX	9.05e-08	1.63e-05	36.0	**96.0**	16
	RelayEvolve	5.79e-08	6.72e-06	48.0	**96.0**	20
	LEVI	5.73e-07	8.36e-06	36.0	68.0	20
	SpecEvo	**4.84e-08**	**1.68e-06**	**52.0**	88.0	**24**

under both evaluation settings. On ID data, it wins 95–109 of the 129 problems, compared with 19–33 losses, while on OOD data it wins 81–100 problems compared with 28–47 losses. Mean paired differences are also positive for every baseline, ranging from $+0.87$ to $+1.64$ decades on ID data and from $+0.79$ to $+1.48$ decades on OOD data, with all bootstrap 95% confidence intervals remaining above zero. The strongest separation is observed against GEPA, for which SpecEvo wins 109/129 ID problems and 100/129 OOD problems, with mean differences of $+1.64$ and $+1.48$

Table 17: Paired comparison of SPECEVO against baselines on LSR-Synth under the ID and OOD evaluation settings.

Split	Baseline	W / T / L	Mean Δ	95% CI
ID	EvoX	97 / 2 / 30	$+0.91$	$[+0.45, +1.37]$
ID	AdaEvolve	95 / 1 / 33	$+0.92$	$[+0.45, +1.38]$
ID	GEPA	109 / 1 / 19	$+1.64$	$[+1.20, +2.06]$
ID	OpenEvolve	96 / 4 / 29	$+0.87$	$[+0.44, +1.30]$
OOD	EvoX	81 / 1 / 47	$+0.93$	$[+0.40, +1.47]$
OOD	AdaEvolve	85 / 0 / 44	$+1.14$	$[+0.49, +1.79]$
OOD	GEPA	100 / 1 / 28	$+1.48$	$[+0.91, +2.02]$
OOD	OpenEvolve	83 / 2 / 44	$+0.79$	$[+0.22, +1.37]$

decades, respectively. The aggregate LSR-Synth improvements are thus reflected at the task level: SPECEVO wins a majority of problems against every baseline under both ID and OOD evaluation.

H PROMPT TEMPLATES

This appendix reports the prompt templates used by the three functional roles in SPECEVO.

H.1 SPECULATOR

The Speculator uses the lightweight model for high-throughput exploration: it creates initial variants, reviews archived parents, and applies several mutation and crossover operators. Operator diversity prevents repeated generic rewrites, while the shared output contract ensures that every accepted candidate has both executable code and an embeddable strategy description. The six variation operators share the template below, and Table 18 lists their operator-specific instructions together with the auxiliary Speculator prompts.

Speculator — Shared template for the six variation operators

You are an expert in optimization heuristics [operator-specific role; see Table 18].

`{Task Description}`

`{Function Signature}`

`{Parent Program(s) and Score(s): one parent, or two for crossover}`

`{Archive Inspirations: descriptions and scores only}`

`{Optional Lessons Learned from the Advisor}`

`{Operator-Specific Instruction: see Table 18}`

`{Output Format}`

Return exactly three sections in this order: `## Analysis`, `## Description`, and `## Code`. The analysis must follow the operator-specific structure below. The description must be one plain-prose paragraph of 2–4 sentences (at most 80 words) naming the algorithmic paradigm, its key data structures, and its distinguishing mechanism. The code section must contain one fenced `python` block with the complete, runnable, self-contained program. The required function signature must match exactly; include every import and do not use placeholders, ellipses, or incomplete branches.

Table 18: Operator-specific instructions inserted into the shared Speculator template, and the auxiliary Speculator prompts. Unless noted otherwise, outputs follow the three-section format (`## Analysis`, `## Description`, `## Code`); the required `## Analysis` parts are listed in italics.

Prompt	Role and instruction
Variation operators (sampled uniformly; shared template)	
General mutation	Improve a candidate algorithm: preserve the parent's useful structure but make one concrete algorithmic change likely to raise the score; no renaming, reformatting, comment edits, random-seed changes, or isolated constant retuning; treat inspirations as high-level ideas. *Components (3–5 main phases); Strengths; Weaknesses (most likely bottleneck, citing a code element); Plan (one sentence).*
Focused single-weakness repair	Make a surgical improvement to an already functional candidate: fix exactly one weakness while preserving the parent's algorithmic identity and every unrelated component; no unrelated improvements, constant retuning, seed-only changes, or unnecessary helpers. *Tightest constraint; Fix (minimum meaningful change); Preserved.*
Mechanism replacement	Replace one interchangeable local mechanism without changing the overall algorithm (e.g., uniform with stratified initialization, a fixed step with an adaptive step or line search, greedy with Metropolis acceptance); preserve the signature, I/O contract, and parent structure. *Mechanism identified; Old versus new; Risk (main failure mode and guard).*
Analysis-targeted mutation	Additional input: a cached analysis of the parent (parent-analysis prompt below). Implement exactly the one suggested change most likely to improve the score, without inventing another direction. *Chosen bottleneck; Implementation plan; What stays unchanged.*
Structural crossover	Two parents (A and B): build a coherent hybrid from the strongest compatible mechanism of each parent within a single data flow, objective, and constraint-handling logic, rather than pasting blocks together. *Component table (initialization, optimization core, constraint handling; each from A, B, or a hybrid); Compatibility note; Expected improvement.*
Component-level crossover	Base and donor parents: keep the base's control flow, data flow, termination logic, and I/O contract, and transplant exactly one clearly stronger donor component with only the necessary glue. *Donor component chosen; Glue work; What is not changed.*
Auxiliary Speculator prompts	
Initialization variant	Input: code and scores of one or two seeds. Write a complete, feasible, and meaningfully different variant rather than copying a seed. Output: `## Description` and `## Code` only.
Parent analysis	Review why a parent likely does not score higher, without writing code: an algorithm summary (1–2 sentences), the top three bottlenecks ranked by expected impact (each tied to a specific code element and labeled High, Medium, or Low), and one implementable change per bottleneck; under 250 words.
Navigator fan-out variant	Input: an accepted Navigator program and its score. Keep its core approach intact while modifying constants, thresholds, secondary heuristics, edge-case handling, or implementation details. Output: `## Description` and `## Code` only.
Missing-description recovery	Input: a program whose description is missing or too short. Write one paragraph (2–4 sentences, under 80 words) naming the paradigm, key data structures, and distinguishing heuristic, without headings or boilerplate.

H.2 NAVIGATOR

The Navigator uses the frontier model for the small number of decisions that require population-level reasoning. It first creates distinct seed families and later selects Synthesis, Surgical, or Reframe according to the stagnation signal.

Navigator — Prompt for population initialization

You are an expert in the domain of optimization heuristics helping to design algorithms that can effectively solve optimization problems.

`{Task Description}`

`{Function Signature}`

{Previously Accepted Seeds: complete programs and scores}

Design a solution using a fundamentally different algorithm from every existing seed. First analyze which paradigm each seed uses, what information it exploits, and what relevant information it ignores. Then design from first principles using a different modeling, decomposition, construction, or search strategy. Do not make a minor variation, tune parameters, reorder existing logic, or refactor the same core algorithm.

{Output Format}

Return `## Description` followed by `## Code`. The description is a 2–4 sentence paragraph (at most 80 words) stating the paradigm, key data structures, and distinguishing mechanism. The code must be complete, runnable, self-contained, and expose the exact required signature.

Navigator — Prompt for Synthesis

You are an expert research lead synthesizing complementary mechanisms from competitive solutions.

{Task Description}

{Function Signature}

{Archive State: evaluation count, occupied niches, and stagnation}

{Top Three Anchors: description, score, and complete program}

{Five Additional Inspirations: descriptions and scores only}

The search is only mildly stalled. Synthesize, rather than invent, one program that coherently combines 2–3 concrete mechanisms from different anchors and plausibly beats each anchor individually. After the line `MOVE: SYNTHESIS`, include exactly three analysis parts: **Component table** (initialization, optimization core, and constraint repair, with anchor provenance), **Coherence note** (shared data layout, units, and call order), and **Why this should beat all anchors** (one sentence per anchor).

{Output Format}

Return exactly three sections in this order: `## Analysis`, `## Description`, and `## Code`. The first line of the analysis must be `MOVE: SYNTHESIS`. The description must be one plain-prose paragraph of 2–4 sentences (at most 80 words) naming the paradigm, key data structures, and distinguishing mechanism. The code must be complete and runnable, reproduce the required signature exactly, define every helper, and include every import.

Navigator — Prompt for Surgical improvement

You are an expert research lead applying one precise structural improvement to the current champion.

{Task Description}

{Function Signature}

{Archive State: evaluation count, occupied niches, and stagnation}

{Current Champion: description, score, and complete program}

{Five Additional Inspirations: descriptions and scores only}

The search is moderately stalled, so improve the champion without changing its algorithmic family. After the line `MOVE: SURGICAL`, include exactly four analysis parts: **Tightest constraint** (cite the exact limiting mechanism), **Structural fix** (one local behavioral change), **Preservation list**, and **Expected delta** (benefit, main risk, and guard). Implement exactly one local structural change; prefer using existing score, feasibility, slack, loss, or quality signals over a blind heuristic, and do not add unrelated improvements.

{Output Format}

Return exactly three sections in this order: `## Analysis`, `## Description`, and `## Code`. The first line of the analysis must be `MOVE: SURGICAL`. The description must be one plain-prose paragraph of 2–4 sentences (at most 80 words) naming the paradigm, key data structures, and distinguishing mechanism. The code must be complete and runnable, reproduce the required signature exactly, define every helper, and include every import.

Navigator — Prompt for Reframe

You are an expert research lead proposing a new algorithmic family when the current population has plateaued.

{Task Description}

{Function Signature}

{Archive State: evaluation count, occupied niches, and stagnation}

{Top Two Anchors: description, score, and complete program}

{Five Additional Inspirations: descriptions and scores only}

The search is deeply stalled. Design a fundamentally different paradigm whose data structures, control flow, and termination rule do not appear in the anchors or inspirations. After the line `MOVE: REFRAME`, include exactly four analysis parts: **Paradigm name** (use the textbook name), **Why it fits the problem** (two problem-specific sentences), **Why the current anchors miss it** (the shared assumption being dropped), and **Risk** (the main implementation pitfall and guard). Forbidden moves are constant retuning, subroutine grafting, and variable renaming.

{Output Format}

Return exactly three sections in this order: `## Analysis`, `## Description`, and `## Code`. The first line of the analysis must be `MOVE: REFRAME`. The description must be one plain-prose paragraph of 2–4 sentences (at most 80 words) naming the paradigm, key data structures, and distinguishing mechanism. The code must be complete and runnable, reproduce the required signature exactly, define every helper, and include every import.

H.3 Advisor

The Advisor converts deterministic, niche-level search statistics and accumulated failures into short natural-language guidance.

Advisor — Prompt for population-level reflection

You are a lessons-learned advisor reviewing the search trajectory of an evolutionary code-optimization system. Write a short prescriptive note that a mutation model will read verbatim before some future attempts. Do not restate the task or generic constraints already enforced by the evaluator.

{Task Description}

{Function Signature}

{Current State: best score, evaluations, accept rate, and stagnation}

{Improving Niches: frontiers advanced in the current window}

{Saturated Niches: recent attempts, unchanged frontiers, and scores}

{Accumulated Failure Knowledge: recurrence count and example}

{Previous Advice}

Write exactly the following sections, in this order, using the literal headers and no other Markdown:
`WORKING:` In 1–2 sentences, name the concrete algorithmic structure or mechanism shared by improving niches and leaders. Describe the algorithm, not the search process; if nothing is clearly working, say so.
`SATURATED:` In 1–2 sentences, name the busy strategy families whose frontiers no longer improve and should be de-emphasized. If there is no clear saturation, write "none."
`AVOID:` Provide up to 12 concrete error-prevention instructions as a numbered checklist. Ground each instruction in the supplied failure evidence or still-applicable previous guidance.
Keep the complete answer under 500 words. Return advice only; do not output code.

I Qualitative Analysis

This appendix examines one SpecEvo search run on each of two tasks: Circle Packing Rectangle, a continuous geometric problem, and EPLB, a discrete resource-allocation problem for mixture-of-experts inference. The evaluation log of each run records every candidate program, the role and model that produced it, its execution status, and the evaluator message. For each task, we describe why candidates were rejected, which algorithmic mechanisms the population used, and the guidance the Advisor derived from the failures. Failure classes are listed from most to least common. We label how widely each mechanism appears in the population as *ubiquitous*, *common*, *occasional*, or *rare*; a program usually combines several mechanisms.

Table 19: Failure classes among rejected candidates on Circle Packing Rectangle, ordered from most to least common. Each rejected candidate belongs to exactly one class.

Class	What goes wrong
Feasibility	Two circles intersect after the radii are inflated or the layout is rescaled, but the program returns without rechecking pairwise feasibility
Output shape	The returned array does not contain 21 rows with three values each, usually because a helper or fallback path uses a different representation
Type	Lists and arrays are combined in arithmetic, or a coordinate is missing or infinite
Name	A variable or import is used before being defined, often within a nested helper or early return branch
Runtime	The program exhausts the evaluation time limit instead of stopping after convergence
Numeric	An array is used directly in a truth test, or a reduction is applied to an empty array
Indexing	An index exceeds the array bounds while circle pairs are enumerated
Syntax	Typographic Unicode symbols or malformed code prevent parsing

I.1 Circle Packing Rectangle

The task is to place 21 non-overlapping circles inside a rectangle of fixed perimeter so as to maximize the sum of their radii. A configuration that violates feasibility receives no score, even if the violation is only one part in a thousand.

Recurring errors. Table 19 lists the failure classes. The most common failure, by a wide margin, is a pair of overlapping circles. These are not near misses: the overlaps appear when a program inflates the radii or rescales a finished layout to fill the container and returns it without re-checking pairwise distances, so relaxing the tolerance would accept almost none of them. The second most common class is a wrong output shape. The evaluator expects an array with 21 rows, each holding a center and a radius, but helper functions, early returns, and especially fallback branches, which run only when the main path fails, often return a different shape. The remaining classes are ordinary programming faults; a few programs cannot be parsed because they contain typographic Unicode symbols, such as a minus sign or a less-than-or-equal sign copied from prose.

The same failure classes appear under all model-allocation strategies in Appendix C, and overlapping circles are the most common failure under every one of them. In those runs, the number of rejections per quarter of the run stays roughly constant, because no component carries information about earlier failures forward. In the SpecEvo run, the rejection rate drops substantially from the initialization phase, which has no Advisor guidance, to the evolutionary phase, in which Advisor notes are included in Speculator prompts.

The RelayEvolve logs also show how failures differ between the two model tiers. When both models receive comparable numbers of calls, their rejection rates and overlap rates are similar. Timeouts, however, come almost entirely from the strong model, whose programs are longer and rely on expensive global optimizers, whereas indexing and shape errors come mostly from the lightweight model. SpecEvo uses the lightweight model for high-volume exploration and reserves the strong model for sparse interventions.

Algorithmic mechanisms. Table 20 groups the mechanisms by role. Nearly every program starts from a structured layout, typically hexagonal rows with alternating offsets, and most then search over the aspect ratio of the rectangle, because its shape determines how many circles fit in each row. Refinement uses penalty descent on pairwise overlaps, simulated annealing, or damped pairwise separation. Repair rescales the layout to the perimeter constraint, grows the radii greedily up to the nearest-neighbor limit, and ends with a shrink pass that removes the remaining overlaps. A wall-clock guard that returns the best feasible configuration found so far is present in almost every program.

The initialization phase and the Speculators introduce individual mechanisms and refine them one at a time, so the record improves in small steps. In this run, the Navigator's record-setting interventions combined existing mechanisms instead of introducing new ones. The final best program initializes a hexagonal layout, relaxes it with damped pairwise separation, grows the radii toward their nearest-

Table 20: Algorithmic mechanisms observed on Circle Packing Rectangle, grouped by their role. A program typically combines mechanisms from every group.

Role	Mechanism	Prevalence and remarks
Construction	Structured lattice or patterned row initialization	Ubiquitous; hexagonal rows with an explicit pattern rather than random points
	Search over the container aspect ratio	Common; the objective couples container shape with row occupancy
Refinement	Simulated annealing with stochastic acceptance	Common; uses a cooling schedule over perturbed configurations
	Continuous penalty descent	Common; penalizes intersections smoothly rather than forbidding them
	Pairwise separation relaxation	Common; applies damped opposing displacements to overlapping pairs
	Enlargement with backtracking for each circle	Occasional; halves the step until an individual radius fits
Repair	Global rescaling to satisfy the perimeter constraint	Common; applied after every stage rather than only at the end
	Final feasibility shrink pass	Common; uses the smallest global factor that removes all intersections
	Greedy radius inflation to the nearest neighbor limit	Occasional; grows each radius to the largest value permitted by its neighbors
Engineering	Wall-clock guard with a best feasible fallback	Ubiquitous; the only defense against the runtime failure class
	Priority queue processing	Occasional; handles the largest radius or violation first
	Multiple restart portfolio	Occasional; retains the best of several independent attempts
	Neighbor queries using a spatial index	Rare; replaces pairwise distance calculations with a spatial tree or triangulation

neighbor limits, adds a backtracking enlargement pass for each circle, rescales to the perimeter constraint after every stage, and ends with a shrink pass that guarantees feasibility. Each of these components already existed in the population, but no program using only one of them reached the score of their combination.

Advisor feedback. The box below shows the guidance the Advisor had consolidated by the end of the run. Each item is an instruction for one failure class, not a diagnosis of a single program, and the items are ordered by the number of rejected candidates they address. The note is rewritten at every checkpoint instead of extended, which keeps it short enough to include verbatim, and it is added to only a fraction of Speculator prompts so that it does not push the whole population toward one defensive coding style.

Consolidated Advisor guidance: Circle Packing Rectangle

1. Ensure that no two circles overlap by keeping the distance between their centers at least equal to the sum of their radii minus the tolerance, as the validator requires.
2. Return exactly 21 circles as an array with 21 rows and three values per row, and preserve this shape through every operation.
3. Convert all coordinates and radii to double precision before arithmetic, and ensure that no value is missing or infinite.
4. Define and import every name before use, and call only documented public interfaces.
5. Keep the total runtime below the evaluation limit by bounding iteration counts and stopping early after convergence.
6. Confirm that an array is nonempty before using it in a truth test or a reduction.
7. Clip indices to the actual array size, especially when enumerating circle pairs.

Table 21: Failure classes among rejected candidates on EPLB, ordered from most to least common. Each rejected candidate belongs to exactly one class.

Class	What goes wrong
Indexing	A physical or logical index derived from capacity arithmetic exceeds a tensor bound
Name	A module or attribute is used without being imported or defined, most often the priority queue module
Type	A tensor has the wrong element type, or an integer computation exceeds its range
Feasibility	Replica counts do not sum to the budget for every layer, or the placement exhausts the available capacity
Runtime	The placement loop exhausts the evaluation time limit
Shape	A tensor dimension does not match the extent expected by the evaluator
Numeric	A nonscalar tensor is used directly in a truth test
Output shape	The function does not return the three required tensors in the required order
Syntax	A typographic Unicode symbol makes the program unparseable

8. Use only plain ASCII characters, replacing typographic minus and less than or equal symbols with their ASCII equivalents.
9. Clip radii to a small positive lower bound before any geometric computation to avoid division by zero and invalid square roots.

I.2 EPLB

The task is to place logical experts onto physical GPU slots for mixture-of-experts inference. Given load statistics for each layer, a program decides how many replicas each expert receives and which slot each replica occupies, subject to a fixed replica budget, a fixed hierarchy of groups, nodes, and GPUs, and a capacity limit for each GPU. The problem is discrete, and a program must keep three indices consistent as replica counts change: the logical expert, the replica, and the physical slot.

Recurring errors. Table 21 lists the failure classes. On Circle Packing Rectangle, most rejected programs violate a geometric constraint; on EPLB, indexing errors dominate: a slot or replica index computed from capacity arithmetic exceeds the bounds of the tensor it indexes, even when the allocation policy itself is sound. Two classes have a single main cause. Most name errors come from programs that schedule each replica on the least-loaded GPU with a priority queue, implement the queue operations correctly, and never import the priority-queue module; some of these programs also call heap routines of the tensor library that do not exist. Most type errors come from a deterministic hash initialization, which multiplies a large hexadecimal constant by a slot index to spread the replicas of an expert across devices without a random number generator. Unless the product is reduced modulo the number of GPUs before it is used as an index, it overflows the integer range and yields an index far outside the tensor. Nearly every overflow rejection in the run came from a program containing such a constant, and a single guidance item can remove this defect while keeping the mechanism.

The rejection rate decreases throughout the run, with the largest drop between the initialization phase and the evolutionary phase. The mechanical faults that are frequent early on, namely the missing import, the overflowing constant, and the invalid slot index, are among the failure signatures accumulated by the Advisor. They are largely absent from the later population, although the same algorithmic ideas remain in use.

Algorithmic mechanisms. Table 22 lists the mechanisms. The interface imposes a per-layer solution and the group, node, and GPU hierarchy, so programs differ in three decisions. To choose the number of replicas per expert, most programs repeatedly give the next replica to the expert whose load per replica it reduces most, which is the classical apportionment rule (some programs implement it explicitly as Huntington–Hill); a substantial minority instead binary-search a load threshold and give each expert enough replicas to fall below it, and the strongest programs use the threshold search to approach the budget and the marginal-gain rule to correct the remainder. To assign slots,

Table 22: Algorithmic mechanisms observed on EPLB, grouped by the design decision they resolve. A program typically combines mechanisms from every group.

Decision	Mechanism	Prevalence and remarks
Structure	Independent solution for each layer	Ubiquitous; imposed by the interface
	Explicit treatment of the group, node, and GPU hierarchy	Ubiquitous; implements placement as a nested traversal of the hierarchy
Replicas	Allocation by marginal gain or apportionment	Ubiquitous; assigns the next replica where it most reduces the load per replica
	Water-filling threshold found by bisection	Common; gives each expert enough replicas to fall below a searched load threshold
	Huntington–Hill apportionment by name	Occasional; implements the classical rule explicitly
Placement	Modular or rotated round-robin assignment	Common; disperses the replicas of each expert across devices
	Deterministic hash initialization of the starting slot	Common; provides dispersion without a random number generator but causes the overflow failures
	Priority queue scheduling by load	Occasional; places each replica on the least loaded GPU
	Local swap or exchange refinement	Rare; exchanges two placed replicas after the initial assignment
Feasibility	Capacity clamping with a repair pass	Ubiquitous; fills remaining slots after the main assignment
	Explicit assertion on the replica budget	Common; checks the contract rather than assuming it
	Vectorized tensor operations instead of Python loops	Common; the main defense against the runtime failure class

programs use a modular or rotated round-robin walk over the GPUs, which spreads the replicas of each expert across devices, or a priority queue that places each replica on the least-loaded GPU. To enforce feasibility, most programs clamp to capacity, fill the remaining slots in a repair pass, and assert that the replica counts sum to the budget; the Advisor reinforces these mechanisms.

As on Circle Packing Rectangle, the Navigator's SYNTHESIS and SURGICAL interventions produce strong, valid programs that reinforce the incumbent design. Its later REFRAME interventions, which introduce a strategy family absent from the population, score below the incumbent. This is expected: REFRAME is triggered by stagnation to increase diversity, and the archive keeps these programs as parents even when they do not improve the record. The final record was set by a lightweight structural crossover that combines threshold-based replica allocation with the marginal-gain correction and a placement that respects the hierarchical capacity constraints; its structure came from an earlier Navigator proposal that lightweight variants then refined.

Advisor feedback. The EPLB guidance below is more mechanical than the Circle Packing Rectangle guidance, because most rejected EPLB programs use sound allocation policies and fail on indexing, imports, or element types.

Consolidated Advisor guidance: EPLB

1. Clip every index to the valid range of the dimension it addresses, keeping physical and logical indices within bounds and consistent with the expected shapes.
2. Import all required modules, including the priority queue module, at the top of the program, and call only documented public interfaces rather than private tensor library routines.
3. Convert tensor values to the intended element type before use, and maintain that type consistently across operations.
4. Ensure that replica counts sum to the replica budget for every layer, and clamp them to both the budget and the available GPU capacity.
5. Bound the number of placement iterations so that the program remains within the evaluation time limit.

6. Give the mapping from logical to physical indices the required three dimensions, with the final extent equal to the largest replica count.
7. Pass tensor shapes as tuples of integers rather than as a single integer.
8. Reduce a tensor to a scalar before using it in a truth test.
9. Return exactly the three required tensors in the order specified by the function signature.
10. Use only plain ASCII characters because a typographic hyphen is not valid syntax.
11. Use floor rounding rather than ceiling rounding in integer division so that the result matches the expected element type.

In both tasks, a few instructions cover most of the observed failures, and the first instructions concern output and feasibility requirements, not the algorithm. A stronger model alone would not be expected to remove these failures: in the RelayEvolve runs, the strong and lightweight models fail at similar overall rates and differ mainly in which requirements they violate. The difference in SPECEVO is that this evidence is carried forward: the note is rebuilt at every checkpoint from all accumulated failure signatures, so a failure observed early in the run still affects later proposals.