ADAEVA: ACCELERATING LLM-DRIVEN ALGORITHM DESIGN WITH ADAPTIVE PARTIAL EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) are increasingly used for automated algorithm design. However the computational cost of evaluating the generated algorithms can be excessive. We consider the common LLM-driven automated algorithm design (LLM4AD) setting in which a candidate algorithm is evaluated by aggregating its performance over a shared set of training instances. This instance-wise structure raises a natural question: *must every candidate be evaluated on the entire instance set before deciding whether it remains competitive?* Taking inspiration from algorithm configuration, we introduce AdaEva, a drop-in **ada**ptive partial-**eva**luation framework that progressively evaluates candidates on larger subsets of the same instance pool and eliminates unpromising candidates as evidence accumulates. Importantly, AdaEva leaves the underlying LLM4AD procedure and per-instance evaluator unchanged and requires no prior knowledge about instance difficulty. We instantiate this idea using successive halving (AdaEva-S) and statistical racing (AdaEva-R), and evaluate both mechanisms across three representative LLM4AD frameworks, multiple LLM backbones, and optimization domains spanning combinatorial and continuous black-box optimization. Under matched evaluation budgets, AdaEva more reliably balances evaluation effort across candidates than fixed partial-evaluation strategies, yielding strong search efficiency and anytime performance together with improved held-out generalization across the evaluated settings. Source code and experiment results are provided in Anonymous repository.

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have generated substantial interest in LLM-driven automated algorithm design (LLM4AD) (Liu et al., 2026b), where LLMs are used to generate and iteratively refine complex heuristic algorithms for challenging optimization tasks (Romera-Paredes et al., 2024; Liu et al., 2024; Stein & Bäck, 2025; Ye et al., 2024). While most work focuses on improving generated algorithms or reducing LLM inference cost, another major source of computational overhead has received comparatively little attention: *candidate evaluation*.

In many LLM4AD applications, evaluating a generated algorithm requires executing it repeatedly over a collection of problem instances. In combinatorial optimization, for example, each candidate heuristic may need to solve tens or hundreds of training instances before its quality can be estimated reliably. Similar costs arise in continuous black-box optimization, where candidate algorithms are typically evaluated across multiple functions, dimensions, or random seeds. As the number of generated candidates increases, exhaustive evaluation can therefore consume a substantial fraction of the overall computational budget, limiting the number of algorithms that can be explored. This motivates the central question of our work: *Can we reduce candidate-evaluation cost in evolutionary LLM4AD without sacrificing the quality of the algorithms discovered?*

We study the common LLM4AD setting in which candidate quality is measured by aggregating performance over a set of training instances. This structure, arising naturally in heuristic and algorithm design for optimization problems, creates an opportunity for *adaptive evaluation*. Rather than evaluating every candidate immediately on the full instance set, we can first evaluate candidate algorithms on a small subset and allocate additional evaluations only to those that seem promising. Clearly inferior candidates can thus be discarded early, before consuming the full evaluation budget.

Building on this observation, we introduce **AdaEva**, an **Ada**ptive **Eva**luation framework for LLM-driven algorithm design. AdaEva acts as an evaluation layer on top of an existing LLM4AD search process (see Figure 1): candidate generation, prompting, evolutionary operators, and the underlying problem evaluator remain unchanged, while evaluation effort is allocated adaptively based on the evidence accumulated during the search. AdaEva progressively allocates additional evaluations to candidates that remain competitive while eliminating unpromising candidates early, so that adaptive resource allocation is done without modifying the underlying LLM4AD framework's search mechanism.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

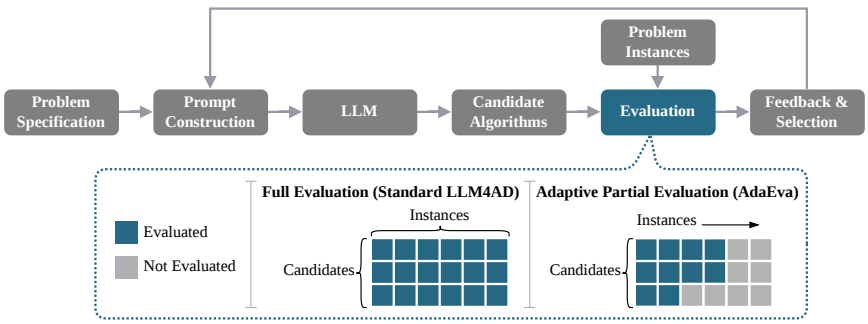

Figure 1: LLM-driven algorithm design as a loop. At each generation, a prompt built from the problem specification and selected parents is passed to an LLM, which generates candidate algorithms; these are evaluated on problem instances, and the best are selected and fed back to guide the next generation. Standard frameworks evaluate every candidate on every instance (left). AdaEva replaces only the evaluation stage with adaptive partial evaluation (right): candidates are evaluated progressively, those that fall behind are eliminated early, and the saved budget is concentrated on the candidates that remain competitive. Prompt construction, generation, and selection are unchanged.

This general design naturally accommodates resource-allocation techniques developed in the adjacent areas of automated algorithm configuration (Hoos, 2012) and hyper-parameter optimization (Yang & Shami, 2020). We instantiate AdaEva using two complementary approaches. The first, **AdaEva-S**, is based on *successive halving* (Karnin et al., 2013), which progressively allocates larger evaluation budgets to smaller sets of promising candidates. The second, **AdaEva-R**, builds on statistical *racing* (Birattari et al., 2002), which eliminates candidates once sufficient evidence indicates that they are unlikely to outperform their competitors (López-Ibáñez et al., 2016). Both approaches are particularly well suited to LLM4AD because they do not require predictive models of candidate performance and because they operate directly on instance-level evaluation outcomes.

We empirically demonstrate that AdaEva can substantially improve evaluation efficiency while preserving the effectiveness of the underlying LLM4AD search. Across combinatorial optimization and continuous black-box optimization, we assess search progress throughout the evaluation budget, the quality of the final returned algorithms, and their performance on held-out instances. We further examine how adaptive allocation reshapes evaluation consumption and candidate survival, characterize the resulting evaluation dynamics, and assess the robustness of AdaEva across different LLM backbones, allocation strategies, statistical tests, and racing configurations. Together, these experiments evaluate not only whether adaptive evaluation improves search efficiency, but also how its allocation decisions influence the behavior and outcomes of evolutionary LLM4AD.

Our main contributions can be summarized as follows:

- We formulate instance-wise candidate evaluation in evolutionary LLM4AD as an adaptive resource-allocation problem and introduce **AdaEva**, a drop-in evaluation framework that adaptively allocates evaluation effort while preserving the underlying candidate-generation and evolutionary search mechanisms.
- We instantiate AdaEva with two complementary allocation mechanisms: **AdaEva-S**, based on successive halving, and **AdaEva-R**, based on statistical racing.
- We show that adaptive evaluation provides more consistent search efficiency and held-out performance under matched evaluation budgets than any single fixed number of evaluations per candidate.

2 Related Work

LLM-Driven Algorithm Design. LLM4AD frameworks use LLMs to generate and revise executable algorithms, with evaluation results guiding subsequent search (Liu et al., 2026b). EoH (Liu et al., 2024) evolves a population of heuristics through LLM-generated variation and recombination, while ReEvo (Ye et al., 2024) uses reflective feedback and memory to guide later generations. LLaMEA (Stein & Bäck, 2025) applies LLM-guided evolutionary search to the design of black-box optimization algorithms. Subsequent work has explored other ways to guide and organize search: HSEvo (Dat et al., 2025) uses harmony search to balance exploration and exploitation, and MCTS-AHD (Zheng et al., 2025) structures exploration as a search tree. More recently, HiFo-Prompt (Zhong et al., 2026) combines foresight prompts that respond to population dynamics with hindsight prompts that distill reusable insights from earlier

generations. Work on generalization has also expanded to more diverse training instances. EoH-S searches for a portfolio of complementary heuristics (Liu et al., 2026a) on a large set of instances with different distributions and sizes, and MoH (Shi et al., 2026) develops heuristic optimizers through self-invocation and multi-instance training. These advances change how candidates are generated, selected, or retained, but they do not address the question of how to efficiently compare generated candidate algorithms.

Evaluation Efficiency in LLM4AD. Candidate evaluation is often a major computational bottleneck in LLM-driven search, particularly when each candidate must be executed across many problem instances, stochastic runs, or expensive simulations. This imbalance can be substantial when generation is cheap relative to evaluation: OrderPlace (Mo et al., 2026), for example, reports that most optimization time is spent evaluating generated strategies, while GEPA (Agrawal et al., 2026) similarly highlights subsampled validation as a promising direction for improved efficiency. Existing methods reduce this cost through pre-specified evaluation cascades (AlphaEvolve (Novikov et al., 2025)), smaller evaluation subsets (HeuriGym (Chen et al., 2026)), learned fitness predictors or proxy evaluators (Hercules-P (Wu et al., 2025); Janus (Liu et al., 2026c)), or more sample-efficient search policies (ShinkaEvolve (Lange et al., 2026); AdaEvolve (Cemri et al., 2026)). In contrast, AdaEva leaves both candidate generation and the per-instance evaluator unchanged, and adaptively determines how much of a shared instance pool each candidate should be evaluated on before allocating further evaluation effort or making a selection decision.

Adaptive Partial Evaluation in Adjacent Domains. Adaptive allocation of evaluation effort has a long history in automated algorithm configuration and hyper-parameter optimization. Racing methods such as F-Race (Birattari et al., 2002) and iterated racing (López-Ibáñez et al., 2016) incrementally evaluate candidates on shared instances and eliminate those with sufficient evidence of inferior performance, while successive halving and Hyperband (Jamieson & Talwalkar, 2016; Li et al., 2018) progressively allocate larger budgets to smaller sets of promising configurations. Extensions such as BOHB (Falkner et al., 2018) and DEHB (Awad et al., 2021) combine such resource allocation with model-based and evolutionary search, and more recent work explicitly treats evaluation as a sequential budget-allocation problem under ranking uncertainty (Zouhar et al., 2026). AdaEva builds on these principles but uses them as *instance-wise adaptive partial evaluation* mechanisms within existing evolutionary LLM4AD frameworks, leaving their candidate-generation and population-update logic unchanged and thereby decoupling evaluation allocation from the underlying search strategy.

3 Instance-wise Adaptive Partial Evaluation for LLM4AD

3.1 Evolutionary Loop in LLM4AD

For a given problem $\mathcal{P}$ and a (typically not further specified) space $\mathcal{H}$ of executable algorithms, LLM-driven algorithm design casts the search as an evolutionary loop in which an LLM acts as the variation operators (Figure 1). At generation t, a population $\Pi^{(t)} \subseteq \mathcal{H}$ of size M supplies parents for a generator $\mathcal{L}$. These parents, together with the problem specification, are used to construct prompts from which the generator produces an offspring set $\mathcal{O}^{(t)} \subseteq \mathcal{H}$.

The current population and offspring are then combined into a candidate pool $\mathcal{U}^{(t)} = \Pi^{(t)} \cup \mathcal{O}^{(t)}$, and the M candidates with the best empirical performance are retained as the elites and copied to the next population. For this comparison, the performance of an algorithm $h \in \mathcal{H}$ is evaluated as the average quality $\hat{g}(h) = \sum_{x \in \mathcal{S}} f(h; x)/|\mathcal{S}|$ that h achieves on a subset $\mathcal{S} \subseteq \mathcal{I}$ of relevant problem instances $\mathcal{I}$, with $f(h; x)$ being the (average, in case of randomized algorithms) solution quality obtained by this heuristic on instance $x \in \mathcal{S}$. In standard LLM4AD frameworks, the evaluation is conducted on the whole instance set ($\mathcal{S} \equiv \mathcal{I}$). Note that at each generation, only the offprings need to be evaluated, as their parents' performance were obtained from the previous generation.

3.2 Adaptive Evaluation-Budget Allocation

Evaluating every candidate on the full pool spends the budget uniformly regardless of how candidates perform. Clearly inferior candidates consume as much as competitive ones, while closely matched candidates may need more evidence than either receives. Our proposed AdaEva framework acts as an evaluation layer that instead allocates budget progressively and leaves candidate generation, parent selection, and population update of the underlying LLM4AD framework untouched. We adapt two commonly-used techniques from the adjacent areas of hyper-parameter optimization and automated algorithm configuration. **AdaEva-S** follows *successive halving* (Karnin et al., 2013; Li et al., 2018) and removes a fixed fraction of candidates after a predetermined number of evaluations, whereas **AdaEva-R** follows iterated racing (Birattari et al., 2002; López-Ibáñez et al., 2016) and removes candidates using statistical tests whenever the accumulated evidence establishes that they are inferior.

Algorithm 1 AdaEva for one generation. AdaEva-S and AdaEva-R differ only in Lines 3 and 5.

Require: Candidate set $\mathcal{U}^{(t)}$, evaluation history $\mathcal{E}$, population size M
Ensure: Survivor set $\mathcal{V}^{(t)}$, incumbent $\hat{h}^{(t)}$, updated history $\mathcal{E}$
1: $\mathcal{V}^{(t)} \leftarrow \mathcal{U}^{(t)}$
2: **while** $|\mathcal{V}^{(t)}| > M$ **do**
3: Evaluate candidates in $\mathcal{V}^{(t)}$ on additional instances, reusing evaluations stored in $\mathcal{E}$
4: Update $\mathcal{E}$ with the new evaluation results
5: Identify inferior candidates $\mathcal{X} \subseteq \mathcal{V}^{(t)}$ according to the elimination rule
6: $\mathcal{X} \leftarrow$ at most $|\mathcal{V}^{(t)}| - M$ candidates from $\mathcal{X}$
7: $\mathcal{V}^{(t)} \leftarrow \mathcal{V}^{(t)} \setminus \mathcal{X}$
8: **end while**
9: $\hat{h}^{(t)} \leftarrow$ best performing candidate in $\mathcal{V}^{(t)}$
10: **return** $\mathcal{V}^{(t)}, \hat{h}^{(t)}, \mathcal{E}$

Algorithm 1 summarizes the common procedure that is used by both AdaEva variants at each generation t of their evolutionary loop. As described in Section 3.1, starting from the combined candidate set $\mathcal{U}^{(t)}$ of parents and offsprings, the goal is to select M candidates to form the next population $\mathcal{U}^{(t+1)}$. AdaEva performs this selection iteratively. At each round, the current survivor set $\mathcal{V}^{(t)}$ is evaluated on additional problem instances, while reusing all previously collected results stored in the evaluation history $\mathcal{E}$ (line 3). Based on the accumulated evidence, a subset of inferior candidates is then eliminated (line 5), and the procedure repeats on the remaining candidates. Elimination is capped to retain at least M candidates (lines 6–7). When M remain, they form the next population and the best survivor is returned as the incumbent. AdaEva-S and AdaEva-R share this overall structure and differ only in how additional evaluations are allocated and how candidates are eliminated.

Partial evaluation also affects how performance feedback is presented to the LLM. In our main experiments, we retain each framework's native prompting, without indicating that feedback may be based on a subset of instances, thereby isolating the evaluation layer. A partial-evaluation-aware variant shows no consistent advantage (Sections D and E).

AdaEva-S as a scheduled elimination strategy. We define the *evaluation depth* of a candidate as the number of problem instances on which it has been evaluated. AdaEva-S follows the resource-allocation principle of successive halving (Karnin et al., 2013): candidates are evaluated at progressively larger depths, while a fixed fraction of inferior candidates is eliminated at each round. AdaEva-S has two hyper-parameters: the initial evaluation depth $\underline{n}_0$ and the reduction factor $\eta > 1$. At each evaluation round, candidates are ranked according to their accumulated evaluation results and approximately a $1/\eta$ fraction is retained, subject to keeping at least M candidates. Starting from $\mathcal{U}^{(t)}$, the number of rounds is $J = \lceil \log_\eta(|\mathcal{U}^{(t)}|/M) \rceil$. The evaluation depth increases geometrically across these rounds, from $\underline{n}_0$ instances in the first round to the full instance set $\mathcal{I}$ in the final round. Following the geometrically increasing resource levels commonly used in Hyperband and successive-halving-based multi-fidelity optimization (Li et al., 2018; Falkner et al., 2018), we define the cumulative evaluation depth at round j, $j = 1, \ldots, J$ as $q_j = \underline{n}_0 \left(|\mathcal{I}|/\underline{n}_0\right)^{\frac{j-1}{J-1}}$. This way, early rounds use inexpensive partial evaluations to eliminate weak candidates, while progressively more evaluation effort is allocated to those that survive.

AdaEva-R as an evidence-driven elimination strategy. AdaEva-R follows the principle of iterated racing (López-Ibáñez et al., 2016), where elimination is driven by accumulated statistical evidence rather than a pre-specified schedule. At each round, every surviving candidate is evaluated on an additional Δ instances. To account for stochasticity in the generated algorithms, the race may continue after all instances in $\mathcal{I}$ have been evaluated by collecting additional independent evaluations on previously seen instances. Thus, unlike AdaEva-S, reaching evaluation depth $|\mathcal{I}|$ does not necessarily terminate the race. Elimination begins only after candidates reach a minimum evaluation depth $\underline{n}_0$, after which a statistical decision rule is applied at every round. If the observed performance differences are not statistically distinguishable from instance-to-instance and stochastic variation, all candidates remain in the race; otherwise, candidates with sufficient evidence of inferior performance are removed. Consequently, both the evaluation depth at which elimination occurs and the number of candidates eliminated in each round are adaptive and may vary across generations. Accordingly, AdaEva-R also modifies the stopping criterion in Algorithm 1: besides terminating when M candidates remain, the race stops when no candidate has been eliminated for ℓ consecutive rounds. Alternatively, as in original iterated-racing López-Ibáñez et al. (2016), the surviving set can be carried forward without being refilled to size M; we study this choice in Section 4.3. By default, we follow the racing procedure of Birattari et al. (2002), using the rank-based Friedman test to detect overall performance differences, followed by Conover's post-hoc test to identify inferior candidates at confidence level α. We ablate alternative statistical procedures, including a paired t-test and the Nemenyi post-hoc test in Section 4.3 and Section E.

Table 1: Benchmark and search settings including the search and held-out instance sets, population size M, offspring generated per generation N, number of generations T, and evaluation budget B.

Framework	Problem	Search Pool	Held-out Set	M	N	T	B
EoH	OBP	5,000 items, capacity 100	1k/5k/10k items, capacities 100/500	20	20	20	10,000
	TSP	100-city, uniform coordinates	29 TSPLIB instances	10	10	10	6,400
	FSSP	50 jobs, 2–20 machines	Taillard benchmark	10	10	10	6,400
HiFo-Prompt	OBP	5,000 items, capacity 100	1k/5k/10k items, capacities 100/500	10	50	10	12,500
	TSP	100-city, uniform coordinates	29 TSPLIB instances	10	50	5	16,000
	FSSP	50 jobs, 2–20 machines	Taillard benchmark	10	50	5	16,000
LLaMEA	BBOB	24 func. $\times$ 3 instances, 5 dims	24 func. $\times$ 5 instances, 5/10/20 dims	10	10	20	14,400

4 EXPERIMENTS

LLM4AD Frameworks. We evaluate AdaEva as an off-the-shelf evaluation layer in three representative LLM4AD frameworks: Evolution of Heuristics (EoH) (Liu et al., 2024), HiFo-Prompt (Zhong et al., 2026), and LLaMEA (Stein & Bäck, 2025). They differ in how candidates are generated and selected, with EoH applying multiple variation strategies and selecting once per generation, HiFo-Prompt applying its operators sequentially with selection after each, and LLaMEA evolving parent to offspring with a parent's measured performance written into the prompt. They also differ in what is evolved, such as a problem-specific decision rule embedded in a fixed solver for EoH and HiFo-Prompt, and a complete executable optimization algorithm for LLaMEA. Adaptive evaluation is therefore tested across both component-level heuristic design and full-algorithm synthesis. We retain each framework's native generation, variation, population-management, and prompting mechanisms, changing only the evaluation stage. Framework descriptions and their instantiations are in Section A.

Benchmark Settings. We examine AdaEva across two broad optimization domains, combinatorial optimization (CO) and continuous black-box optimization (BBO). The CO suite comprises Online Bin Packing (OBP) (Seiden, 2002), where items arriving sequentially must be assigned to bins; the Traveling Salesperson Problem (TSP) (Matai et al., 2010), where the goal is a short tour visiting all cities; and Flow Shop Scheduling (FSSP) (Emmons & Vairaktarakis, 2012), where jobs must be scheduled across multiple machines. The component evolved in each is the bin-scoring function for OBP and the landscape-update heuristic within Guided Local Search (GLS) (Voudouris, 1997) for TSP and FSSP, whereas LLaMEA evolves complete metaheuristic procedures tested on BBOB suite (Hansen et al., 2009).

Generalization remains a central challenge in machine learning and is equally important in LLM-driven algorithm design. We therefore evaluate incumbents recorded at five checkpoints corresponding to 20%, 40%, 60%, 80%, and 100% of the total search budget on held-out instances that are disjoint from the search pools and introduce controlled changes in problem scale, instance source, or dimensionality. For OBP, the held-out instances increase problem scale and vary capacity, using 1k, 5k, and 10k items with $C \in \{100, 500\}$, which are originally tested in (Romera-Paredes et al., 2024; Liu et al., 2024). For TSP, the search uses synthetic 100-city instances with coordinates sampled from $[0, 1]^2$, whereas held-out evaluation uses TSPLIB (Reinelt, 1991) instances, introducing a shift in instance source and distribution. For FSSP, the search uses synthetic instances with uniformly sampled processing times, while held-out evaluation uses the external Taillard benchmark (Taillard, 1993), again changing the instance distribution. For BBOB, the search uses three instances of each function in $d = 5$, while held-out evaluation uses disjoint instances in $d \in \{5, 10, 20\}$; the higher-dimensional settings additionally require the generated algorithm to transfer to larger search spaces and different optimization landscapes. Brief descriptions of the benchmarks are provided in Table 1, with full details given in Section B.

LLM Backbones. The principal experiments use Devstral (Rastogi et al., 2025) as the primary backbone, selected for its open availability and specialization in coding and software-engineering instances. We additionally use the instruction-tuned Qwen2.5-Coder (Hui et al., 2024), another openly available coding model with strong code-generation, reasoning, and code-fixing capabilities, to test whether the observed effects persist across independently developed LLM generators. This focus on open coding models, however, leaves open whether the findings extend to proprietary frontier models. We therefore evaluate selected settings with Gemini-3.1-Flash-Lite (Team et al., 2023), a proprietary model that supports instruction following and configurable internal thinking, providing a complementary test beyond open-weight coding backbones. All models use a maximum generation length of $2{,}048$ tokens.

Baselines. We compare AdaEva-S and AdaEva-R with Fixed-K evaluation, where each candidate is evaluated on $K \in \{1, |\mathcal{I}|/4, |\mathcal{I}|/2, |\mathcal{I}|\}$ instances. This spans evaluation depths from a single instance to the full pool and tests

Table 2: Held-out generalization performance of each evaluation strategy across LLM4AD frameworks, benchmarks, and LLM backbones. Mean ranks (lower is better) are computed across held-out instance group, then averaged across checkpoints. **Bold** and underlined denote the best and second-best methods in each row, respectively. Detailed per-checkpoint and third LLM backbone (Gemini) results are provided in Section C.

Framework	Benchmark	LLM	Fixed-1	Fixed-25%	Fixed-50%	Full	AdaEva-S	AdaEva-R
EoH	FSSP	Devstral	4.967	3.167	3.767	3.933	3.133	**2.033**
		Qwen	4.767	4.033	2.667	4.267	2.867	**2.400**
	OBP	Devstral	2.717	**1.817**	3.067	5.083	3.667	4.650
		Qwen	**1.983**	4.300	3.567	3.867	3.400	3.883
	TSP	Devstral	4.333	2.967	3.133	6.000	2.900	**1.667**
		Qwen	3.933	2.767	2.933	5.833	3.367	**2.167**
HiFo-Prompt	FSSP	Devstral	4.500	3.100	3.133	5.700	2.433	**2.133**
		Qwen	5.167	**2.333**	3.700	4.100	2.600	3.100
	OBP	Devstral	3.250	2.783	4.183	5.100	2.933	**2.750**
		Qwen	3.667	2.933	3.767	4.400	**2.883**	3.350
	TSP	Devstral	5.000	3.067	2.933	6.000	**1.900**	2.100
		Qwen	6.000	3.067	2.733	4.300	2.600	**2.300**
LLaMEA	BBOB	Devstral	5.867	3.833	4.067	3.200	**1.933**	2.100
		Qwen	6.000	3.967	4.400	3.167	2.067	**1.400**

whether one fixed K is robust across framework–problem settings. Within each benchmark, all strategies use the same evaluation budget, instance pool, initial population, and LLM4AD configuration, repeated with five random seeds.

Experimental Configuration. AdaEva-R uses the parameters recommended for IRACE (López-Ibáñez et al., 2016): the first test is applied once the evaluation depth reaches $\underline{n}_0 = 5$ instances, one instance is added per round ($\Delta = 1$), elimination is decided using Friedman test followed by Conover post-hoc test at a significant level $\alpha = 0.05$, and a race ends after $\ell = 2$ rounds without removal or once M candidates remain. AdaEva-S is configured to match AdaEva-R's setting where comparable, with an initial depth $\underline{n}_0 = 5$ and a reduction factor $\eta = 1.25$. Sensitivity to these parameters is examined in Section 4.3.

4.1 Generalization to Unseen Instances

Table 2 compares the generalization performance of different allocation strategies across all 14 combinations of framework, benchmark, and LLM backbone. For each combination, the incumbents recorded at five budget checkpoints are evaluated on held-out instances drawn from a different distribution, the strategies are ranked within each checkpoint and held-out group, and the ranks are then averaged.

The racing variant of AdaEva attains the best mean rank in 8 (out of 14) combinations and ranks among the best two in 13, except for the EoH/OBP setting. No fixed evaluation depth achieves this advantage. Full-pool evaluation, the default in these frameworks, is the worst or second-worst strategy in every combination, except for the LLaMEA/BBOB setting. Meanwhile, shallower fixed evaluation depths tend to perform better, though not consistently. Fixed-1 is the best strategy on EoH/OBP with the Qwen2.5-Coder backbone (mean rank of 1.983) but is almost the worst on the other combinations, while neither Fixed-25% nor Fixed-50% ranks better than third in more than half of the combinations. Which fixed depth performs well therefore depends on the framework, the benchmark, and the backbone, none of which is known before the search is run, whereas both AdaEva instantiations use a single configuration throughout.

Figure 2 shows that both AdaEva instantiations achieve the strongest held-out ranks across checkpoints. Aggregated over backbones, they rank best at nearly every checkpoint in all three frameworks; on LLaMEA, they lead by more than a full rank from the earliest checkpoint onward. Fixed depths remain stable but lower, while full-pool evaluation improves little as budget accumulates, consistent with completing the fewest generations.

Finding 1. On unseen instances, AdaEva ranks best or near-best across every framework, benchmark, and LLM backbone tested, whereas no fixed evaluation depth is competitive across them.

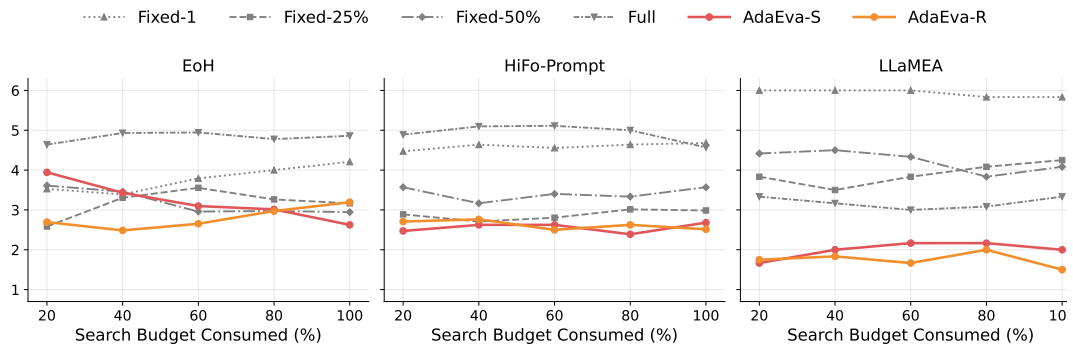

Figure 2: Held-out generalization trajectories of evaluation strategies across three frameworks. Mean ranks (lower is better) are computed within each held-out instance group, then averaged across the corresponding benchmarks, and LLM backbones. Each panel shows performance at increasing fractions of the search budget (20%–100%).

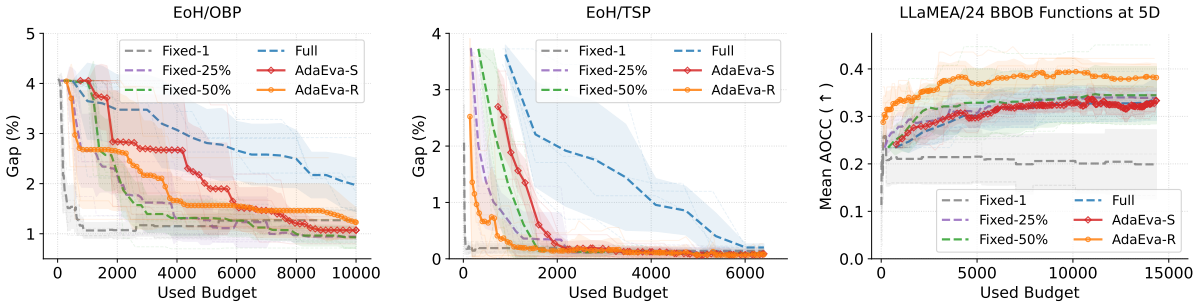

Figure 3: Search performance under increasing evaluation budget with the Devstral backbone, on 3 settings chosen to span the range of behaviours observed. Incumbents are selected during search from partial evaluations and then re-evaluated on the complete pool for reporting; the y-axis gives that value, as optimality gap for OBP, gap for TSP, and mean AOCC for BBOB; lower is better in the first two panels and higher is better in the third. Curves show the mean over 5 runs, with shaded regions indicating variability across seeds. The remaining 4 settings, and results with the Qwen2.5-Coder and Gemini-3.1 backbones are available in Section C.

4.2 Search Progress under Fixed Budget and Adaptive Allocation

The held-out results establish which strategies return better algorithms, but not why. We now examine the search itself, reporting the full-pool objective value of the recorded incumbent against budget consumed. A fixed evaluation budget can be spent on many candidates evaluated on few instances each, or on few candidates evaluated on many. We refer to the number of candidates a strategy examines as its *breadth* and the number of instances each receives as its *depth*; their product is fixed by the budget, so a strategy cannot increase one without conceding the other. Figure 3 shows the validation curves on three settings chosen to span the behaviours observed, with the remaining four in Section C.

The validation curves show that full-pool evaluation makes the least progress per unit budget in all settings. This is especially clear for EoH/TSP, where it uses nearly the entire budget to reach an objective value attained by other strategies within the first third (Figure 3-middle). Its estimates are the most precise, but this precision shortens the search horizon: each generation costs 640 evaluations, so few fit within $B = 6{,}400$. BBOB (Figure 3-right) is an exception, where full-pool evaluation is mid-ranked because heterogeneous instances make shallow evaluation more prone to misranking candidates.

Reducing the depth helps, but which reduction helps depends on the setting. Fixed-1 is fastest on EoH/OBP and EoH/TSP but by far the weakest on LLaMEA/BBOB, where it reaches a mean AOCC of only 0.20 versus more than 0.33 for every other strategy; full-pool evaluation reverses this ordering. The benchmark properties in Table 3 help explain this variation. Kendall's W (Kendall & Smith, 1939) measures agreement among instance-wise candidate rankings, while the Across- and Within-candidate coefficients of variations (McKay, 1932) capture complementary performance variability. On OBP, rankings are identical across instances ($W = 1.00$), making shallow evaluation sufficient; TSP is more subtle, with very low ranking agreement ($W = 0.03$) but negligible Across CV (0.0001) and low Within CV (0.0348), consistent with Fixed-1 remaining competitive. BBOB combines weaker ranking agreement

Table 3: Benchmark properties and the evaluation depth reached by AdaEva-R. Kendall's W and the Across- and Within-candidate coefficients of variation (CVs) are computed from a fixed initial set of candidates evaluated on every instance. Evaluation depth is reported under EoH for the CO benchmarks and under LLaMEA for BBOB. Definitions of these metrics are provided in Section C.1.

	Property			AdaEva-R Depth	
Benchmark	**Kendall's W**	**Across CV**	**Within CV**	**Mean Depth**	**Stop at First Block**
OBP	1.0000	0.3403	0.0016	7.04 ± 0.72	$92.8\% \pm 3.5\%$
TSP	0.0312	0.0001	0.0348	12.49 ± 1.98	$73.8\% \pm 5.6\%$
FSSP	0.6133	0.0088	0.1216	19.25 ± 1.02	$46.1\% \pm 4.4\%$
BBOB	0.6567	0.6809	1.1235	22.79 ± 2.94	$61.9\% \pm 4.4\%$

($W = 0.66$) with substantially larger Across and Within CVs (0.68 and 1.12), consistent with the poor performance of very shallow evaluation. Thus, no single benchmark statistic determines the useful fixed depth, motivating adaptive evaluation. Further definitions and benchmark-specific analysis of these metrics are provided in Section C.1.

Both AdaEva instantiations remain close to the best fixed depth in every panel, and on BBOB AdaEva-R is the strongest strategy overall. They are not uniformly best, and the exceptions are the ones Table 3 predicts. On OBP, where a single instance suffices, Fixed-1 reaches a lower gap than either instantiation; on BBOB, AdaEva-S remains with the mid-ranked group while AdaEva-R leads. What the adaptive strategies avoid is the large loss that each fixed depth incurs in at least one setting, which is what the held-out ranks of Section 4.1 reflect.

Unlike a fixed depth, which commits to a point on the breadth–depth trade-off before the search begins, AdaEva arrives at one during it. Figure 4 counts the candidates each strategy evaluates, normalized by the number full-pool evaluation affords in the same setting. The fixed strategies sit at ratios determined by their depth alone, $|\mathcal{I}|/K$, identical in every setting and invariant across runs, whereas both AdaEva instantiations fall between the extremes at 2.5 to 10.1 times the candidates of full-pool evaluation, differing across benchmarks and between the two instantiations. The depth they reach responds to the benchmark in the direction W implies, as the right columns of Table 3 show that on OBP, where the instances agree, AdaEva-R removes 92.8% of candidates after the first evaluation block at a mean depth of 7.04 instances, whereas on FSSP and BBOB fewer than two-thirds stop there and the mean depth reaches 19 and 23. A fixed depth cannot produce this profile, since it must either pay the depth for every candidate or forgo it for all.

Finding 2. Full-pool evaluation buys precision at the cost of a shorter search horizon, and the best fixed depth depends on how consistently a benchmark's instances agree about candidates. Adaptive allocation finds a suitable depth on its own, evaluating many candidates shallowly and few deeply.

4.3 Sensitivity to the AdaEva Configuration

The main experiments use a single default AdaEva configuration across all settings. We assess sensitivity by varying one parameter or design choice at a time, holding the rest fixed, using Devstral on EoH with three CO benchmarks and on LLaMEA/BBOB. Sensitivity is the percent change in incumbent quality relative to the default, computed within matched seeds by taking the median across seeds at each of five budget checkpoints and then averaging across checkpoints; positive values indicate worse performance. Table 4 reports representative configurations, with the full sweep and measurement details in Section E.

Most parameter changes have small effects on the three CO benchmarks, although their impact can be substantially larger on BBOB. In particular, EoH/FSSP is relatively insensitive to the tested configurations, with most alternatives improving on or closely matching the default, whereas several BBOB configurations change incumbent quality by more than 20%. The AdaEva-S parameters also have different effects across settings: changing $\underline{n}_0$ or η has noticeable but modest effects on EoH, while $\eta = 1.5$ improves BBOB by 7.74% and $\underline{n}_0 = 10$ improves it by 2.01%. These results suggest that the fixed configuration is generally robust on the EoH benchmarks, while BBOB is more sensitive to the choice of evaluation parameters. Two changes degrade quality consistently or nearly so. Disabling stochastic evaluation is worse in all four settings, suggesting that drawing fresh seeds once the instance pool is exhausted provides a useful source of additional evaluation evidence. Replacing the Friedman test with the paired t-test is worse in three of four settings, with a particularly large degradation on BBOB. Refilling the surviving set with previously eliminated candidates has little effect on the EoH benchmarks but produces a substantial degradation on BBOB when disabled, indicating that its practical importance can depend on the interaction between the race and the host framework. Finally, modifying the LLaMEA prompt to account explicitly for partial evaluation yields a 14.99% degradation on BBOB.

Table 4: Selected sensitivity results. Each value is the percent change in incumbent quality relative to the default of the same instantiation, computed within matched seeds, taking the median across seeds at each of five budget checkpoints and then averaging across checkpoints. Positive values indicate worse performance than the default. The complete sweep is reported in Section E.

Strategy	Configuration	EoH/OBP	EoH/TSP	EoH/FSSP	LLaMEA/BBOB
AdaEva-S	$n_0 = 10$	+1.41	−0.17	−0.02	−2.01
	$\eta = 1.5$	+0.25	+0.26	−0.02	−7.74
AdaEva-R	$\mathcal{T}_{\text{stats}} = t$-test	+2.39	+0.04	−0.06	+14.93
	Refill = Off	−0.39	+0.03	−0.01	+19.48
	Stochastic = Off	+0.41	+0.09	+0.03	+21.16
	Prompt = Partial Evaluation	–	–	–	+14.99

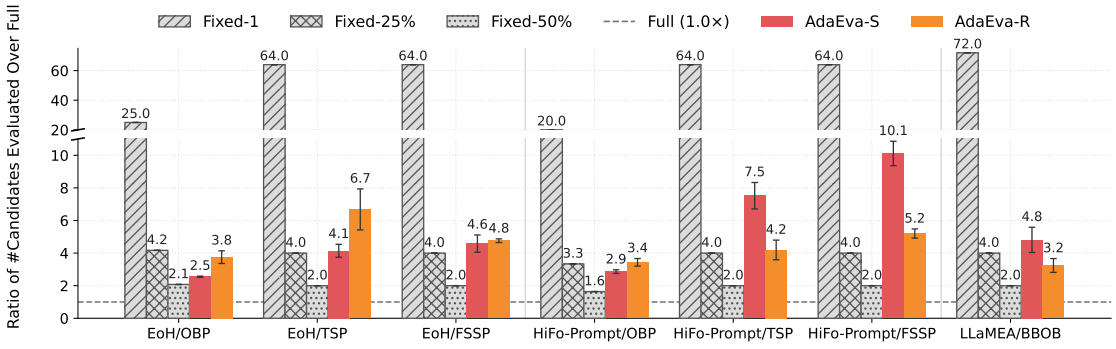

Figure 4: Relative number of candidate algorithms evaluated during the search, under different evaluation strategies across EoH, HiFo-Prompt, and LLaMEA (using the Devstral backbone). The values are normalized by the corresponding full-evaluation baseline, so larger ratios indicate that more candidate algorithms can be sampled within the same evaluation budget; error bars denote variability across five runs.

5 Conclusion

We introduced **AdaEva**, an adaptive evaluation layer for LLM-driven algorithm design that allocates evaluation effort based on evidence accumulated during search. We study two complementary instantiations: **AdaEva-S**, based on successive halving (Karnin et al., 2013; Li et al., 2017), and **AdaEva-R**, based on statistical racing (Birattari et al., 2002; López-Ibáñez et al., 2016). Both integrate into existing LLM4AD frameworks without modifying candidate generation, parent selection, or population update. Across three LLM4AD frameworks, multiple LLM backbones, and combinatorial and continuous black-box optimization problems, no single fixed amount of evaluation per candidate performs consistently well across settings. AdaEva instead allocates more evaluation to candidates that remain competitive, balancing search breadth against evaluation effort and yielding more consistent search and held-out performance than any single fixed evaluation strategy.

Our experiments adopt a fixed-budget perspective, asking how well a given evaluation budget is used. An equally natural alternative is fixed-target optimization, which minimizes the evaluation effort required to reach a specified performance level, as common in black-box optimization benchmarking Hansen et al. (2022). Our anytime results suggest AdaEva may also be effective in this setting, motivating future evaluation of AdaEva under fixed-target criteria.

Several questions remain open. First, the relative performance of AdaEva-S and AdaEva-R varies across settings, and our experiments do not identify a reliable criterion for selecting between them in advance. Developing mechanisms that adapt not only the amount of evaluation but also the allocation strategy itself is a promising direction for future work. Second, although the default configurations are relatively stable on the combinatorial benchmarks, some design choices have a larger effect on BBOB, suggesting that broader studies of parameter adaptation may further improve performance. Finally, our experiments isolate candidate evaluation by preserving the native generation, prompting, and population-management mechanisms of each framework and by matching methods according to candidate-evaluation budget. Consequently, we do not separate framework and benchmark effects in a full factorial design, nor do we account for the additional LLM inference cost incurred when cheaper evaluation enables more generations. Extending adaptive resource allocation jointly across evaluation and other components of LLM4AD remains as future work.

AI USE STATEMENT

In this paper, we employed AI to edit the manuscript and produce portions of the code. All AI-assisted outputs were thoroughly reviewed by us. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Noor Awad, Neeratyoy Mallik, and Frank Hutter. Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization. *arXiv preprint arXiv:2105.09821*, 2021.

Mauro Birattari, Thomas Stützle, Luis Paquete, Klaus Varrentrapp, et al. A racing algorithm for configuring metaheuristics. In *Gecco*, volume 2, 2002.

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.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, et al. Heurigym: An agentic benchmark for llm-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pp. 27466–27520, 2026.

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.

Hamilton Emmons and George Vairaktarakis. *Flow shop scheduling: theoretical results, algorithms, and applications*, volume 182. Springer Science & Business Media, 2012.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018.

Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Technical Report RR-6829, INRIA, 2009. URL `https://hal.inria.fr/inria-00362633/document`.

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, and Tea Tusar. Anytime performance assessment in blackbox optimization benchmarking. *IEEE Trans. Evol. Comput.*, 26(6):1293–1305, 2022. doi: 10.1109/TEVC.2022.3210897. URL `https://doi.org/10.1109/TEVC.2022.3210897`.

Holger H Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pp. 37–71. Springer, 2012.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pp. 240–248. PMLR, 2016.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pp. 1238–1246. PMLR, 2013.

Maurice G Kendall and B Babington Smith. The problem of m rankings. *The annals of mathematical statistics*, 10 (3):275–287, 1939.

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.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations*, 2017.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of machine learning research*, 18(185):1–52, 2018.

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

Fei Liu, Yilu Liu, Qingfu Zhang, Tong Xialiang, and Mingxuan Yuan. Eoh-s: Evolution of heuristic set using llms for automated heuristic design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37090–37098, 2026a.

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, 2026b.

Ximeng Liu, Qianlong Wang, Yingming Mao, Annan Li, Yatao Li, Shizhen Zhao, Jianmin Wu, Dawei Yin, and Dou Shen. Janus: An algorithm-evaluator co-evolution framework for llm-driven discovery under expensive evaluation budgets. *arXiv preprint arXiv:2608.08189*, 2026c.

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25, 2010.

AT McKay. Distribution of the coefficient of variation and the extended" t" distribution. *Journal of the Royal Statistical Society*, 95(4):695–698, 1932.

Shibing Mo, Jing Liu, Jianchu Xu, and Ruilin Wu. Order matters: Unveiling the hidden impact of macro placement sequences via proxy-guided LLM evolution. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=fPjkFDW9j7.

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.

Abhinav Rastogi, Adam Yang, Albert Q Jiang, Alexander H Liu, Alexandre Sablayrolles, Amélie Héliou, Amélie Martin, Anmol Agarwal, Andy Ehrenberg, Andy Lo, et al. Devstral: Fine-tuning language models for coding agent applications. *arXiv preprint arXiv:2509.25193*, 2025.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

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.

Steven S Seiden. On the online bin packing problem. *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

Yiding Shi, Jianan Zhou, Wen Song, Jieyi Bi, Yaoxin Wu, Zhiguang Cao, and Jie Zhang. Generalizable heuristic generation through llms with meta-optimization. In *International Conference on Learning Representations*, volume 2026, pp. 119314–119349, 2026.

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, 2025. doi: 10.1109/TEVC.2024.3497793.

Eric Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

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

Christos Voudouris. *Guided local search for combinatorial optimisation problems.* PhD thesis, University of Essex, 1997.

Xuan Wu, Di Wang, Chunguo Wu, Lijie Wen, Chunyan Miao, Yubin Xiao, and You Zhou. Efficient heuristics generation for solving combinatorial optimization problems using large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 3228–3239, 2025.

Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

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.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in llm-based automatic heuristic design. *arXiv preprint arXiv:2501.08603*, 2025.

Mengyuan Zhong, Jialong Shi, Jianyong Sun, and Ye Fan. Hifo-prompt: Prompting with hindsight and foresight for llm-based automatic heuristic design. In *International Conference on Learning Representations*, volume 2026, pp. 117102–117143, 2026.

Vilém Zouhar, Julia Kreutzer, Alon Lavie, Tom Kocmi, Matt Post, Ondřej Bojar, and Mrinmaya Sachan. Dynamically allocating evaluation effort for model ranking. *arXiv preprint arXiv:2608.03437*, 2026.

SUPPLEMENTARY MATERIAL FOR "ADAEVA: ACCELERATING LLM-DRIVEN ALGORITHM DESIGN WITH ADAPTIVE PARTIAL EVALUATION"

CONTENTS

A LLM4AD Frameworks **13**

B Benchmark Settings **15**

C More Results **16**

C.1 Benchmark Heterogeneity 16

C.2 Search Progress Under Fixed Budget 17

C.3 Generalization to Unseen Instances 17

D Partial Evaluation and Framework Feedback **32**

E Sensitivity to the AdaEva Configuration **33**

A LLM4AD FRAMEWORKS

This appendix describes how the LLM4AD frameworks used in our experiments instantiate their evolutionary loop. At a high level, each generation consists of parent selection and prompt construction, offspring generation, evaluation, and population update. Parents are drawn from the current population and combined with the problem specification to construct prompts; the generator LLM $\mathcal{L}$ returns offsprings $\mathcal{O}^{(t)}$, which are screened for compilation and interface conformance. Newly generated candidates are evaluated while evaluation results for retained candidates are carried over, and population management determines the candidates retained for subsequent generation. The frameworks differ in how prompts are constructed and how generation and population management are organized; these framework-specific details are described below. AdaEva replaces the evaluation stage while leaving the remaining framework components unchanged.

Evolution of Heuristics (EoH). EoH (Liu et al., 2024) represents each candidate by a natural-language description of its high-level idea, referred to as a *thought*, together with its executable code implementation. The initial population is generated from the problem specification, with one LLM call producing one heuristic. At each subsequent generation, parents are selected from the current population using rank-based sampling, with better-ranked candidates receiving higher selection probability. Five prompt strategies then generate new heuristics: E1 promotes exploration by asking the LLM to produce an algorithm that is substantially different from the selected parents, while E2 asks it to identify the common idea in multiple parents and develop a new heuristic based on that idea. M1 modifies a single parent structurally, M2 changes its parameter settings, and M3 simplifies potentially redundant components. Each strategy is invoked repeatedly, with each LLM call producing one offspring; thus, with population size M, the five strategies generate up to $5M$ offspring per generation. These offspring are evaluated and added to the current pool when feasible, after which the best M heuristics are retained for the next generation.

Hindsight-Foresight Prompt (HiFo-Prompt). HiFo-Prompt (Zhong et al., 2026) retains EoH's thought-and-code representation and its five foundational evolutionary strategies, while augmenting each prompt with two additional sources of guidance. The *hindsight* component maintains an Insight Pool of design principles distilled from high-performing heuristics; retrieved insights are injected into subsequent prompts, and their utility is updated according to the performance of the offspring they helped generate. The *foresight* component, implemented by the Evolutionary Navigator, monitors recent population progress, stagnation, and description-level diversity, and selects one of three regimes of explore, exploit, or balance, whose corresponding design directive is added to the prompt. Unlike EoH's generation-level application of its five operators, HiFo-Prompt applies its evolutionary operators sequentially. For each operator, the LLM is invoked repeatedly to generate an offspring batch of size M, where each offspring is produced by an independent LLM call. These offspring are merged with the current population and immediately subjected to elitist

Algorithm 2 One generation of EoH. The evaluation step (Line 12) is the stage AdaEva replaces.

Require: Population $\Pi^{(t)}$ of size M, prompt strategies $\mathcal{S} = \{\mathrm{E1}, \mathrm{E2}, \mathrm{M1}, \mathrm{M2}, \mathrm{M3}\}$, instance pool $\mathcal{I}$
Ensure: Population $\Pi^{(t+1)}$
1: $\mathcal{O}^{(t)} \leftarrow \emptyset$
2: **for all** $s \in \mathcal{S}$ **in parallel do**
3: **for** $j \leftarrow 1$ **to** M **do**
4: $\Pi_p \leftarrow \textsc{SelectParents}(\Pi^{(t)})$
5: $h \leftarrow \mathcal{L}(\textsc{Prompt}(s, \Pi_p, \mathcal{P}))$
6: **if** h compiles and satisfies the interface **then**
7: $\mathcal{O}^{(t)} \leftarrow \mathcal{O}^{(t)} \cup \{h\}$
8: **end if**
9: **end for**
10: **end for**
11: $\mathcal{U}^{(t)} \leftarrow \Pi^{(t)} \cup \mathcal{O}^{(t)}$
12: Evaluate the newly generated candidates in $\mathcal{O}^{(t)}$ on $\mathcal{I}$, retaining prior evaluations for $\Pi^{(t)}$
13: $\Pi^{(t+1)} \leftarrow \mathrm{Best}_M\big(\mathcal{U}^{(t)}; \hat{g}\big)$
14: **return** $\Pi^{(t+1)}$

Algorithm 3 One generation of HiFo-Prompt. Selection is applied after each operator rather than once per generation. AdaEva replaces the evaluation step (Line 15); the quantities marked † consume the resulting costs and are therefore affected by partial evaluation, since candidates surviving longer are evaluated on more instances. The generation prompt itself carries no fitness values.

Require: Population $\Pi^{(t)}$ of size M, operators $\mathcal{S} = \{\mathrm{E1}, \mathrm{E2}, \mathrm{M1}, \mathrm{M2}, \mathrm{M3}\}$ with weights $\{w_s\}$, insight pool $\mathcal{K}$, instance pool $\mathcal{I}$
Ensure: Population $\Pi^{(t+1)}$
1: $\Pi \leftarrow \Pi^{(t)}$
2: **for all** $s \in \mathcal{S}$ **do**
3: **if** $u \sim \mathrm{Uniform}(0, 1) < w_s$ **then**
4: $\mathcal{O} \leftarrow \emptyset$
5: **for** $j \leftarrow 1$ **to** M **do**
6: $\Pi_p \leftarrow \textsc{SelectParents}(\Pi; \hat{g})$ ▷ † rank-based on cost
7: $\kappa \leftarrow \textsc{RetrieveInsights}(\mathcal{K})$ ▷ † utilities scored by cost
8: $\theta \leftarrow \textsc{Regime}(\Pi; \hat{g})$ ▷ † stagnation from best cost
9: $h \leftarrow \mathcal{L}(\textsc{Prompt}(s, \Pi_p, \mathcal{P}, \kappa, \theta))$
10: **if** h compiles and satisfies the interface **then**
11: $\mathcal{O} \leftarrow \mathcal{O} \cup \{h\}$
12: **end if**
13: **end for**
14: $\mathcal{U} \leftarrow \Pi \cup \mathcal{O}$
15: Evaluate the newly generated candidates in $\mathcal{O}$ on $\mathcal{I}$, retaining prior evaluations for Π
16: $\Pi \leftarrow \mathrm{Best}_M(\mathcal{U}; \hat{g})$
17: $\mathcal{K} \leftarrow \textsc{UpdateInsights}(\mathcal{K}, \mathcal{O}, \hat{g})$ ▷ † credit from cost
18: **end if**
19: **end for**
20: **return** $\Pi^{(t+1)} \leftarrow \Pi$

population management, reducing the population back to size M before the next operator is applied. Consequently, survivors of an earlier operator can serve as parents for subsequent operators within the same generation, while each operator operates on a population of size M rather than on the accumulated candidates from preceding operators.

Large Language Model Evolutionary Algorithm (LLaMEA). LLaMEA (Stein & Bäck, 2025) generates complete optimization algorithms as executable code and evolves them through an evolution-strategy-like loop. In its original formulation, LLaMEA uses a single-solution population with $\mu = \lambda = 1$: one parent algorithm is used to generate one offspring algorithm at each iteration. The offspring is produced by prompting the LLM with the selected parent, its performance information, and a condensed history of previously generated algorithms and scores, and the LLM is instructed either to refine or redesign the algorithm. Compilation or runtime errors are incorporated into the feedback

Algorithm 4 One generation of LLaMEA. The original formulation uses $\mu = \lambda = 1$; we use its elitist $(M+N)$ mechanism in a population-based extension with $M, N > 1$. AdaEva replaces the evaluation step (Line 14); the line marked † writes a parent's measured cost directly into the prompt, so this framework exposes partial estimates to the generator.

Require: Population $\Pi^{(t)}$ of size M, offspring count N, selection mode $\varsigma \in \{(M+N), (M, N)\}$, instance pool $\mathcal{I}$, incumbent $\hat{h}$
Ensure: Population $\Pi^{(t+1)}$, updated incumbent $\hat{h}$

1: $\mathcal{O}^{(t)} \leftarrow \emptyset$
2: **for** $j \leftarrow 1$ **to** N **do**
3: $h_p \leftarrow \textsc{SelectParent}(\Pi^{(t)})$
4: $h \leftarrow \mathcal{L}(\textsc{Prompt}(h_p, \hat{g}(h_p), \varepsilon_p, \mathcal{P}))$ ▷ † cost in prompt; ε_p: errors
5: **if** h compiles and satisfies the interface **then**
6: $\mathcal{O}^{(t)} \leftarrow \mathcal{O}^{(t)} \cup \{h\}$
7: **end if**
8: **end for**
9: **if** $\varsigma = (M+N)$ **then**
10: $\mathcal{U}^{(t)} \leftarrow \Pi^{(t)} \cup \mathcal{O}^{(t)}$ ▷ elitist: incumbents compete
11: **else**
12: $\mathcal{U}^{(t)} \leftarrow \mathcal{O}^{(t)}$ ▷ non-elitist: incumbents discarded
13: **end if**
14: Evaluate the newly generated candidates in $\mathcal{O}^{(t)}$ on $\mathcal{I}$, retaining prior evaluations for $\Pi^{(t)}$
15: $\Pi^{(t+1)} \leftarrow \mathrm{Best}_M\big(\mathcal{U}^{(t)}; \hat{g}\big)$
16: $\hat{h} \leftarrow \mathrm{Best}_1\Big(\{\hat{h}\} \cup \Pi^{(t+1)}; \hat{g}\Big)$
17: **return** $\Pi^{(t+1)}$, $\hat{h}$

provided to the LLM. Two selection variants are considered: an elitist $(\mu+\lambda)$ strategy and a non-elitist (μ, λ) strategy, both with $\mu = \lambda = 1$. In the elitist variant, the best-so-far algorithm is retained as the parent and replaced only when the offspring is at least as good; in the non-elitist variant, the newly generated offspring becomes the parent for the next iteration regardless of its performance. Although the original formulation is single-solution based, the same parent–offspring evolution and selection mechanism naturally extends to population-based evolution by allowing $\mu > 1$ and $\lambda > 1$.

B Benchmark Settings

Combinatorial Optimization. EoH and HiFo-Prompt are evaluated on Online Bin Packing (OBP), the Traveling Salesperson Problem (TSP), and Flow Shop Scheduling (FSSP), following the benchmark formulations used by EoH. For OBP, the original EoH search uses five Weibull instances with $5{,}000$ items and capacity $C = 100$, with fitness computed from the ratio between the lower bound and the number of bins used. We enlarge this search pool to 25 independently generated instances while retaining the same distribution and problem size. For TSP, the original search pool contains 64 randomly generated 100-city instances with coordinates sampled from $[0, 1]^2$, and fitness is the average gap to the Concorde optimum. For FSSP, the original search pool contains 64 randomly generated instances with 50 jobs and 2–20 machines, with processing times sampled uniformly from $[0, 1]$; fitness is the average makespan. Candidate execution for TSP and FSSP uses the original GLS-based implementations, with at most $1{,}000$ local-search iterations and a 60-second limit per candidate–instance evaluation.

For generalization, we use held-out evaluation sets that are disjoint from the search instances. For OBP, the original EoH study evaluates five independently generated instances for each combination of problem size 1k, 5k, and 10k and capacity $C \in \{100, 500\}$, reporting percentage gap to the lower bound. We use these size and capacity settings for held-out evaluation. For TSP, EoH evaluates additional random instances at 20, 50, and 100 cities and 29 TSPLIB instances; the latter are evaluated by gap to the TSPLIB best-known solutions, and we use TSPLIB for our held-out evaluation. For FSSP, EoH evaluates Taillard instance sets with 10 instances per job–machine configuration and reports relative gap to the Taillard reference values; we use Taillard instances for held-out evaluation.

Continuous Black-Box Optimization. LLaMEA is evaluated on the BBOB suite (Hansen et al., 2009). The search pool contains three instances (IDs in {1,2,3}) of each of the 24 noiseless BBOB functions in five dimensions, giving

72 function instances in total. BBOB comprises five function groups: separable, low- or moderately-conditioned, highly-conditioned unimodal, multimodal with adequate global structure, and multimodal with weak global structure. Candidate performance is aggregated using the normalized Area Over the Convergence Curve (AOCC).

For generalization, the original LLaMEA study evaluates the generated algorithms on five BBOB instances per function in dimensions $d \in \{5, 10, 20\}$. We follow the same dimensions and number of held-out instances per function, but use explicitly disjoint instance identifiers (IDs in {4,5,6,7,8}) for our evaluation. Each held-out instance is evaluated 3 times in our study using independent evaluation seeds.

C More Results

This section reports results omitted from the main text for space. Section C.1 analyzes benchmark heterogeneity using Kendall's W and the Across- and Within-candidate coefficients of variation (CVs), providing additional context for the benchmark-dependent evaluation-depth results discussed in the main text. Section C.2 gives search-progress curves for the settings and LLM backbones not shown in Figure 3. Section C.3 reports held-out performance at every budget checkpoint, first as mean ranks per framework and then as objective values for each benchmark and instance group.

C.1 Benchmark Heterogeneity

Metrics. To characterize how consistently benchmark instances distinguish between candidate algorithms, we compute Kendall's coefficient of concordance W (Kendall & Smith, 1939) on a fixed set of candidates evaluated on every instance. Each instance ranks the candidates by objective value, and W measures agreement among these rankings, with $W = 1$ indicating identical rankings across instances and $W = 0$ indicating no concordance. Candidates that fail on any instance are excluded so that all instances rank the same candidate set.

We additionally report two coefficients of variation (CVs) (McKay, 1932). Across CV is the coefficient of variation of candidates' mean performance over the instance pool, measuring how differentiated the candidates are on average. Within CV is the mean candidate-level coefficient of variation across instances, measuring how much an individual candidate's performance varies across the pool. These statistics are computed from the same complete candidate–instance matrix and are independent of the evaluation strategy.

Discussion. The four benchmarks exhibit substantially different combinations of ranking agreement and performance variability, as summarized in Table 3. OBP has perfect ranking concordance ($W = 1.00$), indicating that the relative ordering of candidates is invariant across the evaluated instances. Its Across CV is nevertheless non-negligible (0.34), reflecting meaningful differences in candidates' mean performance. This combination is consistent with the strong performance of Fixed-1, although candidates differ substantially in quality, their relative ordering can be recovered from a single instance.

TSP provides a contrasting case. Its ranking concordance is extremely low ($W = 0.03$), but its Across CV is nearly zero (0.0001) and its Within CV is also small (0.0348). Thus, instances can disagree strongly in their fine-grained candidate rankings even though candidate mean performance is tightly concentrated and individual candidates exhibit relatively little variation across instances. This helps explain why the low W does not translate into a clear need for deeper evaluation, and why Fixed-1 remains competitive on TSP.

FSSP lies between these cases, with moderate ranking concordance ($W = 0.61$) and a larger Within CV (0.12). Its instances therefore exhibit more variation in candidate performance than TSP, making additional evaluation more informative. This is reflected in the greater benefit of intermediate or deeper fixed evaluation depths.

BBOB combines moderate ranking concordance ($W = 0.66$) with substantially larger Across and Within CVs (0.68 and 1.12). Candidate performance therefore varies considerably both across candidates and across instances. In this setting, a single-instance estimate provides substantially less information about overall candidate quality, consistent with the poor performance of Fixed-1 and the advantage of deeper or adaptive evaluation.

Overall, these results show that no single benchmark statistic determines the appropriate evaluation depth. Kendall's W captures the consistency of candidate rankings across instances, whereas the CVs capture the magnitude of performance variability. Their combination provides a more informative characterization of when shallow evaluation is likely to be reliable, but does not prescribe a fixed depth across benchmarks.

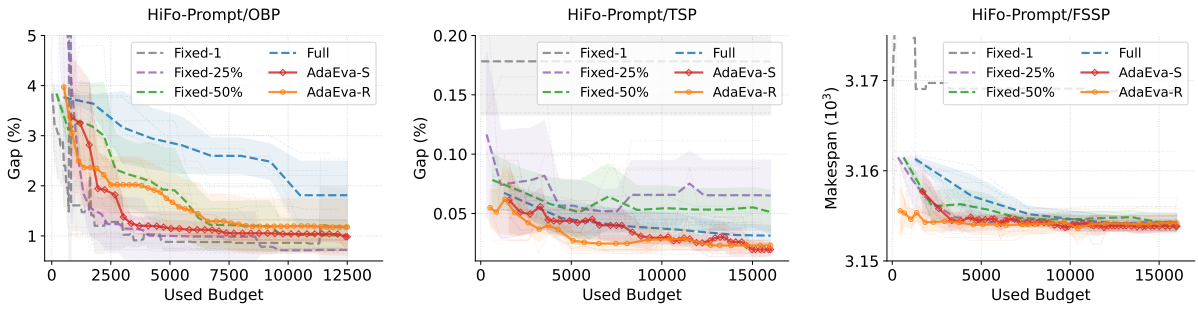

Figure 5: Search performance under increasing evaluation budget with the Devstral backbone for HiFo-Prompt. Each panel compares AdaEva-S and AdaEva-R with Fixed-K evaluation at $K = 1$, 25%, and 50% of the pool, and with full-pool evaluation ($K = |\mathcal{I}|$). Incumbents are selected during search from partial evaluations and re-evaluated on the complete pool for reporting. The y-axis shows this full-pool performance, measured as optimality gap for OBP and TSP and makespan for FSSP, with lower values better. Curves show mean performance across five independent search seeds, with shaded regions indicating variability across seeds.

C.2 Search Progress Under Fixed Budget

Figure 3 in the main text shows three of the seven framework–benchmark settings with the Devstral backbone. Figure 5 completes that backbone with the three HiFo-Prompt settings, while Figure 6 repeats the EoH and HiFo-Prompt comparisons with Qwen2.5-Coder. For the additional LLM backbones, Figure 7 shows LLaMEA/BBOB with Qwen2.5-Coder and Gemini-3.1-Flash-Lite, together with EoH/TSP using Gemini-3.1-Flash-Lite.

The additional search-progress curves largely preserve the qualitative pattern observed in the main text. Across the CO settings, full-pool evaluation generally progresses most slowly, while the adaptive methods reach competitive full-pool quality earlier. The Qwen2.5-Coder-based curves do not show a systematic reversal of this behavior relative to Devstral's results, although the absolute performance and the relative separation between strategies vary by benchmark. On LLaMEA/BBOB, both Qwen2.5-Coder and Gemini-3.1 likewise show sustained gains from adaptive evaluation as the budget increases.

C.3 Generalization to Unseen Instances

C.3.1 Mean Ranks per Checkpoint

Table 2 in the main text macro-averages held-out ranks over the five budget checkpoints. Tables 5 to 7 report these ranks at each checkpoint separately, one table per framework, together with the overall mean rank and the proportions of comparison cells in which each strategy is ranked first or among the top two.

The checkpoint breakdown reveals how the relative performance of the adaptive methods evolves during search. Under EoH (Table 5), AdaEva-R has the best mean rank at the first three checkpoints and the best overall rank of 2.800. AdaEva-S improves steadily from 3.944 at 20% of the budget to the best rank of 2.625 at 100%, overtaking AdaEva-R at the final checkpoint. This difference is consistent with their respective evaluation mechanisms: AdaEva-S follows a predetermined schedule and reaches the full evaluation depth only at its final stage, whereas AdaEva-R allocates further evaluation according to the evidence accumulated during the race.

Under HiFo-Prompt (Table 6), no such crossover occurs. AdaEva-S and AdaEva-R remain the top two strategies at every checkpoint, with overall ranks of 2.558 and 2.622, respectively. Their ranks also remain close throughout the search, indicating that both evaluation mechanisms provide similar held-out performance in this framework.

The LLaMEA results (Table 7) show the clearest separation. Both AdaEva instantiations remain below a mean rank of 2.3 at every checkpoint, with AdaEva-R achieving the best overall rank of 1.850 and AdaEva-S 2.017. Together, they account for the best-ranked strategy in 96.7% of comparison cells. In contrast, Fixed-1 is the worst strategy at every checkpoint, while full-pool population-based LLaMEA remains mid-table with an overall rank of 3.467. The table also reports the original single-solution $(1 + 1)$-LLaMEA formulation as an additional reference to the original LLaMEA search setting; the main comparison instead uses the population-based extension described in Section A.

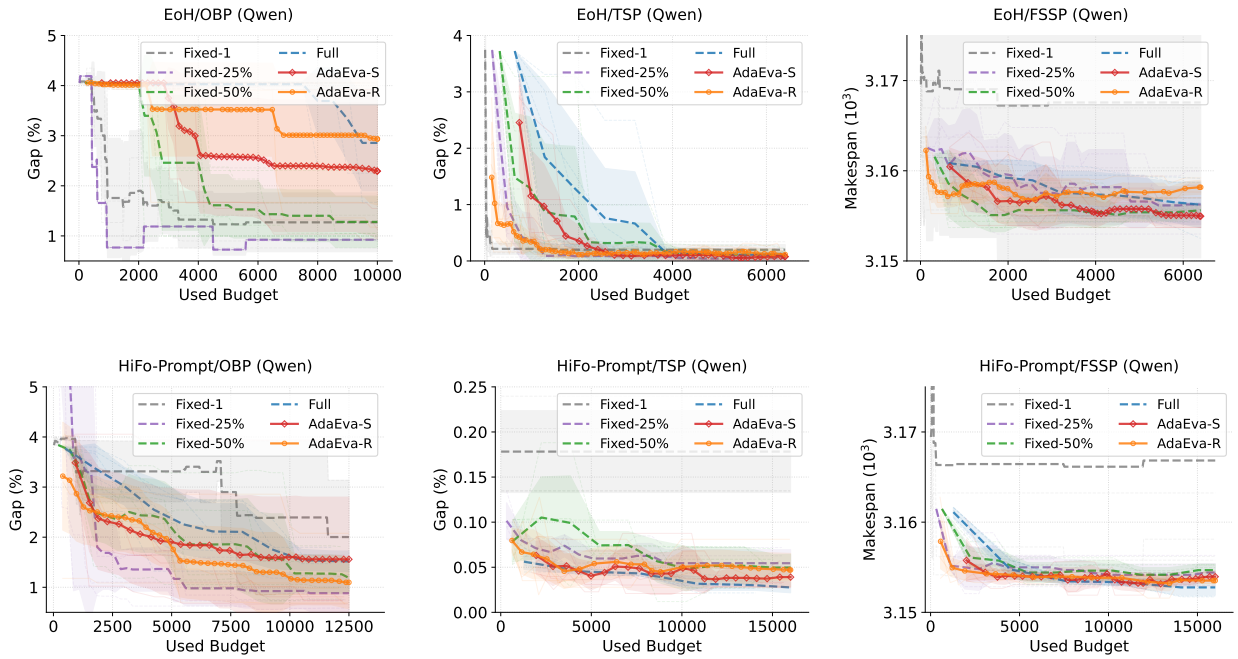

Figure 6: Search performance under increasing evaluation budget with the Qwen2.5-Coder backbone for EoH (top) and HiFo-Prompt (bottom). Each panel compares AdaEva-S and AdaEva-R with Fixed-K evaluation at $K = 1$, 25%, and 50% of the pool, and with full-pool evaluation ($K = |\mathcal{I}|$). Incumbents are selected during search from partial evaluations and re-evaluated on the complete pool for reporting. The y-axis shows this full-pool performance, measured as optimality gap for OBP and TSP and makespan for FSSP, with lower values better. Curves show mean performance across five independent search seeds, with shaded regions indicating variability across seeds.

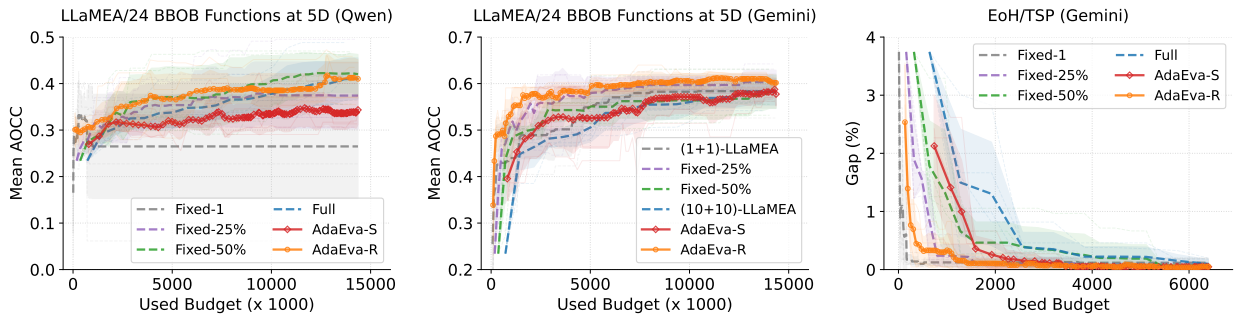

Figure 7: Search performance under increasing evaluation budget with the Qwen2.5-Coder and Gemini-3.1-Flash-Lite backbones. The LLaMEA/BBOB panels and the EoH/TSP panel compare AdaEva-S and AdaEva-R with Fixed-K evaluation at $K = 25\%$ and 50% of the pool, and with full-pool evaluation ($K = |\mathcal{I}|$). Incumbents are selected during search from partial evaluations and re-evaluated on the complete pool for reporting. The y-axis shows mean AOCC for BBOB and optimality gap for TSP; higher AOCC and lower gap indicate better performance. Curves show mean performance across five independent search seeds, with shaded regions indicating variability across seeds.

Across EoH and HiFo-Prompt frameworks, full-pool evaluation is consistently among the weakest strategies at intermediate checkpoints, whereas in LLaMEA it performs closer to the middle of the comparison. The preferred fixed evaluation depth also differs across frameworks: Fixed-25% is the strongest fixed strategy overall under HiFo-Prompt, while Fixed-25% and Fixed-50% alternate as the strongest fixed strategies across the later EoH checkpoints. These differences reinforce that a single fixed evaluation depth does not transfer uniformly across search frameworks.

Table 5: EoH anytime held-out performance, reported as mean rank (lower is better) at each search-budget checkpoint. Results are macro-averaged over the three CO benchmarks and two LLM backbones (Devstral and Qwen2.5-Coder). Bold and underline denote the best and runner-up strategies, respectively, in each column; higher values are better for the Best % and Top-2 % columns.

	Search Budget Consumed					Overall		
Strategy	20%	40%	60%	80%	100%	Rank ↓	Best % ↑	Top-2 % ↑
Fixed-1	3.528	3.389	3.792	4.000	4.208	3.783	<u>15.0</u>	25.0
Fixed-25%	**2.583**	<u>3.306</u>	3.556	3.264	3.167	<u>3.175</u>	11.7	<u>35.0</u>
Fixed-50%	3.611	3.458	<u>2.958</u>	**2.972**	<u>2.944</u>	3.189	10.6	27.2
Full	4.639	4.931	4.944	4.778	4.861	4.831	3.3	8.3
AdaEva-S	3.944	3.431	3.097	<u>3.014</u>	**2.625**	3.222	8.9	27.8
AdaEva-R	<u>2.694</u>	**2.486**	**2.653**	**2.972**	3.194	**2.800**	**27.2**	**47.8**

Table 6: HiFo-Prompt anytime held-out performance, reported as mean rank (lower is better) at each search-budget checkpoint. Results are macro-averaged over the three CO benchmarks and two LLM backbones (Devstral and Qwen2.5-Coder). Bold and underline denote the best and runner-up strategies, respectively, in each column; higher values are better for the Best % and Top-2 % columns.

	Search Budget Consumed					Overall		
Strategy	20%	40%	60%	80%	100%	Rank ↓	Best % ↑	Top-2 % ↑
Fixed-1	4.472	4.639	4.556	4.639	4.681	4.597	6.7	12.8
Fixed-25%	2.889	<u>2.708</u>	2.806	3.014	2.986	2.881	16.7	32.8
Fixed-50%	3.569	3.167	3.403	3.333	3.569	3.408	6.7	18.9
Full	4.889	5.097	5.111	5.000	4.569	4.933	1.7	5.0
AdaEva-S	**2.472**	**2.625**	<u>2.625</u>	**2.389**	<u>2.681</u>	**2.558**	<u>19.4</u>	**47.8**
AdaEva-R	<u>2.708</u>	2.764	**2.500**	<u>2.625</u>	**2.514**	<u>2.622</u>	**20.6**	<u>45.0</u>

C.3.2 Objective Values per Instance Group

Tables 8 and 9 through Tables 14 and 15 report the underlying objective values for the Devstral backbone, broken down by held-out instance group, while Table 19 reports the corresponding EoH/TSP results with the Gemini-3.1-Flash-Lite backbone. These values provide the magnitude behind the rank comparisons. In particular, the tables show that the gaps between strategies can be small in FSSP even when their ranks are consistently separated, while larger differences emerge for some shifted OBP, higher-dimensional BBOB, and larger TSP instance groups.

For OBP, the capacity shift from $C = 100$ to $C = 500$ produces substantially larger gaps in the held-out objective than the in-distribution setting, particularly for smaller item pools. The relative ordering also becomes more variable under the shift: at the final checkpoint, AdaEva-S achieves the lowest objective across all three $C = 500$ size groups, while its advantage is less uniform on the in-distribution groups. This illustrates that the adaptive strategies do not improve uniformly across every held-out condition, but can become particularly useful when the evaluation distribution differs from that used during search.

For TSP, performance degrades systematically as the held-out instance size increases from the in-distribution group ($\leq$ 100 cities) to 101–150 and then $>$ 150 cities. These extrapolation groups therefore provide a more demanding test for all strategies. AdaEva-R is particularly competitive on the larger groups at intermediate checkpoints, while the differences among strategies are smaller on the in-distribution group at later checkpoints.

For FSSP, the objective differences are comparatively small on the $m = 5$ group, where most methods achieve nearly identical performance. Differences become more visible as the number of machines increases, especially for $m = 20$, although the absolute gaps remain modest. AdaEva-R is consistently competitive on the larger machine groups, while AdaEva-S also becomes increasingly competitive as the search progresses.

For BBOB, the higher-dimensional held-out settings are more difficult for all methods, with AOCC generally decreasing as the dimension increases from 5 to 20. The adaptive methods maintain higher AOCC than the fixed-evaluation strategies across dimensions, with AdaEva-S particularly strong at dimensions 10 and 20 and AdaEva-R strongest

Table 7: LLaMEA anytime held-out performance, reported as mean rank (lower is better) at each search-budget checkpoint. Results are macro-averaged over the 24 BBOB functions and two LLM backbones (Devstral and Qwen2.5-Coder), with the original single-solution $(1 + 1)$-LLaMEA configuration included as an additional reference. Bold and underline denote the best and runner-up strategies, respectively, in each column; higher values are better for the Best % and Top-2 % columns.

	Search Budget Consumed					**Overall**		
Strategy	20%	40%	60%	80%	100%	Rank ↓	Best % ↑	Top-2 % ↑
(1+1)-LLaMEA	3.917	5.000	4.083	4.750	4.833	4.517	0.0	3.3
(10+10)-LLaMEA (Full)	3.917	<u>3.333</u>	3.333	3.250	3.500	3.467	0.0	26.7
Fixed-1	7.000	7.000	7.000	6.833	6.833	6.933	0.0	0.0
Fixed-25%	4.500	3.667	4.500	4.583	4.750	4.400	0.0	13.3
Fixed-50%	5.083	5.000	5.083	4.333	4.583	4.817	3.3	10.0
AdaEva-S	**1.667**	**2.000**	<u>2.167</u>	<u>2.250</u>	<u>2.000</u>	<u>2.017</u>	<u>46.7</u>	<u>70.0</u>
AdaEva-R	<u>1.917</u>	**2.000**	**1.833**	**2.000**	**1.500**	**1.850**	**50.0**	**76.7**

Table 8: Anytime held-out performance of EoH on Online Bin Packing using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for in-distribution capacity C=100 and capacity-shifted C=500 across three instance sizes (1k, 5k, and 10k). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. (Part 1)

		C=100 (in-distribution)			C=500 (capacity-shifted)		
Budget	Method	1k	5k	10k	1k	5k	10k
20%	Fixed-1	**3.59** [2.59, 4.33]	**1.09** [0.69, 1.39]	**0.56** [0.39, 1.01]	6.31 [4.82, 7.05]	**1.83** [1.58, 2.43]	**1.23** [0.93, 1.46]
	Fixed-25%	3.84 [2.34, 4.58]	2.23 [1.78, 2.63]	1.83 [1.56, 2.05]	**4.57** [3.08, 5.56]	3.13 [2.68, 3.28]	2.60 [2.30, 2.93]
	Fixed-50%	4.83 [4.33, 5.57]	2.18 [1.69, 2.83]	1.36 [1.11, 1.71]	12.02 [11.03, 13.26]	5.61 [5.27, 6.16]	4.19 [3.67, 4.92]
	Full	4.83 [3.84, 5.82]	3.87 [3.48, 4.22]	3.32 [3.10, 3.77]	6.06 [4.82, 6.81]	4.67 [4.17, 4.92]	3.50 [3.35, 3.80]
	AdaEva-S	4.58 [3.34, 5.08]	2.63 [2.13, 3.03]	2.18 [2.00, 2.58]	4.82 [3.58, 6.06]	4.32 [4.02, 4.67]	3.30 [3.03, 3.50]
	AdaEva-R	4.83 [3.59, 5.57]	2.13 [1.78, 3.23]	1.28 [0.88, 1.61]	12.52 [11.77, 13.51]	4.47 [4.22, 5.22]	4.19 [3.97, 4.44]
40%	Fixed-1	4.08 [3.59, 4.58]	**1.09** [0.69, 1.54]	**0.51** [0.39, 0.78]	12.77 [11.77, 14.01]	4.07 [3.67, 4.82]	2.33 [2.15, 2.60]
	Fixed-25%	**3.09** [2.10, 4.33]	1.29 [0.79, 1.49]	0.83 [0.64, 1.01]	8.54 [7.80, 10.28]	**2.98** [2.43, 3.33]	**1.66** [1.46, 1.93]
	Fixed-50%	3.84 [3.09, 4.83]	1.39 [0.84, 1.69]	0.86 [0.68, 1.06]	11.03 [9.79, 12.02]	4.42 [3.97, 5.02]	2.43 [2.23, 2.88]
	Full	6.07 [4.83, 6.32]	3.33 [2.93, 3.87]	2.82 [2.60, 3.17]	14.51 [13.51, 15.75]	8.10 [7.65, 8.65]	5.59 [5.07, 5.99]
	AdaEva-S	4.58 [3.34, 5.08]	2.13 [1.78, 2.83]	1.56 [1.46, 1.80]	**5.07** [3.58, 5.81]	4.22 [3.97, 4.67]	3.30 [2.85, 3.50]
	AdaEva-R	4.33 [3.59, 5.08]	1.59 [1.14, 1.88]	0.78 [0.61, 1.03]	13.02 [12.27, 14.75]	4.87 [4.47, 5.22]	2.78 [2.45, 3.20]

overall at several checkpoints. Thus, the advantage observed in the rank analysis is also visible in the underlying objective values rather than arising solely from rank aggregation.

Table 9: Anytime held-out performance of EoH on Online Bin Packing using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for in-distribution capacity C=100 and capacity-shifted C=500 across three instance sizes (1k, 5k, and 10k). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. (Part 2)

		C=100 (in-distribution)			C=500 (capacity-shifted)		
Budget	Method	1k	5k	10k	1k	5k	10k
60%	Fixed-1	4.08 [3.09, 4.83]	1.19 [0.94, 1.69]	**0.54** [0.46, 1.06]	12.77 [11.77, 13.51]	3.97 [3.67, 4.67]	2.33 [2.15, 2.60]
	Fixed-25%	**3.34** [2.10, 3.84]	**1.09** [0.64, 1.49]	0.59 [0.41, 0.88]	10.53 [9.04, 11.53]	**3.42** [2.98, 3.87]	**2.10** [1.78, 2.43]
	Fixed-50%	4.08 [2.84, 4.83]	1.19 [0.94, 1.59]	0.86 [0.73, 1.16]	**9.54** [8.79, 11.53]	3.87 [3.42, 4.67]	2.18 [1.93, 2.35]
	Full	4.33 [3.84, 5.08]	2.78 [2.33, 3.08]	1.95 [1.66, 2.20]	10.53 [9.79, 11.77]	4.07 [3.67, 4.52]	3.45 [3.22, 3.67]
	AdaEva-S	4.08 [3.59, 5.08]	1.54 [0.99, 1.93]	0.96 [0.68, 1.11]	12.52 [11.53, 13.76]	4.47 [3.97, 5.22]	3.10 [2.68, 3.22]
	AdaEva-R	4.58 [3.59, 5.08]	1.49 [0.89, 1.78]	0.78 [0.59, 1.11]	13.26 [12.27, 14.51]	4.42 [4.07, 5.22]	2.53 [2.30, 2.93]
80%	Fixed-1	3.84 [3.09, 4.58]	1.19 [0.94, 1.59]	0.61 [0.49, 1.06]	12.52 [11.53, 13.51]	3.87 [3.47, 4.32]	2.13 [1.88, 2.48]
	Fixed-25%	**3.09** [1.85, 3.84]	**1.09** [0.64, 1.34]	**0.59** [0.41, 0.88]	11.03 [9.79, 11.77]	3.57 [2.98, 3.97]	**1.93** [1.66, 2.35]
	Fixed-50%	3.59 [2.34, 4.33]	**1.09** [0.69, 1.39]	0.64 [0.46, 0.86]	12.77 [11.77, 14.01]	3.67 [3.42, 4.47]	2.13 [1.86, 2.30]
	Full	4.08 [3.34, 5.08]	2.68 [2.33, 3.03]	1.88 [1.56, 2.03]	**6.56** [4.82, 7.55]	**3.28** [2.98, 3.57]	3.07 [2.88, 3.25]
	AdaEva-S	3.84 [2.84, 4.33]	1.14 [0.79, 1.59]	0.68 [0.41, 0.93]	13.76 [12.27, 14.75]	3.97 [3.67, 4.77]	2.15 [2.03, 2.45]
	AdaEva-R	4.33 [3.59, 5.33]	1.34 [0.94, 1.69]	0.76 [0.51, 1.01]	13.51 [12.52, 15.00]	4.87 [4.32, 5.41]	2.68 [2.40, 3.07]
100%	Fixed-1	3.84 [3.09, 4.58]	1.29 [0.94, 1.59]	0.61 [0.49, 1.06]	12.52 [11.53, 13.51]	3.87 [3.47, 4.32]	2.13 [1.88, 2.48]
	Fixed-25%	**2.84** [1.85, 3.84]	1.04 [0.64, 1.29]	0.61 [0.41, 0.88]	10.28 [9.29, 11.77]	3.42 [2.98, 3.77]	1.91 [1.63, 2.30]
	Fixed-50%	3.34 [2.34, 4.33]	**0.89** [0.49, 1.24]	**0.56** [0.39, 0.81]	12.52 [12.02, 14.01]	3.67 [3.33, 4.27]	1.91 [1.68, 2.18]
	Full	4.33 [3.34, 5.08]	1.74 [1.49, 2.23]	1.18 [1.06, 1.51]	12.02 [11.28, 12.77]	4.37 [3.92, 4.92]	2.85 [2.65, 3.10]
	AdaEva-S	**2.84** [1.85, 3.84]	1.14 [0.74, 1.49]	0.66 [0.46, 0.88]	**8.54** [7.55, 9.79]	**2.58** [2.18, 3.23]	**1.48** [1.18, 1.68]
	AdaEva-R	4.08 [3.34, 4.83]	1.34 [0.79, 1.59]	0.68 [0.49, 1.01]	13.51 [12.27, 14.75]	4.32 [4.07, 5.02]	2.60 [2.30, 2.78]

Table 10: Anytime held-out performance of EoH on TSPLIB using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for three instance-size groups: ≤ 100 cities (12 in-distribution instances), 101–150 cities (11 instances), and > 150 cities (6 extrapolation instances). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. (Part 1)

		≤ 100 (in-distribution)	101–150	> 150 (extrapolation)
Budget	Method	12 instances	11 instances	6 instances
20%	Fixed-1	0.034 [0.008, 0.406]	0.168 [0.015, 0.370]	0.694 [0.159, 0.906]
	Fixed-25%	0.028 [0.008, 0.403]	0.318 [0.015, 0.499]	**0.521** [0.124, 0.988]
	Fixed-50%	0.244 [0.026, 0.554]	0.744 [0.258, 1.359]	0.873 [0.827, 1.393]
	Full	4.020 [1.836, 5.801]	4.194 [2.115, 6.330]	2.083 [1.645, 4.086]
	AdaEva-S	1.898 [0.929, 2.384]	1.789 [0.752, 3.003]	1.449 [1.002, 3.244]
	AdaEva-R	**0.020** [0.004, 0.403]	**0.110** [0.002, 0.270]	0.545 [0.051, 1.125]
40%	Fixed-1	0.059 [0.008, 0.403]	0.203 [0.015, 0.381]	0.533 [0.132, 0.685]
	Fixed-25%	**0.024** [0.004, 0.403]	0.148 [0.015, 0.380]	0.427 [0.108, 0.500]
	Fixed-50%	0.071 [0.007, 0.403]	**0.093** [0.007, 0.360]	0.471 [0.102, 1.170]
	Full	1.717 [0.924, 2.767]	1.269 [0.670, 4.543]	1.237 [0.986, 3.476]
	AdaEva-S	**0.024** [0.005, 0.403]	0.239 [0.015, 0.504]	0.492 [0.099, 0.861]
	AdaEva-R	**0.024** [0.004, 0.403]	0.102 [0.015, 0.288]	**0.309** [0.070, 0.398]

Table 11: Anytime held-out performance of EoH on TSPLIB using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for three instance-size groups: ≤ 100 cities (12 in-distribution instances), 101–150 cities (11 instances), and > 150 cities (6 extrapolation instances). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. (Part 2)

		≤ 100 (in-distribution)	101–150	> 150 (extrapolation)
Budget	Method	12 instances	11 instances	6 instances
60%	Fixed-1	0.059 [0.008, 0.403]	0.203 [0.015, 0.381]	0.533 [0.132, 0.685]
	Fixed-25%	0.024 [0.004, 0.403]	0.244 [0.077, 0.526]	0.486 [0.102, 0.705]
	Fixed-50%	**0.012** [0.003, 0.403]	0.085 [0.015, 0.367]	0.399 [0.097, 0.579]
	Full	1.070 [0.681, 2.158]	1.340 [1.130, 3.011]	1.642 [1.277, 3.203]
	AdaEva-S	0.020 [0.004, 0.403]	0.147 [0.015, 0.215]	0.339 [0.074, 0.764]
	AdaEva-R	0.020 [0.004, 0.403]	**0.048** [0.015, 0.286]	**0.197** [0.028, 0.489]
80%	Fixed-1	0.059 [0.008, 0.403]	0.203 [0.015, 0.381]	0.533 [0.132, 0.685]
	Fixed-25%	0.020 [0.004, 0.403]	0.045 [0.003, 0.091]	0.421 [0.105, 0.435]
	Fixed-50%	0.021 [0.004, 0.403]	0.100 [0.002, 0.345]	0.365 [0.053, 0.528]
	Full	1.144 [0.978, 1.391]	0.971 [0.621, 1.949]	1.262 [0.921, 1.609]
	AdaEva-S	**0.012** [0.003, 0.403]	**0.033** [0.008, 0.170]	0.305 [0.064, 0.428]
	AdaEva-R	**0.012** [0.003, 0.403]	0.048 [0.015, 0.267]	**0.242** [0.056, 0.322]
100%	Fixed-1	0.059 [0.008, 0.403]	0.203 [0.015, 0.381]	0.533 [0.132, 0.685]
	Fixed-25%	0.024 [0.004, 0.403]	0.045 [0.003, 0.091]	0.421 [0.105, 0.435]
	Fixed-50%	0.021 [0.004, 0.403]	0.050 [0.002, 0.345]	0.365 [0.053, 0.528]
	Full	0.159 [0.016, 0.406]	0.550 [0.025, 0.698]	0.864 [0.382, 1.236]
	AdaEva-S	0.024 [0.004, 0.403]	**0.033** [0.008, 0.170]	**0.305** [0.064, 0.428]
	AdaEva-R	**0.012** [0.003, 0.403]	0.048 [0.015, 0.267]	0.306 [0.070, 0.495]

Table 12: Anytime held-out performance of EoH on the Taillard flow-shop scheduling benchmark using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for three problem-size groups: m=5 (30 instances), m=10 (40 instances), and m=20 (40 instances). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. (Part 1)

Budget	Method	m=5 (30)	m=10 (40)	m=20 (40)
20%	Fixed-1	**0.00** [0.00, 0.07]	0.51 [0.29, 0.81]	1.49 [1.01, 1.86]
	Fixed-25%	**0.00** [0.00, 0.00]	**0.32** [0.22, 0.57]	1.34 [0.73, 1.57]
	Fixed-50%	**0.00** [0.00, 0.04]	0.43 [0.24, 0.66]	1.39 [0.80, 1.62]
	Full	**0.00** [0.00, 0.10]	0.49 [0.22, 0.76]	1.26 [0.74, 1.61]
	AdaEva-S	**0.00** [0.00, 0.10]	0.49 [0.22, 0.76]	1.29 [0.69, 1.61]
	AdaEva-R	**0.00** [0.00, 0.00]	0.39 [0.23, 0.58]	**1.22** [0.53, 1.46]
40%	Fixed-1	**0.00** [0.00, 0.07]	0.43 [0.25, 0.66]	1.43 [0.84, 1.86]
	Fixed-25%	**0.00** [0.00, 0.00]	0.35 [0.19, 0.56]	1.28 [0.66, 1.61]
	Fixed-50%	**0.00** [0.00, 0.05]	0.37 [0.24, 0.57]	1.44 [0.78, 1.62]
	Full	**0.00** [0.00, 0.10]	0.49 [0.22, 0.76]	1.26 [0.69, 1.61]
	AdaEva-S	**0.00** [0.00, 0.00]	0.41 [0.14, 0.49]	1.27 [0.65, 1.59]
	AdaEva-R	**0.00** [0.00, 0.00]	**0.33** [0.12, 0.55]	**1.22** [0.76, 1.46]
60%	Fixed-1	**0.00** [0.00, 0.07]	0.43 [0.25, 0.66]	1.43 [0.84, 1.86]
	Fixed-25%	**0.00** [0.00, 0.00]	0.39 [0.12, 0.56]	1.22 [0.73, 1.62]
	Fixed-50%	**0.00** [0.00, 0.00]	0.41 [0.24, 0.56]	1.39 [0.68, 1.63]
	Full	**0.00** [0.00, 0.10]	0.49 [0.22, 0.76]	1.26 [0.69, 1.61]
	AdaEva-S	**0.00** [0.00, 0.00]	0.35 [0.12, 0.48]	1.35 [0.79, 1.55]
	AdaEva-R	**0.00** [0.00, 0.00]	**0.32** [0.12, 0.55]	**1.19** [0.70, 1.48]
80%	Fixed-1	**0.00** [0.00, 0.07]	0.43 [0.25, 0.66]	1.43 [0.84, 1.87]
	Fixed-25%	**0.00** [0.00, 0.00]	0.39 [0.15, 0.58]	1.23 [0.71, 1.62]
	Fixed-50%	**0.00** [0.00, 0.00]	0.36 [0.21, 0.60]	1.33 [0.72, 1.61]
	Full	**0.00** [0.00, 0.04]	0.42 [0.21, 0.59]	1.27 [0.67, 1.61]
	AdaEva-S	**0.00** [0.00, 0.00]	0.34 [0.10, 0.64]	1.22 [0.51, 1.43]
	AdaEva-R	**0.00** [0.00, 0.00]	**0.33** [0.19, 0.52]	**1.14** [0.67, 1.47]

Table 13: Anytime held-out performance of EoH on the Taillard flow-shop scheduling benchmark using the Devstral backbone. Cells report the incumbent's objective value at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), for three problem-size groups: m=5 (30 instances), m=10 (40 instances), and m=20 (40 instances). Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Lower values are better. The lower block reports the percentage of instances whose normalized gap is at or below the upper bound (gap ≤ 0). (Part 2)

Budget	Method	m=5 (30)	m=10 (40)	m=20 (40)
100%	Fixed-1	**0.00** [0.00, 0.07]	0.43 [0.25, 0.66]	1.43 [0.84, 1.83]
	Fixed-25%	**0.00** [0.00, 0.00]	0.41 [0.18, 0.58]	1.22 [0.71, 1.56]
	Fixed-50%	**0.00** [0.00, 0.00]	0.32 [0.21, 0.61]	1.23 [0.57, 1.60]
	Full	**0.00** [0.00, 0.05]	0.35 [0.24, 0.57]	1.29 [0.79, 1.59]
	AdaEva-S	**0.00** [0.00, 0.00]	**0.30** [0.09, 0.55]	1.24 [0.49, 1.47]
	AdaEva-R	**0.00** [0.00, 0.00]	0.33 [0.21, 0.59]	**1.13** [0.69, 1.44]
Instances at or below the upper bound (gap ≤ 0)				
20%	Fixed-1	67%	8%	2%
	Fixed-25%	77%	**10%**	5%
	Fixed-50%	73%	5%	2%
	Full	60%	8%	2%
	AdaEva-S	60%	8%	2%
	AdaEva-R	**80%**	8%	**10%**
40%	Fixed-1	70%	10%	0%
	Fixed-25%	80%	10%	5%
	Fixed-50%	70%	5%	5%
	Full	60%	8%	2%
	AdaEva-S	83%	8%	**8%**
	AdaEva-R	**87%**	**15%**	**8%**
60%	Fixed-1	70%	**10%**	0%
	Fixed-25%	**83%**	**10%**	**12%**
	Fixed-50%	77%	**10%**	2%
	Full	60%	8%	2%
	AdaEva-S	**83%**	**10%**	8%
	AdaEva-R	**83%**	**10%**	8%
80%	Fixed-1	70%	10%	0%
	Fixed-25%	87%	**12%**	**12%**
	Fixed-50%	87%	5%	10%
	Full	73%	8%	0%
	AdaEva-S	80%	10%	5%
	AdaEva-R	**90%**	5%	8%
100%	Fixed-1	70%	10%	0%
	Fixed-25%	83%	**12%**	5%
	Fixed-50%	87%	8%	**8%**
	Full	70%	8%	2%
	AdaEva-S	87%	**12%**	2%
	AdaEva-R	**90%**	8%	**8%**

Table 14: Anytime held-out performance of LLaMEA on BBOB using the Devstral backbone. Cells report AOCC at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), evaluated on held-out instances across all 24 BBOB functions and dimensions 5, 10, and 20. *All* denotes the mean across the three dimensions. Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Higher AOCC is better. (Part 1)

Budget	Method	d=5	d=10	d=20	All
20%	(1+1)-LLaMEA	0.505 [0.477, 0.517]	0.458 [0.435, 0.463]	0.429 [0.401, 0.435]	0.464 [0.438, 0.472]
	(10+10)-LLaMEA	0.526 [0.517, 0.543]	0.471 [0.452, 0.474]	0.428 [0.389, 0.431]	0.474 [0.462, 0.475]
	Fixed-1	0.486 [0.458, 0.521]	0.433 [0.424, 0.459]	0.402 [0.400, 0.418]	0.439 [0.428, 0.464]
	Fixed-25%	0.535 [0.514, 0.539]	0.460 [0.457, 0.483]	0.427 [0.387, 0.458]	0.464 [0.459, 0.485]
	Fixed-50%	0.548 [0.520, 0.556]	0.460 [0.456, 0.465]	0.417 [0.385, 0.421]	0.469 [0.462, 0.478]
	AdaEva-S	0.543 [0.531, 0.546]	0.472 [0.428, 0.505]	**0.438** [0.363, 0.466]	0.477 [0.444, 0.510]
	AdaEva-R	**0.584** [0.582, 0.586]	**0.487** [0.465, 0.495]	0.427 [0.398, 0.448]	**0.499** [0.483, 0.510]
40%	(1+1)-LLaMEA	0.513 [0.476, 0.530]	0.460 [0.431, 0.470]	0.428 [0.397, 0.438]	0.467 [0.435, 0.480]
	(10+10)-LLaMEA	0.540 [0.532, 0.543]	0.489 [0.483, 0.494]	0.429 [0.405, 0.437]	0.488 [0.480, 0.496]
	Fixed-1	0.480 [0.459, 0.513]	0.434 [0.425, 0.460]	0.403 [0.401, 0.420]	0.438 [0.429, 0.463]
	Fixed-25%	0.561 [0.549, 0.565]	0.483 [0.465, 0.487]	0.411 [0.390, 0.458]	0.485 [0.474, 0.486]
	Fixed-50%	0.548 [0.520, 0.557]	0.463 [0.430, 0.465]	0.405 [0.366, 0.426]	0.467 [0.451, 0.484]
	AdaEva-S	0.546 [0.546, 0.553]	**0.512** [0.499, 0.517]	**0.478** [0.444, 0.497]	**0.515** [0.496, 0.522]
	AdaEva-R	**0.590** [0.586, 0.592]	0.488 [0.487, 0.497]	0.427 [0.420, 0.445]	0.508 [0.500, 0.511]

Table 15: Anytime held-out performance of LLaMEA on BBOB using the Devstral backbone. Cells report AOCC at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), evaluated on held-out instances across all 24 BBOB functions and dimensions 5, 10, and 20. *All* denotes the mean across the three dimensions. Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Higher AOCC is better. (Part 2)

Budget	Method	d=5	d=10	d=20	All
60%	(1+1)-LLaMEA	0.513 [0.476, 0.530]	0.463 [0.436, 0.470]	0.430 [0.400, 0.438]	0.469 [0.437, 0.480]
	(10+10)-LLaMEA	0.552 [0.539, 0.557]	0.494 [0.476, 0.496]	0.433 [0.408, 0.437]	0.488 [0.483, 0.501]
	Fixed-1	0.472 [0.441, 0.513]	0.426 [0.409, 0.460]	0.395 [0.391, 0.417]	0.430 [0.414, 0.463]
	Fixed-25%	0.561 [0.554, 0.580]	0.473 [0.468, 0.516]	0.396 [0.386, 0.492]	0.484 [0.476, 0.519]
	Fixed-50%	0.555 [0.538, 0.562]	0.463 [0.453, 0.478]	0.417 [0.381, 0.421]	0.469 [0.466, 0.489]
	AdaEva-S	0.554 [0.554, 0.558]	**0.525** [0.521, 0.531]	**0.500** [0.474, 0.512]	**0.526** [0.524, 0.532]
	AdaEva-R	**0.587** [0.581, 0.600]	0.488 [0.477, 0.499]	0.426 [0.416, 0.451]	0.511 [0.489, 0.511]
80%	(1+1)-LLaMEA	0.497 [0.412, 0.529]	0.458 [0.368, 0.468]	0.424 [0.328, 0.436]	0.460 [0.370, 0.478]
	(10+10)-LLaMEA	0.552 [0.539, 0.563]	0.494 [0.478, 0.496]	0.433 [0.420, 0.437]	0.488 [0.487, 0.501]
	Fixed-1	0.472 [0.442, 0.513]	0.426 [0.410, 0.460]	0.395 [0.393, 0.417]	0.431 [0.415, 0.463]
	Fixed-25%	0.559 [0.555, 0.570]	0.462 [0.460, 0.515]	0.385 [0.384, 0.479]	0.475 [0.468, 0.516]
	Fixed-50%	0.557 [0.540, 0.562]	0.461 [0.457, 0.477]	0.411 [0.391, 0.416]	0.470 [0.468, 0.488]
	AdaEva-S	0.555 [0.554, 0.556]	**0.533** [0.525, 0.533]	**0.501** [0.498, 0.517]	**0.535** [0.526, 0.536]
	AdaEva-R	**0.591** [0.581, 0.604]	0.488 [0.476, 0.489]	0.427 [0.409, 0.435]	0.502 [0.488, 0.511]
100%	(1+1)-LLaMEA	0.497 [0.412, 0.529]	0.458 [0.368, 0.468]	0.424 [0.328, 0.436]	0.460 [0.370, 0.478]
	(10+10)-LLaMEA	0.547 [0.541, 0.561]	0.476 [0.474, 0.491]	0.420 [0.408, 0.433]	0.485 [0.483, 0.504]
	Fixed-1	0.468 [0.434, 0.514]	0.425 [0.407, 0.460]	0.395 [0.392, 0.417]	0.429 [0.411, 0.463]
	Fixed-25%	0.563 [0.560, 0.568]	0.464 [0.460, 0.516]	0.386 [0.384, 0.492]	0.474 [0.470, 0.519]
	Fixed-50%	0.558 [0.539, 0.562]	0.464 [0.457, 0.476]	0.411 [0.391, 0.415]	0.470 [0.467, 0.488]
	AdaEva-S	0.559 [0.555, 0.562]	**0.531** [0.528, 0.535]	**0.500** [0.494, 0.515]	**0.531** [0.529, 0.533]
	AdaEva-R	**0.591** [0.588, 0.600]	0.489 [0.476, 0.489]	0.426 [0.411, 0.435]	0.504 [0.489, 0.512]

Table 16: Anytime held-out performance of LLaMEA on BBOB using the Qwen2.5-Coder backbone. Cells report AOCC at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), evaluated on held-out instances across all 24 BBOB functions and dimensions 5, 10, and 20. *All* denotes the mean across the three dimensions. Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Higher AOCC is better. (Part 1)

Budget	Method	d=5	d=10	d=20	All
20%	Fixed-1	0.499 [0.000, 0.515]	0.425 [0.000, 0.432]	0.365 [0.000, 0.385]	0.435 [0.000, 0.439]
	Fixed-25%	0.545 [0.526, 0.568]	0.474 [0.453, 0.488]	0.431 [0.377, 0.434]	0.477 [0.473, 0.497]
	Fixed-50%	0.544 [0.537, 0.624]	0.451 [0.451, 0.460]	0.404 [0.389, 0.415]	0.473 [0.462, 0.483]
	Full	0.548 [0.546, 0.563]	0.480 [0.465, 0.481]	0.420 [0.402, 0.425]	0.482 [0.470, 0.489]
	AdaEva-S	0.554 [0.547, 0.558]	**0.516** [0.496, 0.520]	**0.475** [0.456, 0.487]	0.517 [0.498, 0.518]
	AdaEva-R	**0.583** [0.572, 0.598]	0.506 [0.492, 0.513]	0.462 [0.437, 0.465]	**0.520** [0.500, 0.524]
40%	Fixed-1	0.499 [0.000, 0.515]	0.425 [0.000, 0.432]	0.365 [0.000, 0.385]	0.435 [0.000, 0.439]
	Fixed-25%	0.552 [0.548, 0.594]	0.490 [0.456, 0.513]	0.437 [0.430, 0.448]	0.503 [0.476, 0.506]
	Fixed-50%	0.553 [0.546, 0.614]	0.463 [0.458, 0.468]	0.414 [0.385, 0.423]	0.479 [0.477, 0.486]
	Full	0.565 [0.556, 0.572]	0.481 [0.471, 0.497]	0.420 [0.415, 0.426]	0.487 [0.482, 0.506]
	AdaEva-S	0.559 [0.555, 0.566]	0.517 [0.510, 0.519]	**0.475** [0.468, 0.483]	0.518 [0.516, 0.518]
	AdaEva-R	**0.589** [0.585, 0.623]	**0.518** [0.515, 0.526]	0.472 [0.463, 0.479]	**0.530** [0.525, 0.540]

Table 17: Anytime held-out performance of LLaMEA on BBOB using the Qwen2.5-Coder backbone. Cells report AOCC at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), evaluated on held-out instances across all 24 BBOB functions and dimensions 5, 10, and 20. *All* denotes the mean across the three dimensions. Main entries are medians across five search seeds, with interquartile ranges shown in brackets. Higher AOCC is better. (Part 2)

Budget	Method	d=5	d=10	d=20	All
60%	Fixed-1	0.499 [0.000, 0.515]	0.425 [0.000, 0.432]	0.365 [0.000, 0.385]	0.435 [0.000, 0.439]
	Fixed-25%	0.560 [0.433, 0.561]	0.456 [0.341, 0.496]	0.445 [0.428, 0.455]	0.476 [0.400, 0.504]
	Fixed-50%	0.557 [0.419, 0.596]	0.465 [0.453, 0.485]	0.409 [0.376, 0.437]	0.487 [0.411, 0.493]
	Full	0.580 [0.577, 0.581]	0.488 [0.474, 0.500]	0.419 [0.411, 0.456]	0.495 [0.494, 0.512]
	AdaEva-S	0.565 [0.563, 0.567]	0.519 [0.514, 0.521]	0.466 [0.444, 0.491]	0.519 [0.518, 0.523]
	AdaEva-R	**0.609** [0.601, 0.619]	**0.523** [0.523, 0.529]	**0.479** [0.474, 0.488]	**0.533** [0.532, 0.537]
80%	Fixed-1	0.499 [0.000, 0.515]	0.425 [0.000, 0.432]	0.365 [0.000, 0.385]	0.435 [0.000, 0.439]
	Fixed-25%	0.560 [0.560, 0.596]	0.509 [0.498, 0.513]	0.448 [0.431, 0.453]	0.504 [0.503, 0.513]
	Fixed-50%	**0.608** [0.592, 0.628]	0.487 [0.485, 0.516]	0.440 [0.437, 0.468]	0.512 [0.493, 0.526]
	Full	0.594 [0.580, 0.598]	0.510 [0.499, 0.512]	0.448 [0.444, 0.455]	0.520 [0.519, 0.527]
	AdaEva-S	0.563 [0.558, 0.571]	0.517 [0.517, 0.524]	**0.486** [0.463, 0.497]	0.524 [0.524, 0.526]
	AdaEva-R	0.600 [0.600, 0.621]	**0.525** [0.523, 0.533]	0.482 [0.465, 0.485]	**0.534** [0.529, 0.537]
100%	Fixed-1	0.499 [0.000, 0.515]	0.425 [0.000, 0.432]	0.365 [0.000, 0.385]	0.435 [0.000, 0.439]
	Fixed-25%	0.560 [0.560, 0.596]	0.509 [0.498, 0.513]	0.448 [0.431, 0.452]	0.504 [0.503, 0.508]
	Fixed-50%	0.610 [0.568, 0.626]	0.497 [0.489, 0.526]	0.441 [0.439, 0.462]	0.513 [0.502, 0.515]
	Full	0.599 [0.580, 0.615]	0.510 [0.499, 0.515]	0.458 [0.437, 0.459]	0.527 [0.524, 0.527]
	AdaEva-S	0.572 [0.562, 0.573]	0.519 [0.519, 0.522]	**0.484** [0.467, 0.499]	0.526 [0.525, 0.530]
	AdaEva-R	**0.616** [0.605, 0.618]	**0.531** [0.527, 0.535]	0.479 [0.477, 0.485]	**0.538** [0.534, 0.546]

Table 18: Anytime held-out performance of EoH on TSPLIB using the Gemini-3.1-Flash-Lite backbone. Cells report the median optimality gap across five search seeds at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), with the interquartile range shown beneath. Results are grouped by instance size: ≤ 100 (12 instances), 101–150 (11 instances), and > 150 (6 extrapolation instances). Lower values are better, and the best median within each budget block and instance-size group is shown in **bold**. (Part 1)

		≤ 100 (in-distribution)	101–150	> 150 (extrapolation)
Budget	Method	12 instances	11 instances	6 instances
20%	Fixed-1	0.024 [0.004, 0.403]	0.287 [0.015, 0.404]	0.493 [0.107, 0.694]
	Fixed-25%	**0.020** [0.004, 0.403]	**0.037** [0.002, 0.174]	**0.338** [0.057, 0.635]
	Fixed-50%	0.172 [0.028, 0.405]	0.372 [0.097, 0.681]	0.597 [0.385, 1.248]
	Full	0.608 [0.239, 0.951]	1.513 [0.871, 1.880]	1.612 [1.040, 3.635]
	AdaEva-S	0.170 [0.028, 0.409]	0.372 [0.052, 0.492]	0.632 [0.384, 0.930]
	AdaEva-R	0.024 [0.004, 0.403]	0.157 [0.015, 0.404]	0.360 [0.230, 0.553]
40%	Fixed-1	0.103 [0.008, 0.403]	0.287 [0.015, 0.378]	0.493 [0.107, 0.694]
	Fixed-25%	**0.024** [0.004, 0.403]	**0.028** [0.002, 0.231]	**0.253** [0.060, 0.502]
	Fixed-50%	0.128 [0.026, 0.493]	0.523 [0.092, 1.247]	0.905 [0.642, 1.324]
	Full	0.123 [0.014, 0.403]	0.450 [0.039, 0.684]	0.826 [0.343, 1.211]
	AdaEva-S	**0.024** [0.004, 0.403]	**0.028** [0.002, 0.291]	0.331 [0.084, 0.345]
	AdaEva-R	**0.024** [0.004, 0.403]	**0.028** [-0.001, 0.218]	0.279 [0.068, 0.345]

Table 19: Anytime held-out performance of EoH on TSPLIB using the Gemini-3.1-Flash-Lite backbone. Cells report the median optimality gap across five search seeds at five search-budget checkpoints (20%, 40%, 60%, 80%, and 100%), with the interquartile range shown beneath. Results are grouped by instance size: ≤ 100 (12 instances), 101–150 (11 instances), and > 150 (6 extrapolation instances). Lower values are better, and the best median within each budget block and instance-size group is shown in **bold**. (Part 2)

		≤ 100 (in-distribution)	101–150	> 150 (extrapolation)
Budget	Method	12 instances	11 instances	6 instances
60%	Fixed-1	0.052 [0.008, 0.403]	0.247 [0.015, 0.338]	0.400 [0.101, 0.655]
	Fixed-25%	0.024 [0.004, 0.403]	0.028 [0.002, 0.236]	0.333 [0.060, 0.542]
	Fixed-50%	0.103 [0.008, 0.403]	0.107 [0.015, 0.316]	0.623 [0.365, 1.116]
	Full	0.214 [0.014, 0.516]	0.328 [0.136, 0.759]	0.479 [0.259, 1.195]
	AdaEva-S	**0.012** [0.003, 0.403]	0.014 [0.002, 0.066]	0.241 [0.050, 0.401]
	AdaEva-R	0.020 [0.004, 0.403]	**0.012** [0.002, 0.112]	**0.239** [0.053, 0.426]
80%	Fixed-1	0.134 [0.014, 0.403]	0.247 [0.015, 0.338]	0.400 [0.101, 0.655]
	Fixed-25%	**0.024** [0.004, 0.403]	0.028 [0.002, 0.111]	0.414 [0.100, 0.451]
	Fixed-50%	**0.024** [0.004, 0.405]	0.244 [0.015, 0.406]	0.493 [0.225, 0.791]
	Full	0.103 [0.008, 0.419]	0.372 [0.137, 0.502]	0.641 [0.259, 1.277]
	AdaEva-S	**0.024** [0.004, 0.403]	**0.012** [0.000, 0.063]	**0.260** [0.049, 0.378]
	AdaEva-R	**0.024** [0.004, 0.403]	0.056 [0.002, 0.255]	0.323 [0.192, 0.602]
100%	Fixed-1	0.134 [0.014, 0.403]	0.287 [0.015, 0.339]	0.400 [0.101, 0.655]
	Fixed-25%	0.024 [0.004, 0.403]	0.028 [0.002, 0.106]	0.333 [0.059, 0.816]
	Fixed-50%	0.024 [0.004, 0.403]	0.033 [0.007, 0.274]	0.400 [0.214, 0.571]
	Full	**0.020** [0.004, 0.403]	0.028 [0.002, 0.214]	0.348 [0.083, 0.418]
	AdaEva-S	0.024 [0.004, 0.403]	**0.012** [0.000, 0.095]	**0.304** [0.049, 0.524]
	AdaEva-R	0.024 [0.004, 0.403]	0.028 [0.002, 0.112]	0.328 [0.070, 0.444]

D PARTIAL EVALUATION AND FRAMEWORK FEEDBACK

This appendix describes how AdaEva interacts with framework feedback under partial evaluation. In the main experiments, each framework retains its native feedback mechanism; among the frameworks considered, only LLaMEA directly exposes measured costs in the prompt, and therefore requires a prompt modification for the common-evidence condition. EoH and HiFo-Prompt use evaluation results in their search-side mechanisms but do not include fitness values directly in the prompt (see Section A). We provide the native and modified LLaMEA prompts below, with the latter explicitly indicating the partial-evaluation setting and reporting costs on a common evaluation block.

Plain LLaMEA Prompt (Original)

- **System & instance Specification**: Informs the LLM of its role as an algorithm designer synthesizing Python heuristic functions for the BBOB benchmark suite.
- **Uncalibrated Population Context**: Lists all current population candidates with their uncalibrated average scores:
  - Format: `<Algorithm_Name>: <Description> (Score: <Mean_AOCC>)`
- **Selected Parent & Code Context**: Displays the description, Python source code, execution feedback, and error messages (if any) of the target parent solution selected for mutation.
- **Mutation Directive & Output Format**: Injects a randomly selected mutation operator directive and specifies the required Python function output structure.

Common-Evidence LLaMEA Prompt (Partial Evaluation)

- **System & instance Specification**: Informs the LLM of its role as an algorithm designer synthesizing Python heuristic functions for the BBOB benchmark suite.
- **[ADD-ON 1] Shared instance Context Header**: States explicitly that all non-eliminated population members were evaluated on the **exact same set of shared (instance, seed) instances**:
  - Header: *"The non-eliminated algorithms below were evaluated on the SAME N shared (instance, seed) instances $[f_1 i_1, f_2 i_1, \dots]$, so their scores are directly comparable (eliminated candidates are marked and score -inf)."*
- **[ADD-ON 2] Per-BBOB-Class Performance Breakdown**: Exposes a granular breakdown of AOCC performance across the 5 standard BBOB problem classes (*Separable, Low Conditioning, High Conditioning, Multi-Modal Adequate Structure, Multi-Modal Weak Structure*):
  - Format: `<Name>: <Desc> (Score: <Fit>, per-class AOCC [<sep>, <low_cond>, <high_cond>, <mm_struct>, <mm_weak>])`
- **[ADD-ON 3] Explicit Race Status & Elimination Tagging**: Clearly demarcates eliminated or crashed candidates so the LLM is not misled by obsolete or invalid code:
  - Eliminated: `<Name>: <Desc> (Score: -inf, ELIMINATED by the race --- kept only for diversity, not comparable)`
  - Crashed: `<Name>: <Desc> (Score: -inf, FAILED/crashed and eliminated --- kept only for diversity)`
- **Selected Parent & Code Context**: Displays the description, Python source code, execution feedback, and error messages (if any) of the target parent solution selected for mutation.
- **Mutation Directive & Output Format**: Injects a randomly selected mutation operator directive and specifies the required Python function output structure.

E SENSITIVITY TO THE ADAEVA CONFIGURATION

Protocol. We vary one parameter at a time while holding all others at their default values, so that each ablated configuration differs from the default of its own instantiation in exactly one entry and interactions between parameters are not covered. The sweeps use the Devstral backbone on the three EoH benchmarks and on LLaMEA/BBOB, with five seeds per configuration drawn from the same seed set as the main experiments. For each setting, configuration, and seed, we use the incumbent recorded at each of five budget checkpoints (20%, 40%, 60%, 80%, and 100% of the total evaluation budget) and scored on the full search pool. These values are taken directly from the same validation-trajectory data used to produce the corresponding validation curves.

Let $v_{k,i}(c)$ denote the resulting value of configuration c at checkpoint k and seed i, and let c_0 denote the default configuration of the same AdaEva instantiation. Because configurations and their defaults share the same seed set and therefore the same initial population for each paired seed, differences are computed within seed. We first verify that the seed identifiers match between c and c_0; unmatched seeds are excluded from the comparison, and configurations with fewer than four matched seeds are flagged rather than silently averaged. For each checkpoint k and matched seed i, we compute the percent difference relative to the default:

$$d_{k,i}(c) = \frac{v_{k,i}(c) - v_{k,i}(c_0)}{v_{k,i}(c_0)} \times 100\%, \tag{1}$$

for minimization objectives. For BBOB, where higher AOCC is better, the sign is reversed so that positive values consistently indicate worse performance:

$$\Delta(c) = \frac{1}{5} \sum_{k} \mathrm{median}_i \; d_{k,i}(c), \tag{2}$$

where the median is taken over matched seeds at each checkpoint. Thus, $\Delta(c) > 0$ indicates that configuration c performs worse than its default, while $\Delta(c) < 0$ indicates an improvement. The default configuration has $\Delta(c_0) = 0$ by construction.

Discussion. The sensitivity results show that the default configurations are generally stable on the three EoH benchmarks, with most changes producing deviations of less than 1% in incumbent quality. The largest effects on CO occur for AdaEva-R on OBP, where replacing Friedman with a paired t-test increases the deviation to $+2.39\%$, and for AdaEva-S when increasing $\underline{n}_0$ to 10, which gives a $+1.41\%$ deviation. Other changes are comparatively small, and several configurations are slightly better than the default on individual benchmarks, indicating that the defaults are not uniformly optimal for every setting.

The sensitivity is substantially larger on LLaMEA/BBOB. For AdaEva-R, changing $\underline{n}_0$ from its default of 5 to either 1 or 10 increases the deviation to $+21.37\%$ and $+32.29\%$, respectively, while increasing the evaluation step Δ to 3 gives $+28.92\%$. The statistical and population-management choices also matter: replacing Friedman with a paired t-test, replacing Conover with Nemenyi, disabling population refill, and disabling stochastic evaluation result in deviations of $+14.93\%$, $+21.26\%$, $+19.48\%$, and $+21.16\%$, respectively. For AdaEva-S, the effects are smaller, with $\underline{n}_0 = 10$ changing performance by -2.01% and $\eta = 1.5$ by -7.74% on BBOB. Finally, replacing the native LLaMEA prompt with the partial-evaluation prompt increases the deviation by $+14.99\%$, providing no evidence that the modified feedback is beneficial in this setting.

Overall, the results indicate that the sensitivity of AdaEva depends on both the instantiation and the benchmark. The CO results suggest that the default parameters are relatively robust to moderate changes, whereas BBOB is more sensitive to how AdaEva-R accumulates and interprets evaluation evidence. This supports using fixed, prescribed defaults rather than tuning the configuration separately for each benchmark, while also showing that the choice of racing parameters can materially affect performance in some settings.

Table 20: Configuration sensitivity of AdaEva. Each cell is the percent change in the incumbent's full-pool quality under the ablated configuration relative to the default within the same seed, averaged across budget checkpoints ($\Delta(c)$); positive values indicate worse performance than the default. One factor is varied at a time, so each configuration differs from its default in exactly one parameter. $\underline{n}_0$ is varied separately for AdaEva-S and AdaEva-R, while η is specific to AdaEva-S and the racing parameters are specific to AdaEva-R. The prompt rows correspond to the LLaMEA feedback ablation. Defaults are marked †, and – denotes a configuration with no usable run.

Parameter	Value	EoH/OBP	EoH/TSP	EoH/FSSP	LLaMEA/BBOB
$\underline{n}_0$ (AdaEva-R)	1	-0.22	$+0.02$	0.00	$+21.37$
	5	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	10	$+0.84$	$+0.05$	-0.03	$+32.29$
Δ	1	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	3	$+1.05$	$+0.02$	-0.01	$+28.92$
$\underline{n}_0$ (AdaEva-S)	1	-0.03	-0.08	$+0.02$	$+2.83$
	5	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	10	$+1.41$	-0.17	-0.02	-2.01
α	0.05	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	0.01	$+0.48$	$+0.08$	-0.02	$+24.23$
	0.005	$+0.61$	$+0.03$	-0.04	-0.13
Statistical Test	Friedman	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	t-test	$+2.39$	$+0.04$	-0.06	$+14.93$
Post-hoc Test	Conover	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	Nemenyi	$+1.07$	$+0.04$	-0.04	$+21.26$
ℓ	2	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	6	$+0.96$	$+0.01$	-0.02	$+4.81$
η	1.25	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	1.5	$+0.25$	$+0.26$	-0.02	-7.74
Prompt	Original	–	–	–	$0^{\dagger}$
	Partial Evaluation	–	–	–	$+14.99$
Population Refill	True	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	False	-0.39	$+0.03$	-0.01	$+19.48$
Stochastic Evaluation	True	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$	$0^{\dagger}$
	False	$+0.41$	$+0.09$	$+0.03$	$+21.16$