PACEvolve++: Improving Test-time Learning for Evolutionary Search Agents

Minghao Yan[1,2,†] **Bo Peng**[1] **Benjamin Coleman**[3] **Ziqi Chen**[1] **Zhouhang Xie**[3]
Shuo Chen[3] **Zhankui He**[3] **Noveen Sachdeva**[3] **Weili Wang**[1] **Ed H. Chi**[3]
Shivaram Venkataraman[2] **Wang-Cheng Kang**[3] **Derek Zhiyuan Cheng**[3] **Beidou Wang**[1]
[1]Google [2]University of Wisconsin–Madison [3]Google DeepMind
[†]Work done during an internship at Google.

Abstract

Large language models have become drivers of evolutionary search, but most systems rely on a fixed, prompt-elicited policy to sample next candidates. This limits adaptation in practical engineering and research tasks, where evaluations are expensive, and progress depends on learning task-specific search dynamics. We introduce PACEvolve++, an advisor-model reinforcement learning framework for test-time policy adaptation in evolutionary search agents. PACEvolve++ decouples strategic search decisions from implementation: a trainable advisor generates, assesses, and selects hypotheses, while a stronger frontier model translates selected hypotheses into executable candidates. To train the advisor under non-stationary feedback, we propose a phase-adaptive approach that adapts its optimization strategy to different phases of the evolutionary process. Early in evolution, it uses group-relative feedback to learn broad search preferences; later, as reward gaps compress, it emphasizes best-of-k frontier contribution to support stable refinement. Across expert-parallel load balancing, sequential recommendation, and protein fitness extrapolation, PACEvolve++ outperforms the state-of-the-art evolutionary search framework with frontier models, achieving faster convergence and stabilizing test-time training during evolutionary search.

1 Introduction

Large language models (LLMs) have recently emerged as effective drivers of evolutionary program search, enabling autonomous discovery for open-ended optimization problems [29, 19, 33]. In this paradigm, an agent repeatedly inspects the current best solution, its evaluation metrics, and the search history, proposes candidate mutations, and retains the best-performing descendant. This simple loop has proved remarkably effective: AlphaEvolve [26] demonstrated state-of-the-art algorithm discovery in domains such as bin packing, matrix multiplication, and circle packing, while subsequent open-weight systems extended these gains to symbolic regression and kernel optimization [35, 19]. More recent systems improve the external mechanics of this loop through stronger context management, backtracking, population maintenance, and self-adaptive workflows [45, 5, 23]. These advances make long-horizon search substantially more reliable. Still, they typically rely on a fixed-parameter, prompt-elicited reasoning policy: useful search experience may accumulate in the scaffold, but it is not directly internalized into the model's decision preferences. This leaves a central question open: *how should we adapt an LLM's reasoning policy to make better search decisions during long-horizon evolutionary optimization?*

This need for policy adaptation becomes especially consequential in practical research and engineering tasks [47, 6, 28]. In these domains, effective search decisions often depend on recognizing patterns across previous attempts: which mutation families repeatedly fail, which partial improvements are

Preprint.

worth revisiting, and which directions remain novel relative to the evolving frontier. In recommender-system design [56, 55], MoE load balancing [1, 21], and protein fitness extrapolation [41], candidate directions may range from architectural changes and optimization choices to routing strategies [8], feature interactions [43], and sequence-level transformations [14]. Many such directions can be justified by generic LLM reasoning, but only a few produce measurable improvement after evaluation [22]. A fixed policy can condition on this history through context, but it does not internalize the resulting search feedback into stable decision preferences [2, 57, 25]. Thus, the key challenge is not merely generating plausible hypotheses, but adapting the model's decision policy to prioritize directions that are novel, feasible, and likely to improve the evolving frontier.

We introduce a dedicated advisor model [3] to make search-specific policy adaptation explicit. The advisor learns the strategic decisions in evolutionary search [45], such as hypothesis generation, novelty assessment, and mutation selection, while a stronger frontier implementation model translates the selected hypothesis into executable code [39]. This design departs from standard evolutionary coding frameworks, which often use the same model to both decide what to try and implement the resulting mutation [44, 50]. Such coupling can be suboptimal in practical research and engineering tasks, where implementation failures can arise from complex codebases, integration details, and system constraints [46]. In these settings, the search-specific signal lies primarily in deciding which hypothesis is novel, feasible, and likely to improve the evolving frontier, separate from the model's general coding capabilities [40, 51]. End-to-end training, therefore, entangles hypothesis quality with implementation correctness, making them noisy signals for adapting search preferences. By isolating the advisor as the trainable decision layer, our framework focuses reinforcement learning on what to evaluate next while leveraging frontier coding models for implementation.

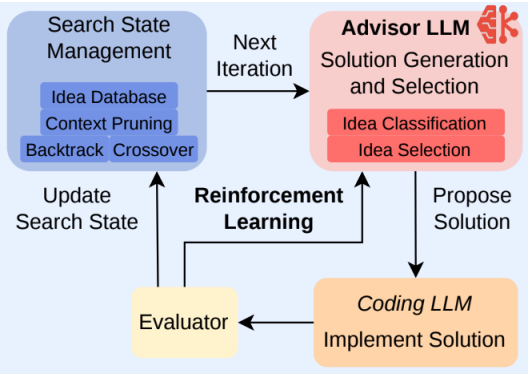


Figure 1: Overall PACEvolve++ workflow. A trainable advisor handles idea generation, novelty assessment, and hypothesis selection, while a frontier implementation model writes code. The RL objective is coupled to rollout batches and adapts its credit assignment to the search phase.

With the advisor model paradigm (§ 3.2), the remaining challenge is to learn from feedback whose usefulness evolves over time. Early in search, the policy should be encouraged to explore broad search directions: candidates often differ substantially in mechanism and quality, and group-relative feedback provides an informative signal for learning which mutation families are promising [22]. Later, however, the search increasingly mutates already strong descendants [20], resulting in marginal differences between candidates that make group-relative signals ineffective. We address this with *phase-adaptive RL* (§ 3.3). During early exploration, we aim to incentivize the advisor to identify useful search directions from diverse candidates without prematurely collapsing onto a few high-scoring rollouts (Figure 2). As search moves toward refinement and reward gaps compress, the objective gradually shifts toward frontier-contribution feedback and assigning credit based on whether a candidate contributes to the evolving best-of-k frontier. This late-stage signal does not simply imitate the highest-scoring rollout; it credits candidates based on their contributions to frontier improvement [42]. The resulting recipe aligns training with the log-diminishing reward structure of evolutionary search dynamics [8], stabilizing late-stage training while avoiding early-stage exploitation (Theorem 1), enabling the policy first to learn broad search preferences and then focus on high-value refinements near the frontier (§ 4). In summary, we introduce an advisor-model reinforcement learning framework (§ 3.1) for self-evolving agents. Our contributions include:

- We design an advisor-based policy adaptation (§ 3.2), where we decouple search-decision learning from code implementation by training an advisor for hypothesis generation, novelty assessment, and mutation selection, while delegating executable-code realization to a stronger frontier implementation model.

- We design a search-dynamics-aware reinforcement learning algorithm (§ 3.3) based on this framework. We develop a phase-adaptive recipe that shifts credit assignment from group-relative feedback

during exploration to frontier-contribution during refinement, aligning policy learning with evolutionary search dynamics.

- Empirically, we demonstrate strong performance across a range of real-world research and engineering tasks (§ 4.1), including expert-parallel load balancing [10], sequential recommendation [48], and protein fitness extrapolation [41], outperforming while converging faster than existing methods with and without RL (§ 4).

2 Background

2.1 Evolutionary Search Agents

An evolutionary search agent improves a program through repeated proposal, evaluation, and selection [16, 11, 17]. Given an initial program p_0, an evaluator $\mathcal{E} : \mathcal{P} \to \mathbb{R}$, and a policy π_θ, the agent generates candidate modifications, evaluates them, and updates the current solution whenever a higher-scoring descendant is identified. At iteration t, the policy conditions on (one of) the current best programs p_t, their evaluation metrics, and the accumulated search history to generate candidates $\{p_t^{(1)}, \ldots, p_t^{(n)}\}$. If the candidate scores high, it is then added to a set of the best candidate programs for future reference.

This line of work has progressed along two complementary directions. The first improves the *search scaffold*. FunSearch [29] and AlphaEvolve [26] showed that strong results can emerge from repeated in-context mutation and selection. At the same time, PACEvolve [45] strengthened long-horizon search through hierarchical context management, momentum-based backtracking, island-style collaboration, and a persistent idea pool. These systems improve how the agent stores, revisits, and coordinates search trajectories over time. The second direction improves the *policy acting within the search loop*. ThetaEvolve [44] trains the mutation policy while treating the evolving program database as the environment, showing that this dynamic search state is essential: reinforcement learning from a static starting point performs worse than learning within the non-stationary evolutionary process. TTT-Discover similarly couples policy learning with evolutionary search with an entropic objective [50]. These results suggest that reinforcement learning in self-evolving systems should be understood as learning over *search dynamics*, rather than optimizing isolated prompts.

2.2 Reinforcement Learning in Evolutionary Search Agents

Two representative self-evolving systems integrate reinforcement learning into an evolutionary search agent. ThetaEvolve [44] uses a GRPO-style objective to train the mutation policy from grouped candidates sampled from the same search state [32]. Given rewards $\{R_1, \ldots, R_n\}$, the normalized advantage for sample i is $\hat{A}_i^{\text{GRPO}} = \frac{R_i - \bar{R}}{\sigma_R + \epsilon_{\text{num}}}$, $\bar{R} = \frac{1}{n}\sum_{j=1}^{n} R_j$, $\sigma_R = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(R_j - \bar{R})^2}$. TTT-Discover [50] instead adopts an entropic reinforcement learning objective with a KL penalty that concentrates gradient mass on exceptional rollouts [18]. Given rewards $\{R_1, \ldots, R_n\}$, the adaptive inverse temperature β is selected such that $\mathrm{KL}(q_\beta \,\|\, \text{uniform}) = \gamma$, where $q_\beta(i) = \exp(\beta R_i)/\sum_{j=1}^{n}\exp(\beta R_j)$, and the leave-one-out advantage for sample i is computed as $\hat{A}_i^{\text{entropic}} = \frac{\exp(\beta(R_i - R_{\max}))}{Z_{-i} + \epsilon_{\text{num}}} - 1$, where $Z_{-i} = \frac{1}{n-1}\sum_{j \neq i}\exp(\beta(R_j - R_{\max}))$. In TTT-Discover, the entropic objective is paired with state reuse, making it well-suited to discovery settings where a single breakthrough branch matters more than average batch quality. These methods show that evolutionary trajectories can provide useful test-time supervision for policy learning. In many research and engineering tasks, strong mutations require domain-specific reasoning about architectural design, optimization, and system trade-offs [53, 52, 14]. At the same time, evaluators are often too expensive for only small rollout groups to be feasible [13]. Under this regime, the choice of reinforcement learning signal inside the search loop becomes a central design decision. In addition, both train the policy as an end-to-end actor, implicitly assuming that the same model can both identify promising search directions and implement them reliably.

3 Method

3.1 Agent Workflow

Figure 1 summarizes the full workflow. The method assumes a population-based evolutionary search agent that exposes the current parent program, recent search history, evaluator scores, and a synchronization point at rollout boundaries. At each iteration, the advisor conditions on the parent program and search history to generate and select a hypothesis. A frontier implementation model converts this hypothesis into a concrete code edit, which the task-specific scorer then evaluates. The resulting outcomes are incorporated into the evolutionary population before the corresponding policy update is performed. After optimization on that rollout batch, the updated advisor parameters are synchronized to the rollout workers and used for the next iteration.

The workflow is organized around two design choices. § 3.2 describes the advisor decomposition, which learns the strategic reasoning policy while delegating code realization to a stronger implementation model. § 3.3 describes the search-dynamics-aware objective, which changes the source of credit assignment as the search moves from exploration to frontier refinement. This design retains the advantages of strong context and search-state management while enabling test-time policy refinement through learned, task-specific search priors. Its decoupled structure also naturally admits off-policy training, requiring only changes to the synchronization barriers imposed by the top-level orchestrator.

3.2 Advisor Model Training

The workflow above separates implementation from reasoning. In MLE tasks (§ 4.1), high-level search reasoning and low-level code implementation have different capacity requirements. Training an open-weight model end-to-end to produce full function-level mutations often fails because the model cannot reliably implement complex candidates, causing the reward to reflect implementation success as much as idea quality (Appendix C).

We therefore apply reinforcement learning to an advisor model [3] tasked with proposing new candidate ideas. The advisor learns the strategic parts of evolutionary search, including idea generation, novelty classification, and hypothesis selection, while a stronger frontier model translates the selected hypothesis into concrete code modifications [9]. This separates *what to try* from *how to implement it*, aligning with the broader post-training practice of developing reasoning and coding as distinct capabilities before composing them in agentic systems.

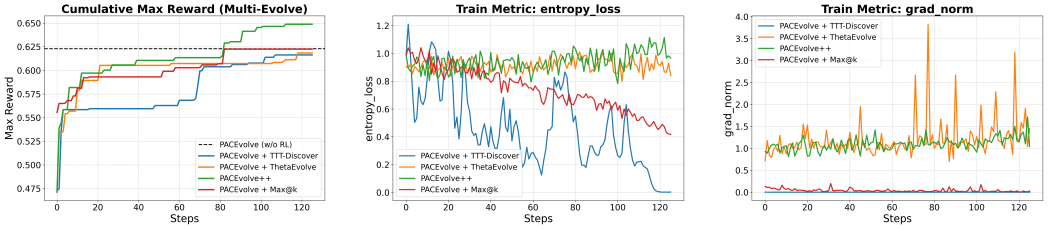
The trained policy therefore serves as an adaptive reasoning layer over the evolving search landscape. Useful mutations depend not only on the current code state, but also on the current phase of the search: whether the frontier requires broader exploration, architectural consolidation, or fine-grained refinement under a fixed evaluation budget. This division enables the advisor to internalize not only static domain knowledge but also dynamic search priors: which mutation families tend to unlock new regions of the search space early, which ideas are worth revisiting after partial progress, and which refinements are likely to yield improvements over the current frontier.

3.3 Search Dynamics Aware Policy Optimization

Training the advisor model requires an RL objective that remains stable when candidate evaluation is costly, and the search frontier is non-stationary. The key design issue is not only the reward scale but also the geometry of credit assignment. Early in the search, candidates often differ in mechanism and quality, so centered-score differences provide useful, dense feedback. Late in search, candidates are often near-neighbor variants of an already strong parent, so the decisive event is whether a response changes the best-of-k frontier. Our objective is designed around this transition.

This transition is especially important in realistic optimization tasks. A single candidate may require GPU training, large-scale simulation, system benchmarking, or multi-dataset validation, leading to evaluation budgets measured in minutes and hours rather than seconds. Under this regime, rollout groups are necessarily smaller, and the RL objective must extract useful learning signals from far fewer candidates.

Prior self-evolving systems [19, 35] were primarily developed for settings with inexpensive evaluators, such as mathematical verification or kernel microbenchmarks, where each candidate can be scored in seconds, and hundreds of rollouts can be generated per optimization step. Recent work on efficient



(a) Multi-Evolve cumulative max reward.

(b) Multi-Evolve policy entropy.

(c) Multi-Evolve gradient norm.

Figure 2: Training dynamics for DeepSeek-R1-0528-Qwen3-8B on Multi-Evolve. PACEvolve++ reaches the best final reward while avoiding the instability patterns observed in baselines.

evolutionary search reduces the full search horizon to a few hundred iterations [19, 4, 23], but existing RL methods for evolutionary agents still rely on much larger rollout batches, often generating 512 candidates per training step [50, 44]. Under this setup, each reinforcement learning step can cost more than an entire sample-efficient evolutionary run [5, 45]. We therefore investigate how to enable robust test-time reinforcement learning within evolutionary search while retaining its sample efficiency.

Search phases in long-horizon evolution. Long-horizon evolutionary search often exhibits log-like marginal reward increase as search progresses due to the increasing difficulty of discovering new state-of-the-art solutions [26, 50]. Early in training, the frontier is broad and diverse: sampled candidates differ substantially in their mechanisms, implementation strategies, and quality [22]. In this exploratory regime, dense token-level relative feedback is particularly valuable when candidate solutions differ significantly.

Later, the search enters a refinement regime, where new state-of-the-art solutions become more difficult to discover. Candidates become local variants of already strong solutions, reward gains exhibit diminishing returns, and absolute score gaps compress toward the level of evaluator noise [8]. The optimization question changes from "which mutation class is broadly better?" to "which candidate meaningfully changes the frontier?" In this regime, entropic weighting over-concentrates on reward outliers, while GRPO amplifies small numerical differences into disproportionately large gradient magnitudes, often causing optimization instability. Recent lines of work have systematically analyzed GRPO's deficiency in small-batch, low-reward-variance regimes, such as high variance [54, 15] and bias for high-likelihood solutions [27].

Figure 2 illustrates these failure modes in practice. The auxiliary traces reveal unstable optimization behavior: entropy can collapse as the objective over-commits to exploitation, while gradient norms spike when compressed rewards are amplified into large updates. These dynamics motivate a training objective whose credit geometry changes with the search phase.

Phase-aligned advantage design. To mitigate the above challenges, we design the training signal around the search dynamics themselves. In the exploratory regime, a raw group-relative baseline preserves dense within-group credit assignment without the late-stage variance blow-up: $\hat{A}_i^G = R_i - \bar{R}$ [24]. To encourage exploration, we also adopt the asymmetric clipping introduced in DAPO, so that more rare but promising tokens can still receive meaningful positive updates [49].

In the refinement regime, we use a pass@k-based marginal-contribution signal (PKPO) [42]. Given N sampled responses from search state x with rewards $g_1, \ldots, g_N$, PKPO constructs unbiased gradient weights w_i such that

$$\nabla_\theta \mathbb{E}[\max(g_1, \ldots, g_k)] = \mathbb{E}\left[\sum_{i=1}^{N} w_i \nabla_\theta \log \pi_\theta(a_i \mid x)\right]. \tag{1}$$

The corresponding PKPO weight can be written as a normalized sum of best-of-k scores over all size-k subsets that contain sample i:

$$w_i = \frac{1}{\binom{N}{k}} \sum_{\substack{I \subseteq \{1,\ldots,N\} \\ |I|=k,\ i \in I}} \max_{j \in I} g_j. \tag{2}$$

Equivalently, this is $\frac{k}{N}$ times the conditional average over size-k subsets that contain i. In practice, we use the low-variance SLOO_{k-1} estimator, which turns this into an explicit marginal-contribution signal by subtracting the best alternative available when i is removed:

$$\hat{A}_i^{\text{top-k}} = w_i^{\text{SLOO}} = \frac{1}{\binom{N}{k}} \sum_{\substack{I \subseteq \{1,\dots,N\} \\ |I|=k,\ i \in I}} \left(\max_{j \in I} g_j - \max_{b \in I \setminus \{i\}} g_b \right). \tag{3}$$

Theorem 1 (Scale-conditioned credit assignment under reward compression)**.** *Let $g_i^{(\delta)} = c + \delta r_i$ be a reward batch with fixed ranking, compression scale $\delta > 0$, base mean $\bar{r}$, and base standard deviation σ_r. Let $\Phi_{\epsilon_{\text{num}}}(B)_i = (B_i - \mu(B))/(\sigma(B) + \epsilon_{\text{num}})$. For the raw group-relative branch $A_i^G(\delta) = g_i^{(\delta)} - \bar{g}^{(\delta)}$ and the SLOO branch $w_i^{\text{SLOO}}(\delta)$, the standardized branches satisfy*

$$\Phi_{\epsilon_{\text{num}}}(A^G(\delta))_i = \frac{\delta(r_i - \bar{r})}{\delta\sigma_r + \epsilon_{\text{num}}}, \qquad \Phi_{\epsilon_{\text{num}}}(w^{\text{SLOO}}(\delta))_i = \frac{\delta\left(w_i^{\text{SLOO}}(1) - \mu(w^{\text{SLOO}}(1))\right)}{\delta\sigma(w^{\text{SLOO}}(1)) + \epsilon_{\text{num}}}.$$

Both standardized branch vectors have squared L_2 norm at most N.

Theorem 1 formalizes the scale-conditioned view used by our objective (proof in Appendix F). When the corresponding branch standard deviation dominates ϵ_{num}, standardization removes the global affine reward scale and preserves the branch-specific credit ordering. In early search, reward variance is large enough that standardized group-relative feedback is a well-conditioned centered score-difference signal. As the search progresses and candidates become increasingly similar, the more important distinction is the geometry of credit assignment: SLOO_{k-1} assigns credit according to whether a response changes a best-of-k frontier. This frontier-contribution geometry is invariant to affine reward rescaling, aligning with late-stage refinement, where absolute gaps are small, but the identity of frontier-changing candidates remains informative.

Phase-adaptive advantage computation. In our training setup, each rollout iteration contains a group of candidates sampled from their respective evolutionary search process. Let $\mathcal{G}_t$ denote this rollout group at iteration t. The raw group-relative and SLOO signals can have different numerical ranges, and PPO-style clipped objectives are sensitive to arbitrary advantage scale. We therefore standardize each scalar estimator within the current rollout group before mixing: $\tilde{A}_i^{(\cdot)} = \frac{\hat{A}_i^{(\cdot)} - \mu_{\mathcal{G}_t}(\hat{A}^{(\cdot)})}{\sigma_{\mathcal{G}_t}(\hat{A}^{(\cdot)}) + \epsilon_{\text{num}}}$, $i \in \mathcal{G}_t$. This step makes the two branches numerically comparable before clipping. The standardized group-relative branch remains a dense z-score over rollout rewards, while the standardized PKPO branch is an affine transform of a frontier-contribution score. Thus, the phase-adaptive mixture changes the source and semantics of credit assignment rather than merely changing the update scale. If the corresponding standard deviation is non-finite or below the numerical threshold ϵ_{skip}, suggesting that the branch has collapsed to numerical noise, we skip this gradient update rather than normalizing an uninformative signal. We then form a mixed scalar score

$$A_i^{\text{mix}}(t) = (1 - \alpha_t)\tilde{A}_i^G + \alpha_t \tilde{A}_i^{\text{top-k}}, \tag{4}$$

The phase schedule, therefore, changes which signal dominates rather than inadvertently changing the overall update magnitude. We scale α_t linearly from 0 to 1 throughout the course of training.

Progress-normalized reward shaping. The RL objective uses a task-specific score as the raw objective, as in any evolutionary search framework [26, 33, 19]. Let y denote a successfully parsed finite score for a given task. Each task specifies an optimization direction together with lower and upper normalization bounds $y_{\min}$ and $y_{\max}$. When explicit score-transform bounds are not provided, these are set from the task configuration as $y_{\min} = \min(y_{\text{init}}, y_{\text{target}})$, $y_{\max} = \max(y_{\text{init}}, y_{\text{target}})$. We then compute a direction-aware normalized progress variable $u(y) = \text{clamp}\left(\frac{y - y_{\min}}{y_{\max} - y_{\min}}, 0, 1\right)$ if the metric is maximized, and change the numerator to $y_{\max} - y$ otherwise. We then define the RL reward as $R_{\text{RL}}(y) = c\,u(y)^{\alpha_r}$, where $c > 0$ is a positive multiplier and $\alpha_r > 0$ is a shaping exponent. In practice, we reduce this to a linearly scaled progress reward on $[0, 5]$. Scores outside the configured range are clamped before normalization. If evaluation fails or the result cannot be parsed, we assign -1.0 as the reward. Parsed but non-finite scores are also mapped to -1.0. Equivalently,

$$R = \begin{cases} R_{\text{RL}}(y), & \text{if } y \text{ is successfully parsed and finite,} \\ -1.0, & \text{otherwise.} \end{cases} \tag{5}$$

This transformation places heterogeneous task metrics, including both maximization and minimization objectives, into a shared progress-based reward scale for RL training.

Loss function. Our estimator produces a scalar advantage per sampled response. Concretely, Eq. 4 defines a response-level score $A_i^{\mathrm{mix}}(t)$ for response i, which is then broadcast to all response tokens: $A_{i,\tau}^{\mathrm{tok}} = A_i^{\mathrm{mix}}(t)$ for all $(i,\tau) \in \mathcal{T}_t$. We then optimize a masked clipped surrogate objective over valid response tokens [31]:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(i,\tau)\sim\mathcal{T}_t}\left[\min\left\{r_{i,\tau}(\theta)A_{i,\tau}^{\mathrm{tok}}, \mathrm{clip}(r_{i,\tau}(\theta), 1-\epsilon_{\mathrm{lo}}, 1+\epsilon_{\mathrm{hi}})\, A_{i,\tau}^{\mathrm{tok}}\right\}\right]. \tag{6}$$

Here $\mathcal{T}_t = \{(i,\tau) : i \in \mathcal{G}_t,\ m_{i,\tau} = 1\}$ denotes the valid response tokens in rollout group $\mathcal{G}_t$, $m_{i,\tau}$ is the response-token loss mask, and $r_{i,\tau}(\theta)$ is the token-level importance ratio.

4 Experiments

4.1 Task Selection

We evaluate PACEvolve++'s performance on a variety of real-world machine-learning-related engineering and research tasks, spanning algorithm design for model routing [21], improvements over the state-of-the-art recommender models [13, 48], and model design for protein engineering [41]. These tasks are grounded in real-world challenges and require innovative solutions for further improvements. Shared training settings and task-specific evaluator timeouts are reported in Appendix B.

4.1.1 Expert-parallelism Load Balancing

Problem. Mixture-of-experts (MoE) models route computation through specialized expert subnetworks, but balancing load across devices during parallel inference remains challenging [34]. The EPLB task asks for an algorithm that, given a workload profile of per-expert demand, assigns experts to devices to minimize the maximum per-device load while remaining computationally efficient.

Evaluation. Candidates are tested on expert-load profiles derived from production MoE traces. We report two metrics: *balancedness*, which measures the uniformity of device load, and *speed*, defined as the inverse of the algorithm's wall-clock time. The final score is their arithmetic mean, as in [8].

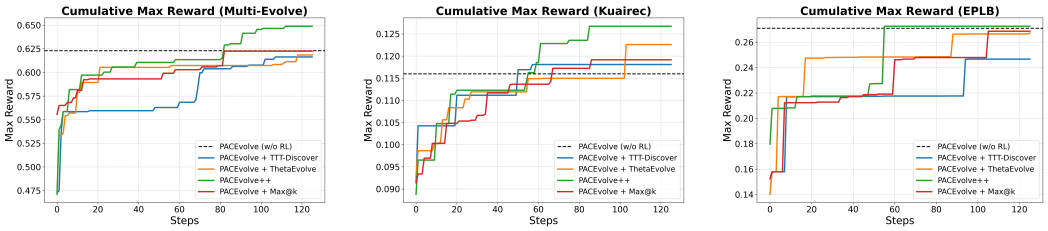
Evolution surface. The evolvable block implements only the assignment logic: its input is an expert-load tensor and its output is a device-assignment map. The evaluation harness, data loading, and metrics are fixed.

4.1.2 Sequential Recommendation

Problem. Sequential recommendation aims to predict a user's next interaction from their history. Our KuaiRec task uses a FuXi-linear-style sequential recommender [48, 13]. Concretely, the benchmark evolves a fixed-budget next-item ranking model on KuaiRec, a fully observed user-item interaction dataset from the Kuaishou short-video platform, with long user histories and time-aware sequence modeling.

Evaluation. Each candidate model is trained for 16 epochs with sampled softmax and evaluated by full-catalog ranking. We report NDCG@10, Hit Rate@10, and MRR, and optimize their arithmetic mean. Each candidate is subject to a 1,200-second wall-clock budget; exceeding this budget results in evaluation failure.

Evolution surface. The evolvable block covers sequence feature construction and the FuXi-linear-style encoder/scoring logic. In particular, candidates can modify how raw histories are converted into item, timestamp, and positional features, as well as the multi-channel sequence mixer, pooling strategy, and item-scoring module. The data pipeline, training loop, sampled-softmax objective, and evaluation protocol are fixed.



(a) Multi-Evolve max reward. (b) KuaiRec max reward. (c) EPLB max reward.

Figure 3: Comparison of different RL algorithms on DeepSeek-R1-0528-Qwen3-8B across three tasks. PACEvolve++ reaches the best final reward and converges the fastest.

4.1.3 Protein Fitness Extrapolation

Problem. Predicting the fitness effect of multiple simultaneous protein mutations from single- and double-mutant training data is challenging [30]. The Multi-Evolve benchmark measures extrapolation: models are trained on wild-type, single, and double mutants, and must then predict fitness for mutants of order three or higher [41].

Evaluation. For each dataset, we report Pearson correlation (r) and Precision@5, defined as the fraction of top-5 predictions that are truly top-5. We define the combined score as $0.7 \times \overline{r} + 0.3 \times \overline{\text{P@5}}$, averaged across datasets.

Evolution surface. The evolvable block covers mutation featurization, pairwise epistatic interactions, regularization and calibration, sample weighting across mutation orders, and lightweight ensembling.

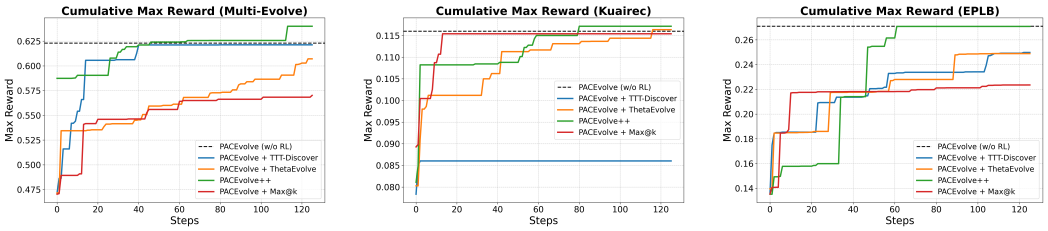
4.2 Baselines

We evaluate all RL variants in the same long-horizon evolutionary search harness. We compare against methods that integrate RL into evolutionary search (ThetaEvolve [44], TTT-Discover [50], and Max@k training [42]) by varying the training setups during advisor training. We also compare against a no-RL scaffold baseline to isolate the effect of test-time advisor training. This setup preserves a strong adaptive workflow while directly comparing approaches for efficient test-time training during evolution, providing a fair testbed for various reinforcement learning recipes. We evaluate on Qwen3.5-4B and DeepSeek-R1-0528-Qwen3-8B to demonstrate our method across different model sizes using Gemini-3.1-pro-preview for candidate implementation. More details on baselines and experiment setups are discussed in Appendix B.

4.3 Results

Figures 3 and 4 compare the main training trajectories. Across EPLB, Sequential Recommendation, and Protein Fitness Prediction, PACEvolve++ consistently provides the strongest final reward while optimizing smoothly and converging the fastest. We note that on EPLB, both PACEvolve++ and the non-RL PACEvolve baseline reach a saturated near-optimal solution [8], but PACEvolve++ uses only half of the evolution budget. On Sequential Recommendation and Protein Fitness Extrapolation, our method converges to a better solution than the baselines. The entropy trace further indicates that PACEvolve++ exhibits fewer spikes and instabilities than baseline methods. In Appendix D.2, we provide disaggregated metrics for each metric we aim to optimize jointly.

Analysis. The auxiliary metrics clarify why the baselines stall. ThetaEvolve's GRPO-style objective remains competitive in raw reward for much of EPLB. Still, Appendix D.1 shows repeated late-stage gradient-norm spikes, including several excursions above 2 and a peak above 4, consistent with variance blow-up once grouped rewards compress. PKPO, shown as Max@k in the figures, exhibits the opposite pathology: its entropy decreases almost monotonically from roughly 1.0 to below 0.4, indicating that it commits to exploitation too early and loses diversity before the search frontier is saturated. The entropic objective is the least stable overall due to concentrated reward distribution.



(a) Multi-Evolve max reward. (b) KuaiRec max reward. (c) EPLB max reward.

Figure 4: Comparison of different RL algorithms on Qwen3.5-4B across three tasks. PACEvolve++ demonstrates faster convergence to better results.

Our search-dynamic-aware objective avoids these pathologies by matching the training signal to the search phase. Early in training, the grouped-relative branch maintains high entropy to sustain exploration, unlike Max@k, which collapses exploration too quickly. Later, the frontier-contribution branch improves refinement without inheriting GRPO's gradient spikes. This behavior is visible in the appendix diagnostics: PACEvolve++ keeps gradient norms in a comparatively narrow band around 1 while maintaining materially higher entropy than Max@k, and these smoother dynamics translate into the strongest final search performance.

5 Related work

Evolutionary search agents. Evolutionary search with language models has developed along two closely related threads. FunSearch [29] and AlphaEvolve [26] showed that repeated propose-evaluate-select loops can turn LLMs into effective algorithmic search operators. At the same time, open-weight successors such as OpenEvolve [33] broadened the set of accessible domains. PACEvolve [45] shifted attention from single-step mutation quality to long-horizon search organization, emphasizing context compression, backtracking, and collaborative exploration. Our work is complementary: we retain a strong scaffold but focus on developing a stronger reasoning policy within it.

Test-time training Test-time training aims to modify the model during inference for better performance [37, 36]. Prior work has explored test-time training in a variety of setups [58, 38, 12]. More recently, methods have been developed to integrate test-time training into evolutionary search agents, as the evolutionary search process can naturally generate on-policy data for reinforcement learning. ThetaEvolve [44] trains the mutation policy against the evolving program database, highlighting the importance of learning within the non-stationary search process rather than relying solely on static prompts. TTT-Discover [50] combines an entropic objective with search-time state reuse and PUCT-style traversal [26], emphasizing discovery settings in which rare breakthroughs matter more than average batch quality.

Policy optimization for LLMs. PPO [31] and its variants underpin RLHF and related post-training methods. GRPO [32] replaces the learned value function with grouped baselines; DAPO [49] adds asymmetric clipping to encourage exploration of low probability tokens. Dr. GRPO [24] removes both standard deviation and length normalization biases from GRPO. Pass@k training [7] develops an entropy-guided approach to optimize pass@k for verifiable tasks. PKPO [42] also targets the pass@k objective and derives unbiased, low-variance gradient estimators via combinatorial weighting.

Advisor models and small-model steering. Advisor-model approaches train compact open-weight models to generate instance-specific guidance for stronger frozen models [3]. Our formulation adopts the same high-level separation of concerns: the trained advisor is responsible for strategic reasoning, whereas the larger frontier model is responsible for faithful implementation. In the self-evolving setting, this design allows task-specific search priors to be learned in the smaller model while preserving the coding strength of the larger implementation model.

6 Conclusion

We introduced PACEvolve++, an advisor-style reinforcement learning framework for self-evolving agents that learns task-specific search priors under expensive evaluation regimes. By decoupling high-level reasoning from implementation and aligning the optimization objective with search dynamics, our approach stabilizes training in practical machine learning research and engineering settings where existing methods struggle. Empirically, PACEvolve++ achieves stronger and more stable search performance across diverse machine learning engineering tasks. These results highlight the importance of improving the reasoning policy, rather than just the search scaffold, to scale self-evolving agents to realistic domains.

Acknowledgment

We gratefully acknowledge the support of the NSF Diamond project OAC-2311767 (Democratizing Large Neural Network Model Training for Science).

References

[1] 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. *arXiv preprint arXiv:2507.19457*, 2025.

[2] GX-Chen Anthony, Dongyan Lin, Mandana Samiei, Doina Precup, Blake Aaron Richards, Rob Fergus, and Kenneth Marino. Language agents mirror human causal reasoning biases. how can we help them think like scientists? In *Second Conference on Language Modeling*, 2025.

[3] Parth Asawa, Alan Zhu, Abby O'Neill, Matei Zaharia, Alexandros G Dimakis, and Joseph E Gonzalez. How to train your advisor: Steering black-box llms with advisor models. *arXiv preprint arXiv:2510.02453*, 2025.

[4] Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. Codeevolve: An open source evolutionary coding agent for algorithm discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025.

[5] 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.

[6] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[7] Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. Pass@ k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*, 2025.

[8] Audrey Cheng, Shu Liu, Melissa Pan, Zhifei Li, Bowen Wang, Alex Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, et al. Barbarians at the gate: How ai is upending systems research. *arXiv preprint arXiv:2510.06189*, 2025.

[9] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[10] DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

[11] David B Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60(2):139–144, 1988.

[12] Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. *Advances in Neural Information Processing Systems*, 35:29374–29385, 2022.

[13] Chongming Gao, Shijun Li, Wenqiang Lei, Jiawei Chen, Biao Li, Peng Jiang, Xiangnan He, Jiaxin Mao, and Tat-Seng Chua. Kuairec: A fully-observed dataset and insights for evaluating recommender systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 540–550, 2022.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[15] Kevin Han, Yuhang Zhou, Mingze Gao, Gedi Zhou, Serena Li, Abhishek Kumar, Xiangjun Fan, Weiwei Li, and Lizhu Zhang. Ebpo: Empirical bayes shrinkage for stabilizing group-relative policy optimization. *arXiv preprint arXiv:2602.05165*, 2026.

[16] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[17] Gregory Hornby, Al Globus, Derek Linden, and Jason Lohn. Automated antenna design with evolutionary algorithms. In *Space 2006*, page 7242. 2006.

[18] Yuhua Jiang, Jiawei Huang, Yufeng Yuan, Xin Mao, Yu Yue, Qianchuan Zhao, and Lin Yan. Risk-sensitive rl for alleviating exploration dilemmas in large language models. *arXiv preprint arXiv:2509.24261*, 2025.

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

[20] Johannes Lengler. Drift analysis. In *Theory of evolutionary computation: Recent developments in discrete optimization*, pages 89–131. Springer, 2019.

[21] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[22] Fei Liu, Qingfu Zhang, Jialong Shi, Xialiang Tong, Kun Mao, and Mingxuan Yuan. Fitness landscape of large language model-assisted automated algorithm search. *arXiv preprint arXiv:2504.19636*, 2025.

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

[24] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

[25] Allen Nie, Yi Su, Bo Chang, Jonathan N Lee, Ed H Chi, Quoc V Le, and Minmin Chen. Evolve: Evaluating and optimizing llms for exploration. *arXiv preprint arXiv:2410.06238*, 2024.

[26] 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.

[27] Daniil Plyusov, Alexey Gorbatovski, Boris Shaposhnikov, Viacheslav Sinii, Alexey Malakhov, and Daniil Gavrilov. F-grpo: Don't let your policy learn the obvious and forget the rare. *arXiv preprint arXiv:2602.06717*, 2026.

[28] Rushi Qiang, Yuchen Zhuang, Anikait Singh, Percy Liang, Chao Zhang, Sherry Yang, and Bo Dai. Mle-smith: Scaling mle tasks with automated multi-agent pipeline. *arXiv preprint arXiv:2510.07307*, 2025.

[29] 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.

[30] Philip A Romero and Frances H Arnold. Exploring protein fitness landscapes by directed evolution. *Nature reviews Molecular cell biology*, 10(12):866–876, 2009.

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[32] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[33] Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025.

[34] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[35] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

[36] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

[37] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pages 9229–9248. PMLR, 2020.

[38] Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025.

[39] Gemini 3 Team. Gemini 3, Nov 2025.

[40] Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

[41] Vincent Q Tran, Matthew Nemeth, Liam J Bartie, Sita S Chandrasekaran, Alison Fanton, Hyungseok C Moon, Brian L Hie, Silvana Konermann, and Patrick D Hsu. Rapid directed evolution guided by protein language models and epistatic interactions. *Science*, page eaea1820, 2026.

[42] Christian Walder and Deep Karkhanis. Pass@ k policy optimization: Solving harder reinforcement learning problems. *arXiv preprint arXiv:2505.15201*, 2025.

[43] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pages 1785–1797, 2021.

[44] Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

[45] Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, et al. Pacevolve: Enabling long-horizon progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026.

[46] John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch?, 2026.

[47] Sherry Yang, Joy He-Yueya, and Percy Liang. Reinforcement learning for machine learning engineering agents. *arXiv preprint arXiv:2509.01684*, 2025.

[48] Yufei Ye, Wei Guo, Hao Wang, Luankang Zhang, Heng Chang, Hong Zhu, Yuyang Ye, Yong Liu, Defu Lian, and Enhong Chen. Fuxi-linear: Unleashing the power of linear attention in long-term time-aware sequential recommendation. *arXiv preprint arXiv:2602.23671*, 2026.

[49] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[50] Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

[51] Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

[52] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152*, 2024.

[53] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. Wukong: Towards a scaling law for large-scale recommendation. *arXiv preprint arXiv:2403.02545*, 2024.

[54] Hongyi Zhou, Kai Ye, Erhan Xu, Jin Zhu, Ying Yang, Shijin Gong, and Chengchun Shi. Demystifying group relative policy optimization: Its policy gradient is a u-statistic. *arXiv preprint arXiv:2603.01162*, 2026.

[55] Jieming Zhu, Quanyu Dai, Liangcai Su, Rong Ma, Jinyang Liu, Guohao Cai, Xi Xiao, and Rui Zhang. Bars: Towards open benchmarking for recommender systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2912–2923, 2022.

[56] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pages 2759–2769, 2021.

[57] Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

[58] Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, et al. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.

A Limitations

Due to the high costs of both RL training and evolutionary search and limited resources, exacerbated by the fact that evaluating each evolutionary candidate involves training a model, we could not repeat the experiments or run them over a longer horizon. We leave it to future work to further scale up our experiments or train models with stronger coding capabilities that may handle evolution end-to-end.

B Training configuration

We show the default training configurations used in the experiments in Table 1. We use the prompt templates (Appendix E) and evolution setups from [45] for candidate generation, selection, and code implementation. We set the temperature to 1 for all prompts. Our experiments are performed in an online on-policy setting; each of the n evolutionary search threads generates a candidate, which is then used for training in the same step. The experiments are performed on A2 instances on GCP.

B.1 Task complexity

We note that the tasks we selected are more complex to implement than those in existing work [44]. Therefore, we found that tasking small open-weight models (with 4B to 8B parameters) with end-to-end evolution results in a low success rate in implementation correctness, thereby biasing the reward toward ideas that are valid when implemented correctly. This also makes it infeasible for an end-to-end ThetaEvolve-style RL training. However, we note that this is mainly a model capacity concern when tasking a compact open-weight model with complex coding tasks. This motivated our design to separate idea generation from code implementation. General coding capability for small, open-weight models is an important concern but out of scope for our study; we leave it to future work to improve few-shot capabilities in implementing a research prototype for complex MLE tasks.

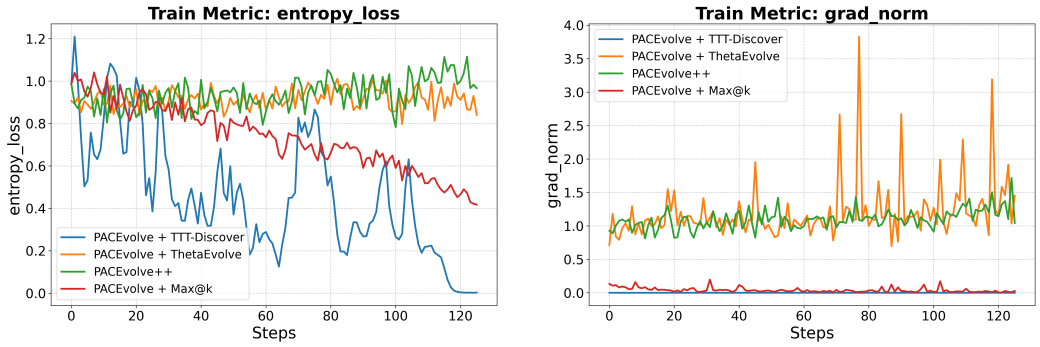
Table 1: Hyperparameter setups

Setting	Value
Evolution iterations	1000
Samples per prompt (n)	8
Top k	4
Optimizer	AdamW
Learning rate	1e-6
Weight decay	0.1
Adam β_1, β_2	0.9, 0.98
Gradient steps per rollout	1
Clip ϵ / ϵ_h	0.2 / 0.28
EPLB timeout	600s
KuaiRec timeout	1,200s
Multi-Evolve timeout	1,200s

C Task details

C.1 EPLB

The EPLB task is drawn from DeepSeek-V3's infrastructure for MoE model serving [21]. Given a workload tensor of per-expert activation counts across a batch, the algorithm assigns experts to parallel devices so that (i) the maximum per-device workload is minimized and (ii) the assignment procedure remains fast. The evolvable code block takes the workload tensor as input and returns a device-assignment map. Workload profiles are derived from the public expert-load dataset introduced in [8].



(a) Multi-evolve entropy.

(b) Multi-evolve gradient norm.

Figure 5: Multi-Evolve training dynamics of 8B models. ThetaEvolve exhibits large gradient-norm spikes, Max@k steadily collapses entropy, and TTT-Discover remains unstable with repeated entropy collapses. PACEvolve++ remains comparatively well conditioned on both metrics.

C.2 KuaiRec

KuaiRec is a fully observed user-item interaction dataset from Kuaishou's short-video platform, containing roughly 7,176 users, 10,728 items, and 12.5 million interactions [13]. The current benchmark instantiates a FuXi-linear-style sequential recommender with a maximum history length of 1,024, an embedding width of 128, four sequence-mixing blocks, and separate retention, temporal, and positional channels. The evolvable surface covers the sequence encoder and scoring logic: candidates can redesign item-, timestamp-, and position-aware token features, the multi-channel sequence mixer, the sequence summarization mechanism, and the item-scoring module. The surrounding scaffold remains fixed, including 16 epochs of sampled-softmax training, full-catalog evaluation, and the relaxed 1,200-second evaluator budget.

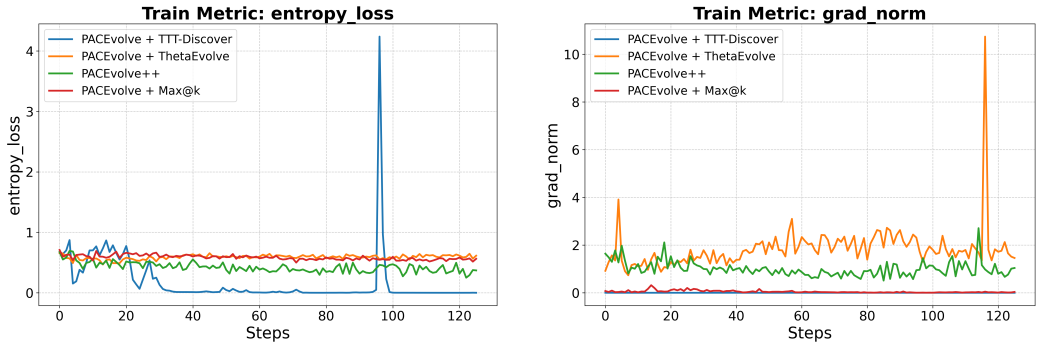
C.3 Multi-Evolve

Multi-Evolve evaluates combinatorial protein fitness prediction [41]. Proteins are mutated at multiple sites simultaneously, and the goal is to predict the joint fitness effect when training data contains only lower-order mutants. Under the same settings as [41], models are trained on wild-type, single, and double mutants, and then predict fitness for mutants with three or more substitutions. The evolvable block covers mutation featurization, pairwise epistatic interaction terms, regularization, sample weighting, and lightweight ensembling. Evaluation is performed across multiple protein datasets using Pearson correlation and Precision@5.

D Additional Results

D.1 Training Diagnostics

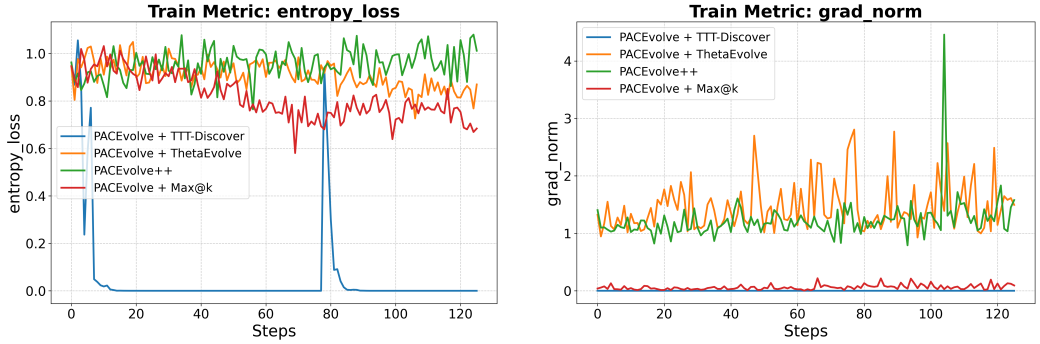
We show more training diagnostics metrics below (Figures 5, 6, 7, 8, 9, 10). These figures further show that PACEvolve++ not only achieves the best performance among other RL algorithms, but also has the most stable training, exhibiting the least unexpected increases or decreases in entropy or gradient norm.



(a) Multi-evolve entropy.

(b) Multi-evolve gradient norm.

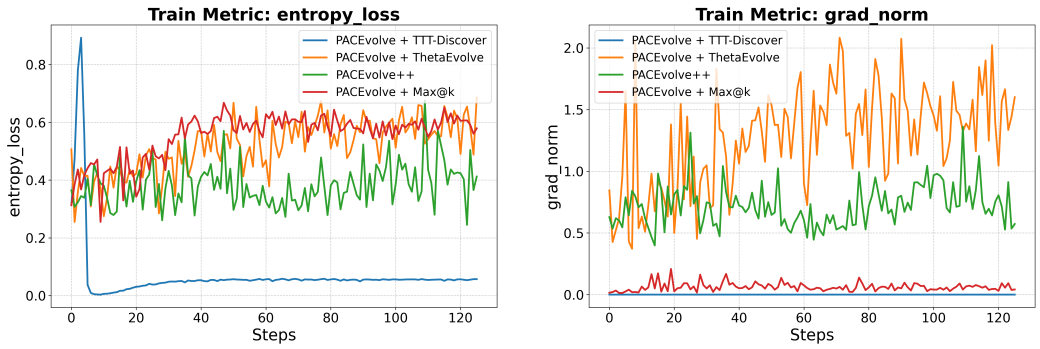
Figure 6: Multi-Evolve training dynamics of 4B models. PACEvolve++ remains the most stable on auxiliary metrics.



(a) KuaiRec entropy.

(b) KuaiRec gradient norm.

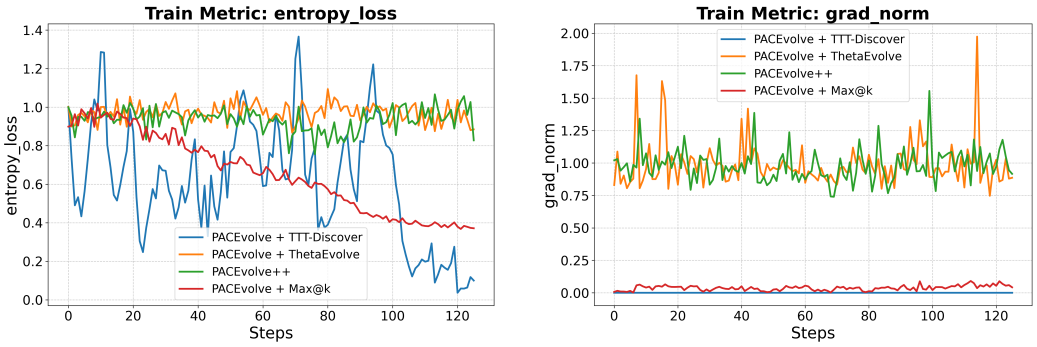
Figure 7: KuaiRec training dynamics of 8B models. ThetaEvolve exhibits large gradient-norm spikes, Max@k steadily collapses entropy, and TTT-Discover training collapsed quickly. PACEvolve++ remains comparatively well conditioned on both metrics.



(a) KuaiRec entropy.

(b) KuaiRec gradient norm.

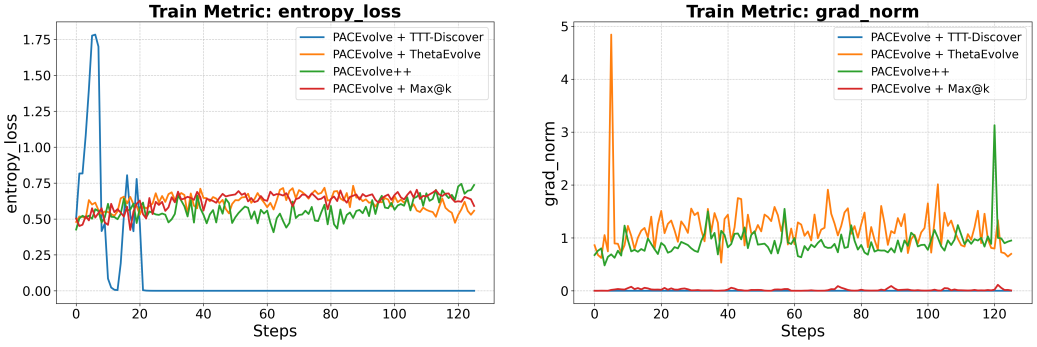
Figure 8: KuaiRec training dynamics of 4B models. ThetaEvolve exhibits large gradient-norm spikes, Max@k steadily collapses entropy, and TTT-Discover training collapsed quickly. PACEvolve++ remains comparatively well conditioned on both metrics.



(a) EPLB entropy.

(b) EPLB gradient norm.

Figure 9: EPLB training dynamics of 8B models. ThetaEvolve exhibits large gradient-norm spikes, Max@k steadily collapses entropy, and TTT-Discover training collapsed quickly. PACEvolve++ remains comparatively well conditioned on both metrics.



(a) EPLB entropy.

(b) EPLB gradient norm.

Figure 10: EPLB training dynamics of 4B models. ThetaEvolve exhibits large gradient-norm spikes, Max@k steadily collapses entropy, and TTT-Discover training collapsed quickly. PACEvolve++ remains comparatively well conditioned on both metrics.

D.2 Disaggregated Metrics

In this section, we report disaggregated metrics for the best candidate found by each RL variant. We omit the combined evolution score and the iteration at which the candidate was found. All values are rounded to three decimal places. We abbreviate DeepSeek-R1-0528-Qwen3-8B as DS-R1-Qwen3-8B in the tables. Method abbreviations are TTT-D for TTT-Discover, PACE++ for PACEvolve++, and Max@k for the PKPO objective.

Table 2: Disaggregated EPLB metrics. Bal. denotes balancedness.

Method	Qwen3.5-4B		DS-R1-Qwen3-8B	
	Bal.	Speed	Bal.	Speed
TTT-D	0.446	0.054	0.443	0.050
GRPO	0.443	0.055	0.478	0.056
PACE++	0.489	0.053	0.510	0.035
Max@k	0.407	0.040	0.458	0.080

Table 3: Disaggregated KuaiRec metrics. N@10/50 denotes NDCG@10/50 and H@10/50 denotes HR@10/50.

Method	Qwen3.5-4B					DS-R1-Qwen3-8B				
	N@10	N@50	H@10	H@50	MRR	N@10	N@50	H@10	H@50	MRR
TTT-D	0.073	0.101	0.116	0.250	0.069	0.103	0.134	0.155	0.302	0.097
GRPO	0.101	0.133	0.153	0.304	0.095	0.107	0.138	0.159	0.303	0.101
PACE++	0.102	0.133	0.154	0.302	0.096	0.110	0.145	0.166	0.327	0.104
Max@k	0.099	0.130	0.156	0.300	0.091	0.104	0.136	0.156	0.309	0.098

Table 4: Disaggregated Multi-Evolve metrics. P@5 denotes Precision@5.

Method	Qwen3.5-4B		DS-R1-Qwen3-8B	
	Pearson r	P@5	Pearson r	P@5
TTT-D	0.703	0.430	0.703	0.415
GRPO	0.675	0.449	0.688	0.456
PACE++	0.726	0.440	0.733	0.452
Max@k	0.630	0.430	0.697	0.448

These disaggregated results show that methods might be exploring different fronts along the Pareto-optimal curve. While a higher combined score generally signals stronger overall solutions, it does not guarantee Pareto dominance. For example, EPLB exposes a trade-off between balance and speed, KuaiRec separates short-horizon ranking quality from broader hit-rate coverage, and Multi-Evolve separates correlation from top-ranked mutant precision.

E Prompt templates

The following are the prompt templates used for advisor and code implementation (Replace task-related information to deploy on other tasks). Texts in red represent task-specific placeholders; texts in blue represent dynamic context managed during the evolutionary search.

Idea Generation Prompt Template

```
We are conducting an evolutionary optimization process for the Expert
Parallelism Load Balancer (EPLB).
BACKGROUND
TASK INTRO
CODING REQ
Current state-of-the-art
The current state-of-the-art solution is as follows:
SoTA Solution
Idea Repo
Idea repos contain the ideas we have generated so far and the
experiments we have run to test these hypotheses.
Idea Repo
Your Task
When proposing a new design, you should start by conducting a research
brainstorming exercise to develop 3 options to explore the design
space.  Go through each option and provide a comprehensive explanation
of the proposed changes.
Go through the idea and experiment history carefully; DO NOT
re-propose an idea that has already been well tested.
You should follow the following format when generating ideas:
Idea 1
Hypothesis:
Reasoning:
Idea 2
Hypothesis:
Reasoning:
Idea 3
Hypothesis:
Reasoning:
```

Idea Selection Prompt Template

```
We are conducting an evolutionary optimization process for the Expert
Parallelism Load Balancer (EPLB).
BACKGROUND
TASK INTRO
CODING REQ
Current state-of-the-art
The current state-of-the-art solution is as follows:
SoTA Solution
Idea Repo
Idea repos contain the ideas we have generated so far and the
experiments we have run to test these hypotheses.
Idea Repo
Your Task
Your job is to come up with an experiment to test one of the ideas in
the idea repo.  Think about an experiment to run that will have the
best shot of helping you accomplish your overall goal.
You should use the following format for the idea selection part:
Idea ID: Experiment description:
```

Code Implementation Prompt Template

```
We are conducting an evolutionary optimization process for the Expert
Parallelism Load Balancer (EPLB).
BACKGROUND
TASK INTRO
CODING REQ
Current state-of-the-art
The current state-of-the-art solution is as follows:
SoTA Solution
Your Task
Your job is to implement the selected idea above.
Idea ID: Idea ID
Experiment description:  Experiment description
Selected idea:  Selected Idea
```

F Training stability

We analyze the scale-conditioned advantages of the hybrid objective when absolute reward differences compress, but relative ordering is preserved. This corresponds to late-stage evolution, where candidate solutions become local variants of already strong programs. The goal is to understand what standardization preserves: not an absolute reward scale, but a bounded credit-assignment geometry.

Let $r_1, \dots, r_N \in \mathbb{R}$ be a non-constant reward profile with mean $\bar{r} = \frac{1}{N}\sum_{i=1}^{N} r_i$ and standard deviation $\sigma_r > 0$. Assume strict ordering (i.e., $r_i \neq r_j$ for $i \neq j$) and $2 \leq k \leq N$.

For any offset $c \in \mathbb{R}$ and scale $\delta > 0$, define a scaled reward batch

$$g_i^{(\delta)} = c + \delta r_i.$$

This construction models *reward compression*: as δ becomes small, rewards become closer together while their rankings remain unchanged.

Then the ordering is preserved:

$$r_i > r_j \iff g_i^{(\delta)} > g_j^{(\delta)}.$$

Define the raw group-relative branch

$$A_i^G(\delta) = g_i^{(\delta)} - \bar{g}^{(\delta)},$$

and the SLOO_{k-1} weight

$$w_i^{\text{SLOO}}(\delta) = \frac{1}{\binom{N}{k}} \sum_{\substack{I \subseteq \{1,\dots,N\} \\ |I|=k,\ i \in I}} \left(\max_{j \in I} g_j^{(\delta)} - \max_{b \in I \setminus \{i\}} g_b^{(\delta)} \right).$$

For any non-constant branch vector $B = (B_1, \dots, B_N)$, define its scale-conditioned version

$$\Phi_{\epsilon_{\text{num}}}(B)_i = \frac{B_i - \mu(B)}{\sigma(B) + \epsilon_{\text{num}}}.$$

Then:

$$A_i^G(\delta) = \delta(r_i - \bar{r}), \tag{7}$$

$$\|A^G(\delta)\|_2^2 = N\delta^2\sigma_r^2, \tag{8}$$

$$w_i^{\text{SLOO}}(\delta) = \delta\, w_i^{\text{SLOO}}(1), \tag{9}$$

$$\|w^{\text{SLOO}}(\delta)\|_2^2 = \delta^2\, \|w^{\text{SLOO}}(1)\|_2^2. \tag{10}$$

Proof of Theorem 1. We prove the result in three steps: first, deriving the scale-conditioned forms; second, showing boundedness; and finally, characterizing the SLOO credit geometry.

We first analyze how the group-relative branch changes when rewards are scaled.

Since $g_i^{(\delta)} = c + \delta r_i$, we compute:

$$\bar{g}^{(\delta)} = c + \delta \bar{r}, \qquad \sigma(g^{(\delta)}) = \delta \sigma_r.$$

We note that adding a constant shifts the mean but does not affect variance, while scaling by δ scales the standard deviation linearly.

Substituting into the standardization operator gives

$$\Phi_{\epsilon_{\text{num}}}(A^G(\delta))_i = \frac{\delta(r_i - \bar{r})}{\delta\sigma_r + \epsilon_{\text{num}}}.$$

Taking the squared norm:

$$\|\Phi_{\epsilon_{\text{num}}}(A^G(\delta))\|_2^2 = \frac{N\delta^2\sigma_r^2}{(\delta\sigma_r + \epsilon_{\text{num}})^2}.$$

For the raw group-relative signal, since it only centers the rewards, we obtain directly:

$$A_i^G(\delta) = g_i^{(\delta)} - \bar{g}^{(\delta)} = \delta(r_i - \bar{r}),$$

and thus

$$\|A^G(\delta)\|_2^2 = N\delta^2\sigma_r^2.$$

We now analyze how the SLOO estimator behaves under the same transformation.

The key observation is that the $\max$ operator has two properties:

1. *Translation equivariance:* $\max(c + x_i) = c + \max(x_i)$

2. *Positive homogeneity:* $\max(\delta x_i) = \delta \max(x_i)$ for $\delta > 0$

Applying these to any subset I, we obtain:

$$\max_{j \in I} g_j^{(\delta)} = c + \delta \max_{j \in I} r_j,$$

$$\max_{b \in I \setminus \{i\}} g_b^{(\delta)} = c + \delta \max_{b \in I \setminus \{i\}} r_b.$$

Subtracting, the constant c cancels:

$$\max_{j \in I} g_j^{(\delta)} - \max_{b \in I \setminus \{i\}} g_b^{(\delta)} = \delta \left(\max_{j \in I} r_j - \max_{b \in I \setminus \{i\}} r_b \right).$$

Therefore, SLOO depends on winner-changing margins, and its raw signal scales linearly with δ.

Averaging over subsets yields

$$w_i^{\text{SLOO}}(\delta) = \delta\, w_i^{\text{SLOO}}(1),$$

and thus

$$\|w^{\text{SLOO}}(\delta)\|_2^2 = \delta^2 \, \|w^{\text{SLOO}}(1)\|_2^2.$$

Applying $\Phi_{\epsilon_{\text{num}}}$ gives

$$\Phi_{\epsilon_{\text{num}}}(w^{\text{SLOO}}(\delta))_i = \frac{\delta \left(w_i^{\text{SLOO}}(1) - \mu(w^{\text{SLOO}}(1)) \right)}{\delta\sigma(w^{\text{SLOO}}(1)) + \epsilon_{\text{num}}}.$$

We now show boundedness. For any non-constant branch B,

$$\|\Phi_{\epsilon_{\text{num}}}(B)\|_2^2 = \frac{N\sigma(B)^2}{(\sigma(B) + \epsilon_{\text{num}})^2} \leq N.$$

This applies to both $A^G(\delta)$ and $w^{\mathrm{SLOO}}(\delta)$. Moreover, if $B_i^{(\delta)} = \delta B_i^{(1)}$, then $\Phi_{\epsilon_{\mathrm{num}}}(B^{(\delta)})$ approaches the z-score vector $(B_i^{(1)} - \mu(B^{(1)}))/\sigma(B^{(1)})$ whenever $\delta\sigma(B^{(1)}) \gg \epsilon_{\mathrm{num}}$, and approaches zero whenever $\delta\sigma(B^{(1)}) \ll \epsilon_{\mathrm{num}}$. Thus, the scale-conditioned branch is bounded in the diverse regime and naturally becomes uninformative in the collapsed regime, where our implementation skips the update.

It remains to characterize what the SLOO branch preserves after scale conditioning. For any subset I containing i, the term

$$\max_{j \in I} r_j - \max_{b \in I \setminus \{i\}} r_b$$

is positive if and only if i is the highest-reward element in I. Otherwise, removing i does not change the subset maximum, and the term is zero. If responses are ranked in decreasing reward order and i has rank m, then i can be the winner only in subsets whose other $k-1$ elements are drawn from the $N-m$ lower-ranked responses. Therefore, the bottom $k-1$ responses cannot win any size k subset and receive zero raw SLOO contribution. Standardization is an affine transform with positive scale, so it preserves the ordering induced by these SLOO frontier-contribution scores.

The theorem follows. The group-relative branch provides dense, centered reward credit, which is useful when early-stage rollout groups are diverse. The SLOO branch gives frontier-contribution credit, retaining the same ordering of candidates after scale conditioning and aligning better with late-stage best-of-k evolutionary survival.

□