What Makes an LLM a Good Optimizer? A Trajectory Analysis of LLM-Guided Evolutionary Search

Xinhao Zhang, Xi Chen, François Portet, Maxime Peyrard
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
{xinhao.zhang, maxime.peyrard}@univ-grenoble-alpes.fr

Project Website: github.io/traj_evo_search

Abstract

Recent work has demonstrated the promise of orchestrating large language models (LLMs) within evolutionary and agentic optimization systems. However, the mechanisms driving these optimization gains remain poorly understood. In this work, we present a large-scale study of LLM-guided evolutionary search, collecting optimization trajectories for 15 LLMs across 8 tasks. Although zero-shot problem-solving ability correlates with final optimization outcomes, it explains only part of the variance: models with similar initial capability often induce dramatically different search trajectories and outcomes. By analyzing these trajectories, we find that strong LLM optimizers behave as local refiners, producing frequent incremental improvements while progressively localizing the search in semantic space. Conversely, weaker optimizers exhibit large semantic drift, with sporadic breakthroughs followed by stagnation. Notably, various measures of solution novelty do not predict final performance; novelty is beneficial only when the search remains sufficiently localized around high-performing regions of the solution space. Our results highlight the importance of trajectory analysis for understanding and improving LLM-based optimization systems and provide actionable insights for their design and training.

1 Introduction

Large language models (LLMs) are increasingly deployed as search operators in iterative optimization systems (Lehman et al., 2022; Peyrard et al., 2025). Across diverse domains, such as prompt optimization (Agrawal et al., 2026; Guo et al., 2025; Fernando et al., 2023) and scientific discovery (Romera-Paredes et al., 2023; Ellenberg et al., 2025; Novikov et al., 2025; Gottweis et al., 2025), LLMs are embedded into evolutionary or agentic loops as black-box optimizers where they repeatedly propose candidate solutions, receive feedback, and refine solutions iteratively. While

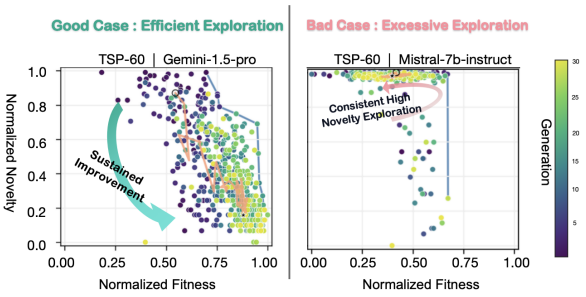


Figure 1: **Different optimization trajectories for two LLMs with similar zero-shot performance on TSP-60.** Each point represents a candidate solution, colored by generation. Gemini-1.5-Pro (left) displays sustained fitness improvement and progressive localization. Mistral-7B-Instruct (right) maintains high novelty but fails to exploit it into fitness gains.

such LLM-guided evolutionary workflows have been shown to deliver substantial empirical gains, the mechanisms underlying these improvements remain poorly understood. In particular, even under strictly controlled optimization loops, selection rules, and evaluation functions, different LLMs exhibit vastly different optimization trajectories and final performances. This observation motivates the central question of this work: ***what explains such large model-to-model differences in optimization performance?*** Are these differences primarily a reflection of base model capability, or do they arise from more subtle differences in the exploration–exploitation dynamics induced by the models?

To address these questions, we conduct a large-scale study of LLM-based evolutionary optimization, collecting optimization trajectories for 15 LLMs across 4 task families (8 tasks), resulting in 72K analyzed candidate solutions. As expected, zero-shot performance correlates positively with final optimization outcomes. However, this relationship explains partially the variance: models with nearly identical zero-shot performance can diverge quickly after evolution, following qualitatively distinct optimization trajectories (see Fig-

Findings of the Association for Computational Linguistics: ACL 2026, pages 24987–25011

ure 1). To explain this residual gap, we turn to the classical exploration–exploitation trade-off underlying evolutionary algorithms (Sun et al., 2018). In LLM-driven metaheuristics, the mutation operator is no longer fully random but is strongly shaped by the LLM's prior toward producing improved solutions conditioned on parent candidates and their fitness feedback. Consequently, exploration is more constrained than in classical, non-LLM-driven evolution. Under this view, optimization processes should benefit primarily from higher novelty and diversity, which would expose the system to a broader range of potentially useful regions in the search space. Surprisingly, our results contradict this intuition. We find that the most successful trajectories are not those exhibiting high novelty, but rather those characterized by *frequent and sustained breakthroughs*, where a breakthrough represents an incremental improvement of the best-so-far fitness. In other words, what distinguishes strong optimization runs is more of the ability to reliably produce incremental improvements repeatedly. This is different from typical behavior observed in meta-heuristics, where large breakthroughs occur at rare intervals, followed by long plateaus or small refinements (Mitchell and Taylor, 1999). This interpretation is also supported by our perturbation experiments, where we directly manipulate the refinement behaviour of the search trajectory through model mixing, leading to predictable changes in optimization performance.

To further elucidate when and why breakthroughs happen, we analyze the **geometry of optimization trajectories in semantic space**. By embedding candidate solutions and characterizing within-generation distributions with entropy-based and dispersion measures, we reveal a clear distinction between strong and weak optimizers. Effective LLM operators progressively *localize* their search around high-performing regions of semantic space, whereas weaker optimizers continue to diffuse and drift across distant regions. A generation-level mixed-effects analysis further uncovers an interesting interaction between novelty and semantic dispersion: novelty increases the probability of breakthroughs *only when* the search remains sufficiently localized. Outside this regime, novelty is largely unproductive.

Contributions. We make the following contributions: **(i)** We conduct a large-scale, controlled study of LLM-based evolutionary optimization and release the resulting optimization trajectories. **(ii)** We show that differences in optimization performance between LLMs are only partially explained by zero-shot capability, unveiling a distinct notion of *optimizable ability*. **(iii)** We identify effective LLM optimizers as *local refiners*, whose trajectories progressively localize in semantic space and yield frequent incremental breakthroughs, and we support this mechanism through perturbation experiments that highlight the role of refinement behavior. **(iv)** We demonstrate that novelty is not inherently beneficial; its utility is conditional on the geometric regime of search, and it becomes productive only when search remains localized. **(v)** We derive practical implications for model selection and for learning better search operators, showing that smaller or cheaper models can outperform stronger base models when they exhibit more reliable refinement behavior.

More broadly, our semantic trajectory analysis offers a reusable framework for studying LLM-driven optimization processes. Our findings suggest that, rather than solely pursuing general-purpose capability, future work may benefit from understanding, controlling, training models as effective search operators (Šurina et al., 2025) emphasizing local refinement and error correction.

2 Related Work

Evolutionary Computation with LLMs LLMs are increasingly integrated into evolutionary computation frameworks (Yang et al., 2024; Wu et al., 2025; Brahmachary et al., 2024; Tao et al., 2024), revitalizing meta-heuristic optimization. Unlike classical approaches that rely on stochastic operators to explore optimization landscapes (Holland, 1992), LLM-assisted methods instantiate variation operators through the semantic priors of LLMs (Josifoski et al., 2023; Peyrard et al., 2025; ang Gao et al., 2026; Fang et al., 2025). These frameworks have demonstrated strong empirical performance across domains including combinatorial optimization (Yang et al., 2025a; Yu et al., 2026) and scientific discovery (Yang et al., 2025b; MacKnight et al., 2025; Chen et al., 2026; Abhyankar et al., 2026; Zhou et al., 2024). Recent work has further extended these approaches to algorithm discovery in open-ended scenario (Lu et al., 2024; Gottweis et al., 2025; Qu et al., 2026), as well as more advanced settings incorporating co-evolution or meta-reflection to mitigate limited global search perspectives (Fei Liu, 2024; Ye et al.,

2024). Our work complements these efforts by analyzing the search behavior and optimization trajectories induced by LLMs. Trajectory analyses can also inform the design of agentic systems (Lee et al., 2026; Zhao et al., 2026, 2025; Lin et al., 2025). Related to our work are behaviour-space studies (van Stein et al., 2025) and LAS landscape analyses (Liu et al., 2025), which similarly associate effective optimization with sustained improvements and increased exploitation. We extend these findings with a unified cross-model, cross-task analysis using semantic entropy measures.

Evaluating LLMs as Search Operators The evaluation paradigm for LLMs has evolved accordingly. Early optimization benchmarks relied on single-pass prompting (Fan et al., 2024; Duchnowski et al., 2025). More recent work evaluates LLMs within iterative or evolutionary search loops, treating them as search operators guided by external feedback (Li et al., 2025; Huang et al., 2024; Ouyang et al., 2025; Shojaee et al., 2025a,b). In this setting, LLMs are no longer assessed as one-shot solvers but as *semantic search operators*, whose preferences and biases matter (Zhou et al., 2026). While these benchmarks demonstrate strong end-to-end performance, evaluation remains largely outcome-centric. Our results show that base model capability and operator effectiveness are distinct skills, with direct implications for model selection and motivating work on learning specialized search operators, such as Brahmachary et al. (2024) and EvoTune (Šurina et al., 2025).

3 Methodology

Following Novikov et al. (2025), we implement a lightweight evolutionary search loop where LLMs act as semantic variation operators, iteratively generating candidate solutions in order to optimize task-specific fitness.

Population Initialization. For each task, we construct an initial population $\mathcal{P}_0$ consisting of valid genomes and their corresponding fitness values $(g, f_T(g))$. This initial population is fixed and shared across all models for the same task.

Fitness Evaluation. Each genome is evaluated by a task-specific fitness function $f_T(\cdot)$. Invalid or unparsable outputs are assigned zero fitness.

Selection (Top-q Weighted). At generation t, we form an elite subset $\mathcal{E}_t = \mathrm{Top}_{\lceil qN \rceil}(\mathcal{P}_t)$ with q fixed to 0.2. Parents are sampled from $\mathcal{E}_t$ with probability proportional to their fitness: $\Pr(x \mid \mathcal{E}_t) \propto f_T(x)$.

Mutation. Selected parent genomes are provided as the context of prompts to the LLM, which generates a set of offspring genomes $\mathcal{C}_t$ conditioned on the task and parent structure.

Population Pool Update. Generated offspring are deduplicated and merged into the population pool. If the pool size exceeds N, only the top-N genomes ranked by fitness are retained. The best-so-far fitness is updated as $f_t^\star = \max_{x \in \mathcal{P}_t} f_T(x)$.

3.1 Tasks & Genome Representations

Our evaluation includes tasks across four domains, spanning combinatorial, linguistic, symbolic, and algorithmic optimization where previous work has shown benefits from LLM-guided search.

Route Optimization The Traveling Salesman Problem (TSP) is a classical route optimization task. Prior work shows that while LLMs struggle to produce high-quality tours in a single pass (Fan et al., 2024), evolutionary search can largely improve performance (Huang et al., 2024). For each optimization run, one randomly generated distance matrix is given and the output must be a valid tour as a permutation of city indices. We evaluate two TSP variants with 30 and 60 cities, respectively. Each genome represents a permutation of the tour, and the fitness function is the inverse total distance.

Prompt Optimization It aims to automatically improve prompts quality. The genome is a textual instruction that conditions a frozen LLM (`gpt-4o-mini`). Evolutionary approaches have been shown effective in this setting (Guo et al., 2025; Fernando et al., 2023). We evaluate on dialogue summarization (SAMSum (Gliwa et al., 2019)) and text simplification (ASSET (Alva-Manchego et al., 2020)). Fitness is computed as the average generation quality on a held-out 25% validation subset, using ROUGE-L for SAMSum and SARI for ASSET, following Guo et al. (2025).

Equation Discovery Symbolic regression aims to discover concise mathematical expressions that fit observed input–output pairs (Grayeli et al., 2024). This domain is well-suited for LLM-guided evolutionary search which combines prior scientific knowledge with iterative refinement (Shojaee et al., 2025a,b). We adopt two nonlinear oscillation benchmarks from Shojaee et al. (2025a): Oscillator 1 (three variables) and Oscillator 2 (four variables). Each solution encodes a candidate symbolic expression executable as $f(x)$. Fitness is measured

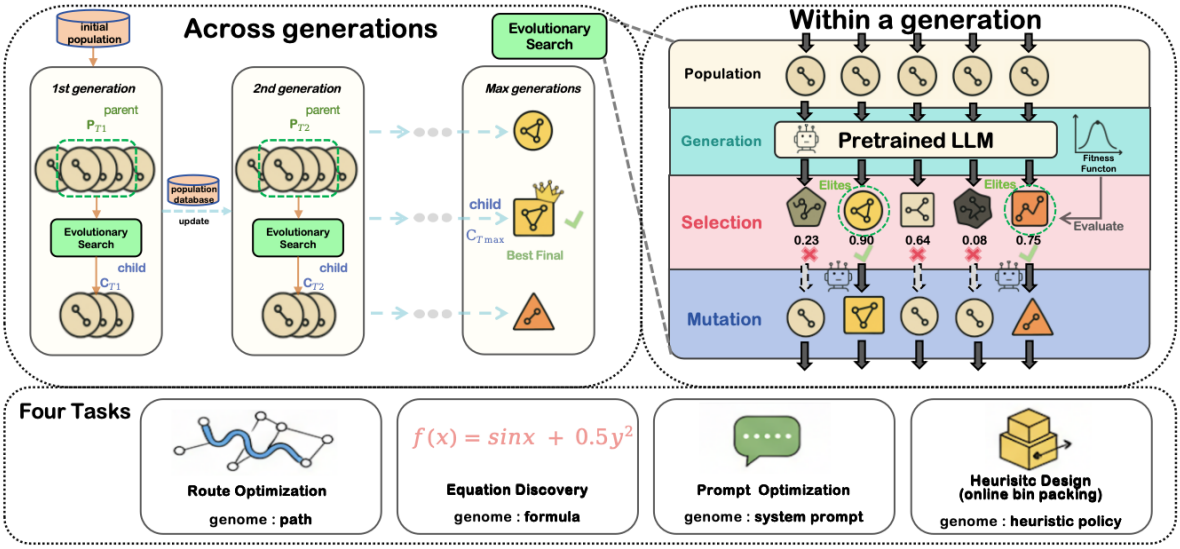


Figure 2: **Overview of the LLM-driven evolutionary search framework and tasks.** Left: the evolutionary process across generations. Right: the within-generation loop—population initialization, LLM-guided mutation, fitness evaluation, and selection. Bottom: the four tasks and their corresponding genome representations.

as $f_T(\text{expr}) = 1 - \text{norm}(\text{MSE}(\hat{y}, y))$, where $\hat{y}$ denotes model predictions, and norm is the min–max normalization computed over all candidate solutions for the same task instance.

Heuristic Design Heuristic design for combinatorial optimization aims to evolve executable programs rather than direct solutions. This paradigm has been successfully explored in recent work such as FunSearch and EoH (Romera-Paredes et al., 2023; Fei Liu, 2024). We focus on the online bin packing problem. Each genome encodes a heuristic policy in the form of a priority function that determines item placement. We evaluate on two datasets: OR3 (20 instances, 500 items each) and Weibull (5 instances, 5,000 items each), representing synthetic and real-world-like distributions. Fitness is the inverse number of bins used.

3.2 Novelty Computation

Task-agnostic novelty. Besides fitness, we further quantify semantic diversity along the trajectory. We defined the novelty of a solution a with respect to a set of all previous solutions $\mathcal{A}^{\text{prior}}$ including the initial parents under a task-specific semantic distance metric D_T: $nov(a, \mathcal{A}^{\text{prior}}) = \min_{b \in \mathcal{A}^{\text{prior}}} D_T(a, b)$. Novelty is normalized at subtask-level to ensure comparability.

Task-specific semantic distance. For TSP, we use an edge-set distance invariant to rotation and starting city. For prompt optimization, we compute cosine distance in a fixed embedding space using OpenAI's `text-embedding-ada-002`. For equation discovery and heuristic design, we adopt a functional behavior distance measured over a fixed input grid to capture divergence in output behavior.

3.3 Evolution Scale and Parameters

Our study involves 15 LLMs, with 30 generations conducted for each (model, task) pair. In each generation, the population produces 10 offspring, each corresponding to a model call. Every model–task pair is repeated twice with the same initial population, thereby totaling over 72,000 API calls. All evolutions are conducted using a default temperature of 0.7. The total cost of running experiments is estimated to be around $500.[1]

The selected LLMs span six model families: OpenAI (GPT-4o (OpenAI et al., 2024), GPT-4o-mini, GPT-3.5-turbo), Google's Gemini (Gemini-1.5-flash, Gemini-1.5-pro (Team et al., 2024)), Google's Gemma-3n-4b (Team et al., 2025), Meta's Llama (llama-3.1-70b-instruct, llama-3.1-8b-instruct, llama-3.2-1b-instruct, llama-3.2-3b-instruct (Grattafiori et al., 2024)), Deepseek AI's Deepseek-V3 (DeepSeek-AI et al., 2025), MistralAI's Mistral (Mistral-7b-instruct (Jiang et al., 2023), Mistral-24b-instruct, Mistral-large, Magistral-small (Mistral-AI et al., 2025)). Additional experimental details are provided in Appendix C.

[1] All trajectory data is available at https://huggingface.co/datasets/LivevreXH/evo_llm_trajectories.

4 Results and Analysis

The Optimization Gap. We evaluate each LLM under a fixed evolutionary budget and conditions. Performance is measured by the best fitness at the end of evolution. Across task families, we observe a pronounced **optimization gap** between models. Concretely, under identical conditions, different LLMs lead to different optimization outcomes (see Table 2). Strong early performance does not reliably predict long-horizon outcomes. For instance, Deepseek-V3 performs the best in the first generation yet fail to achieve the largest gains over time. This suggests that LLMs differ not only in solution quality, but in the search process. In the following section, we progressively rule out alternative explanations, i.e. base capability, novelty, and identify a consistent mechanism for successful optimization.

4.1 Base Model Capability

We first hypothesize that the gap may simply stem from base model capability, specifically the model's intrinsic task-specific problem-solving ability in zero-shot settings. Since all tasks are optimization-oriented, we define **zero-shot performance** as the best fitness achieved via temperature-swept *best-of-*N sampling: for each model–task pair, we generate candidates across six temperatures ($T \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$), with two samples per temperature, and report the best fitness among them (See Appendix C for more details).

As revealed in Figure 3, zero-shot performance is strongly correlated with post-optimization performance when aggregated across tasks. Similar trends are observed at the sub-task level (Appendix Figure 13), where higher zero-shot scores generally correspond to better final outcomes, only except for equation discovery tasks. This confirms that base capability is a strong predictor of optimization potential, yet insufficient to fully explain long-horizon optimization success. Models with nearly identical zero-shot performance can diverge substantially after evolution. For instance, around an average zero-shot score of 0.4 in Figure 3, multiple models cluster tightly along the zero-shot axis yet spread widely in their best final performance. This residual variance persists across tasks (See Figure 13).

4.2 Trajectory-Level Analysis

Since being a strong one-shot problem solver does not necessarily imply being an effective evolutionary search operator, the differences might lie in the search process. We therefore investigate trajectory-level properties of evolutionary search.

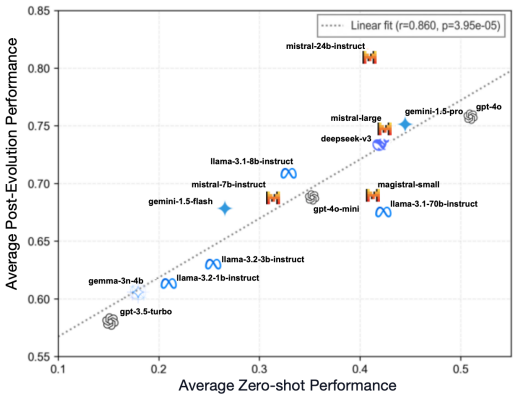


Figure 3: Scatter plot between zero-shot performance and final optimized performance across models.

4.2.1 Novelty vs. Breakthrough Dynamics

In classical evolutionary algorithms, novelty is treated as a proxy for exploration. However, this equivalence becomes problematic in LLM-guided evolution. Different from *blind* stochastic mutation operators, LLMs generate offspring by conditioning on parent solutions and task context aiming to produce the best solution. Novelty in this setting does not arise from random exploration, but from semantic variation within the LLM's output.

More Novelty Doesn't Yield Better Optimization. A natural hypothesis is therefore that models generating more novel solutions should explore the search space more effectively and achieve better optimization outcomes. We test this hypothesis in Figure 4, which summarizes both effect sizes and explanatory power across different trajectory-level descriptors. Astonishingly, novelty-based measures, including both average novelty and initial novelty (in the first generation), exhibit coefficients close to zero and are not statistically significant. Moreover, their explanatory power is negligible, meaning that increasing diversity alone does not contribute to improved optimization performance.

Breakthrough Rate Strongly Predicts Optimization Performance. We define a **breakthrough** as a best-so-far improvement event, e.g., an offspring generation in which the current solution exceeds the best fitness solution in all previous generations. We quantify each optimization trajectory's tendency to produce breakthroughs by its **breakthrough rate**. The breakthrough rate is also averaged per pair of models and tasks. As shown in

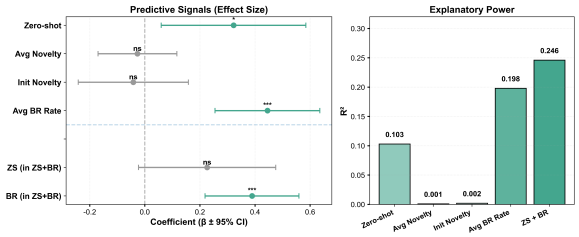


Figure 4: **OLS regression results across different trajectory descriptors.** (Left) Standardized coefficients. (Right) Explanatory power. $^{***}p < 0.001$, $^{*}p < 0.05$, ns means non-significant p-values. Novelty-based predictors are not significant, whereas breakthrough rate (BR) strongly predicts performance and improves fit beyond zero-shot capability (ZS).

Figure 4 (left), the breakthrough rate has the largest positive coefficient among all predictors. This is further reflected in its explanatory power (right), where breakthrough rate alone explains around two times more variance than zero-shot capability. Beyond this, when combining breakthrough rate with zero-shot performance, the overall explanatory power increases further, while the coefficient of zero-shot performance decreases. This indicates that part of the predictive power of base capability is mediated through the ability to generate consistent improvements during search.

These results show that good optimization trajectories tend to frequently produce small improvements instead of big and rare breakthroughs followed by long plateaus, as it often happens in evolutionary search (Mitchell and Taylor, 1999).

4.2.2 Semantic Geometry

In LLM-guided evolution, mutation operators are black-boxes, making it difficult to directly interpret how search progresses. To understand why some trajectories yield more breakthroughs than others, we instead examine the geometry of the search process by analyzing how solutions are distributed in semantic space over time.

We embed all candidate solutions into a task-specific shared semantic space, enabling us to analyze the within-generation distribution of candidates. Precisely, we measure the spatial organization of search using kernel-based entropy. Let $x_i \in \mathbb{R}^d$ denote the embedding of solution i, and $K(\cdot,\cdot)$ a Gaussian kernel. For any weighting w_j, we compute a local density estimate

$$g_i = \sum_j w_j K(x_i, x_j), \quad q_i = \frac{g_i}{\sum_k g_k},$$

and define

$$H = -\sum_i q_i \log q_i.$$

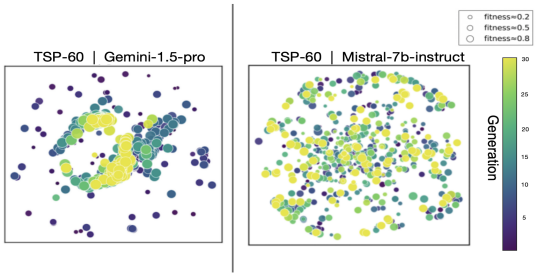
This framework yields two complementary views. Setting $w_j = 1$ gives **(i) spatial entropy (H_{spatial})**, which measures how broadly candidates spread across semantic space. Setting $w_j = f_j$ gives **(ii) fitness spatial entropy (H_{fitness})**, which measures whether high-quality solutions cluster or distribute across regions. These metrics intuitively summarize *how solutions are spatially organized* within a generation as they distinguish between diffuse versus localized search globally (H_{spatial}), and whether high-fitness solutions concentrates or spreads across the semantic landscape (H_{fitness}). We also complement them with multidimensional scaling visualizations that project all solutions onto a shared two-dimensional space, with points colored by generation and scaled by fitness (All MDS plots are on our website[2]). Figure 5 illustrates a representative case. Despite similar zero-shot performance and identical initial populations, Gemini-1.5-Pro progressively localizes its search into a smaller semantic region, while Mistral-7B-Instruct continues to drift across distant regions.

4.2.3 Generation-Level Statistical Test

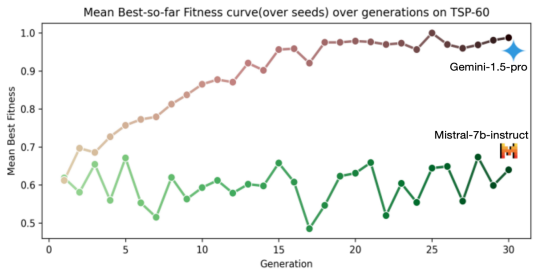
From the preceding case study in Figure 5, we hypothesize that the optimization success may specifically depend on the geometric properties of the population. We then use a generation-level mixed-effects regression analysis to examine which effects influence breakthrough events' production.

We model breakthrough probability at generation t as a function of population-level descriptors, including spatial entropy (H_{spatial}), fitness spatial entropy (H_{fitness}), mean and maximum novelty, generation index and their interaction for each generation. To account for repeated measurements and systematic differences across LLMs, we include model-specific random intercepts. Results are presented in Figure 6, reporting both concurrent effects (generation t) and lagged effects (predicting $t+1$). There are several consistent patterns. First, the generation index is strongly negative in both specifications, indicating that breakthroughs occur mostly in early generations. Second, higher fitness spatial entropy is negatively associated with breakthrough probability, which counter-intuitively

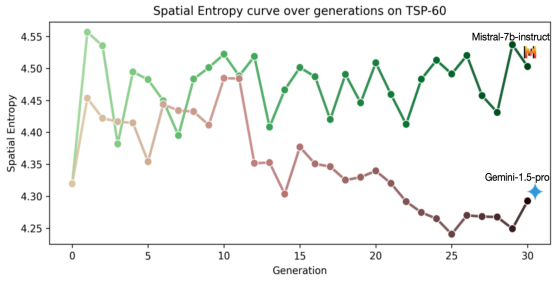
[2] https://xinhao-zhang.github.io/traj_evo_search/



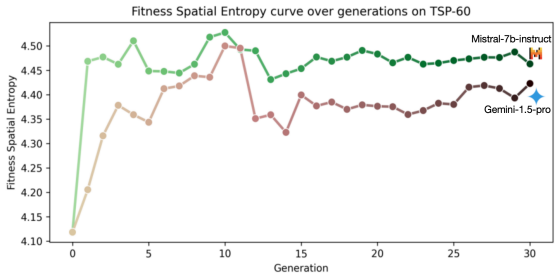
(a) Search Topology Visualization (MDS)



(b) Best Fitness Progression



(c) Spatial Entropy



(d) Fitness Spatial Entropy

Figure 5: **A qualitative contrast of evolutionary search geometry analysis.** (a) Visualization of the search space topology using MDS. Gemini-1.5-pro forms a convergent solution cluster (yellow). All points are projected using a shared MDS space learned from all task-specific candidates. (b) The Mean Best Fitness curve shows the convergence speed and quality over seeds. (c) Spatial Entropy quantifies the candidates' organization. (d) Fitness-Spatial Entropy illustrates Gemini's solutions are high-quality and topologically concentrated.

suggests that maintaining multiple dispersed high-quality regions would hinder breakthrough production. Third, although mean novelty is positively associated with breakthroughs within a generation, this effect is strongly conditioned on population geometry: the interaction between novelty and spatial entropy is significantly negative in both concurrent and lagged analyses. In other words, **novelty increases the likelihood of breakthroughs only when search remains sufficiently localized**. Crucially, while the effect of novelty fades under lagged prediction, the interaction effect remains significant, indicating that the productivity of novelty depends on the geometric state of the population rather than on contemporaneous correlations alone. Figure 12 further visualizes this interaction. Breakthrough events concentrate in regions featured by high novelty and low spatial entropy, whereas high novelty under high dispersion is usually related with low breakthrough probability.

4.3 Operator-Level Validation

The trajectory-level analyses above characterize *what* successful optimization runs look like: strong models progressively localize in semantic space and generate sustained best-so-far improvements, whereas weaker models exhibit semantic drift and

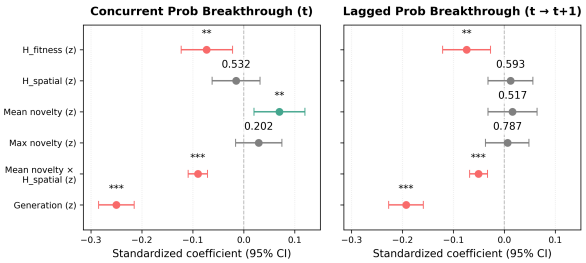


Figure 6: **Generation-level mixed-effects regression of breakthrough probabilities.** Standardized coefficients are shown for concurrent (left) and lagged (right) models, with predictors at generation t explaining breakthroughs at t or $t{+}1$. $^{***}p < 0.001$, $^{**}p < 0.01$, numeric labels report non-significant p-values.

stagnation. However, at operator-level, this suggests that beyond base capability, effective LLM optimizers behave as **local refiners**: they frequently produce offspring that strictly improve upon their prompted parents while maintaining a controlled semantic step sizes. We then validate this hypothesis through two studies below.

4.3.1 Model-Level Regression

We first conduct a fine-grained regression at model level. We employ two operator-level metrics defined at the parent→child mutation step. First,

	ZS + PCD	ZS + LRR + PCD
Zero-shot Perf. (z)	0.233^{*}	0.144
	(0.028)	(0.112)
Avg. Parent–Child Distance (z)	-0.329^{**}	-0.024
	(0.001)	(0.838)
Avg. Local Refinement Rate (z)		0.528^{***}
		(<0.001)
R^2	0.204	0.367

Table 1: Model–task OLS regressions predicting best final performance (z-score), with task fixed effects and model-clustered standard errors. p-values in parentheses. $^{***}p < 0.001$, $^{**}p < 0.01$,$^{*}p < 0.05$. Cells are shaded by coefficient magnitude.

the **local refinement rate (LRR)** represents the frequency of strict improvements of the offspring over prompted parents at the fraction of valid offspring attempts. Second, the **parent–child distance (PCD)** quantifies the average semantic distance between each offspring and its prompted parents in the same task-specific semantic space.

Table 1 reveals that when considered alone, larger semantic step sizes (PCD) are negatively correlated with final performance. However, this effect vanishes once LRR is included, while LRR remains strongly positive and highly significant. This is also consistent with our interaction finding: larger edits tend to reduce the probability of producing refinements. In other words, the negative effect of large semantic modifications is largely explained by their impact on refinement reliability.

This operator-level regression once again highlights that good LLM optimizers act as **local refiners**, where performance is governed by the ability to produce reliable incremental improvements rather than by the magnitude of semantic variation.

4.3.2 Perturbation Study: Model Mixing

To provide interventional evidence of the role of local refinement behavior, we further perform a perturbation study through model mixing experiments.

At each generation, a fraction of offspring are generated by an alternative model (weak refiner), while the remaining offspring are produced by the primary model (strong refiner). This intervention directly manipulate the refinement behavior of the search process. We construct task-specific model pairs with comparable zero-shot performance but contrasting refinement capabilities (see Appendix Figure 10). We evaluate this intervention across three representative sub-tasks across different regimes: TSP-60, bin packing-OR3, and Prompt Optimization-Summarization.

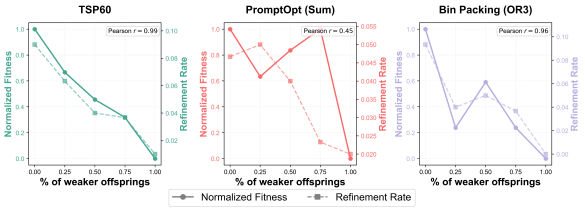


Figure 7: **Effect of model mixing on optimization performance and refinement rate.** A fraction of offspring is generated by a weaker refiner. Solid lines denote fitness; dashed lines denote refinement rate.

As shown in Figure 7, as the proportion of weak-refiner offspring increases, in particular, on TSP-60 and bin packing tasks, performance degrades sharply and monotonically. The same effect exists yet appears to be a bit weaker and less consistent in prompt optimization. Moreover, higher weaker offspring's ratios consistently reduce the overall refinement rate, which co-varies with the observed degradation in performance. These results suggest that weakening refinement behavior, by injecting lower-refinement operators, could impair the system's ability to produce sustained improvements, therefore leading to worse optimization outcomes.

4.4 Cost-Efficiency Implication

Our findings also carry practical implications for cost-sensitive deployment. Since optimization performance is not fully determined by base model capability, strong optimizers are not necessarily the most expensive models. We thus estimate the monetary cost of evolutionary optimization for each model based on the average number of input and output tokens per run, using API pricing in OpenRouter platform [3]. Optimization efficacy is measured as the fitness gain achieved over evolution. Figure 8 situates all models in a cost-improvement space, aggregated across all tasks (See Figure 15).

This aggregated view reveals large variation in cost–performance trade-offs. Notably, some mid-sized models achieve large fitness improvements at relatively low cost, whereas stronger zero-shot models do not always yield proportional gains per dollar. For example, Mistral-24B-Instruct lies on the Pareto frontier, combining large fitness improvement with moderate cost, and thus represents an efficient optimization operator rather than merely a strong base model. Overall, this result reinforces our central claim: effective evolutionary optimization depends more on how a model

[3] https://openrouter.ai

refines solutions over time than on its raw problem-solving capability. For practitioners, this can help build cost-efficient evolutionary systems by selecting models with favorable optimization behavior, instead of defaulting to the most powerful LLM.

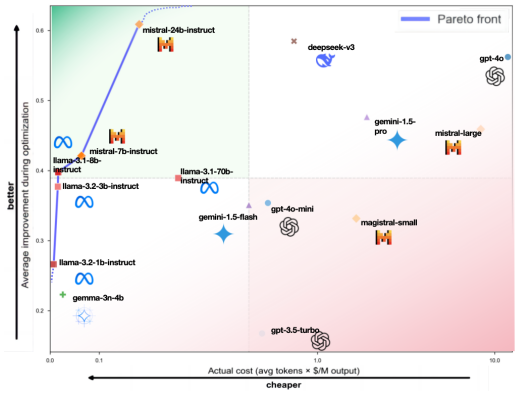


Figure 8: **Optimization gain versus cost across LLMs.** Each point represents a model, plotting average fitness improvement achieved through evolution against estimated monetary cost.

5 Discussion and Conclusion

In this work, we examined the exploration–exploitation trade-off in LLM-guided evolutionary search to understand why some models act as substantially better search operators than others. Although zero-shot task performance correlates with final optimization outcomes, it explains only part of the variance: models with similar zero-shot capability can induce markedly different optimization trajectories and final fitness.

Compared to classical evolutionary algorithms relying on stochastic mutation/crossover and selection to balance exploration and exploitation, LLM-guided evolution alters this paradigm. The mutation operator is no longer random but instantiated by a learned generative prior that induces structured, semantically meaningful variations, thereby strongly biasing the search toward exploitation.

A natural hypothesis is that this lack of randomness makes novelty or diversity a bottleneck, such that increased exploration should improve performance. Our results contradict this view. Higher novelty is not systematically associated with better outcomes and often signals failure: ineffective operators drift across semantic space without refining promising solutions. In contrast, strong LLM operators behave as effective local refiners. Their trajectories progressively localize around high-performing regions, with LLMs producing frequent, incremental improvements. In this regime, novelty is beneficial only when deviations occur within already promising regions. Our perturbation experiments further validate that directly degrading refinement behavior through model mixing brings about drops in optimization performance. These results also offer a refined interpretation of the role of novelty in LLM-guided search. Rather than being a stochastic explorer, novelty acts as an immediate driver of exploratory breakthroughs, but more importantly its long-term utility depends on whether the search regime allows these deviations to be selectively retained and amplified.

However, the observed local refinement behavior should not be viewed as an inherent capability of the base model alone. Instead, it emerges as a property of the entire agentic system that generates offspring, including the model, the prompting strategy, and the decoding configuration. While changes in temperature affect both refinement rates and performance, the relationship between refinement behavior and performance remains stable across a range of settings (see Appendix D.1).

Our findings have direct implications for the design of LLM-based optimization systems. Rather than focusing solely on maximizing base model capability, our results suggest that the key objective is to *control and optimize refinement behavior*. First, stronger base models do not necessarily yield better search operators, smaller or cheaper LLMs can outperform larger ones when their inductive biases favor stable local refinement. This highlights the importance of model selection. Besides, refinement behavior can potentially be modulated via system-level design choices, including prompting and decoding hyperparameter. More broadly, our results support training or fine-tuning LLMs as effective operators (Brahmachary et al., 2024; Šurina et al., 2025), emphasizing local refinement and error correction rather than general-purpose capability. Understanding and shaping such operator-specific behaviors is a promising avenue for boosting LLM-guided optimization.

Finally, the geometric analysis framework we developed to study LLM-guided optimization trajectories is broadly applicable and can be repurposed to analyze other types of iterative search or agentic behaviors. For illustration, a rich, interactive collection of visualizations showing trajectories in semantic space across models and tasks can be found on our project website.

Limitations

Our study is subject to several limitations. First, while we conduct robustness analyses on decoding hyperparameters (notably temperature; see Appendix D.1), our experiments still rely on a fixed evolutionary protocol. Other design choices, such as selection pressure, offspring size, and alternative sampling strategies, may influence the balance between exploration and exploitation, and could further shape cross-model differences. Second, in our study novelty is primarily operationalized as nearest-neighbor distance. Broader comparisons to KNN/average-distance novelty and alternative diversity indices would help assess robustness. Third, although we include perturbation experiments via model mixing, the intervention is still hard to fully isolate local refinement in a strictly controlled manner. Replacing the model that generates offspring may also affect other latent and invisible characteristics (e.g., reasoning patterns or exploration tendencies), thus making it difficult to attribute all performance differences solely to local refinement.

Acknowledgments

This work was supported by ANR (grant ANR-22-CPJ2-0036-01). It was also partially supported by ANR through the MIAI "AI & Language" chair (ANR-23-IACL-0006).

References

Nikhil Abhyankar, Sanchit Kabra, Saaketh Desai, and Chandan K. Reddy. 2026. Llema: Evolutionary search with llms for multi-objective materials discovery. *Preprint*, arXiv:2510.22503.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Fernando Alva-Manchego, Louis Martin, Antoine Bordes, Carolina Scarton, Benoît Sagot, and Lucia Specia. 2020. ASSET: A dataset for tuning and evaluation of sentence simplification models with multiple rewriting transformations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4668–4679, Online. Association for Computational Linguistics.

Huan ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, and 8 others. 2026. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Preprint*, arXiv:2507.21046.

Shuvayan Brahmachary, Subodh M. Joshi, Aniruddha Panda, Kaushik Koneripalli, Arun Kumar Sagotra, Harshil Patel, Ankush Sharma, Ameya D. Jagtap, and Kaushic Kalyanaraman. 2024. Large language model-based evolutionary optimizer: Reasoning with elitism. *Preprint*, arXiv:2403.02054.

Xiangsen Chen, Ruilong Wu, Yanyan Lan, Ting Ma, and Yang Liu. 2026. Molevolve: Llm-guided evolutionary search for interpretable molecular optimization. *Preprint*, arXiv:2603.24382.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Alex Duchnowski, Ellie Pavlick, and Alexander Koller. 2025. A knapsack by any other name: Presentation impacts LLM performance on NP-hard problems. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 6628–6651, Suzhou, China. Association for Computational Linguistics.

Jordan S. Ellenberg, Cristofero S. Fraser-Taliente, Thomas R. Harvey, Karan Srivastava, and Andrew V. Sutherland. 2025. Generative modeling for mathematical discovery. *Preprint*, arXiv:2503.11061.

Lizhou Fan, Wenyue Hua, Lingyao Li, Haoyang Ling, and Yongfeng Zhang. 2024. NPHardEval: Dynamic benchmark on reasoning ability of large language models via complexity classes. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4092–4114, Bangkok, Thailand. Association for Computational Linguistics.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. 2025. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *Preprint*, arXiv:2508.07407.

Mingxuan Yuan Xi Lin Fu Luo Zhenkun Wang Zhichao Lu Qingfu Zhang Fei Liu, Xialiang Tong. 2024. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning (ICML)*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *Preprint*, arXiv:2309.16797.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, and 15 others. 2025. Towards an ai co-scientist. *Preprint*, arXiv:2502.18864.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2025. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *Preprint*, arXiv:2309.08532.

John H. Holland. 1992. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.

Beichen Huang, Xingyu Wu, Yu Zhou, Jibin Wu, Liang Feng, Ran Cheng, and Kay Chen Tan. 2024. Exploring the true potential: Evaluating the black-box optimization capability of large language models. *Preprint*, arXiv:2404.06290.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Martin Josifoski, Lars Klein, Maxime Peyrard, Nicolas Baldwin, Yifei Li, Saibo Geng, Julian Paul Schnitzler, Yuxing Yao, Jiheng Wei, Debjit Paul, and 1 others. 2023. Flows: Building blocks of reasoning and collaborating ai. *arXiv preprint arXiv:2308.01285*.

Hyomin Lee, Sangwoo Park, Yumin Choi, Sohyun An, Seanie Lee, and Sung Ju Hwang. 2026. T-map: Red-teaming llm agents with trajectory-aware evolutionary search. *Preprint*, arXiv:2603.22341.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. 2022. Evolution through large models. *Preprint*, arXiv:2206.08896.

Xiaozhe Li, Jixuan Chen, Xinyu Fang, Shengyuan Ding, Haodong Duan, Qingwen Liu, and Kai Chen. 2025. Opt-bench: Evaluating llm agent on large-scale search spaces optimization problems. *Preprint*, arXiv:2506.10764.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. 2025. Seagent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *Preprint*, arXiv:2508.02085.

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

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. 2024. The ai scientist: Towards fully automated open-ended scientific discovery. *Preprint*, arXiv:2408.06292.

Robert MacKnight, Jose Emilio Regio, Jeffrey G. Ethier, Luke A. Baldwin, and Gabe Gomes. 2025. Pre-trained knowledge elevates large language models beyond traditional chemical reaction optimizers. *Preprint*, arXiv:2509.00103.

Mistral-AI, :, Abhinav Rastogi, Albert Q. Jiang, Andy Lo, Gabrielle Berrada, Guillaume Lample, Jason Rute, Joep Barmentlo, Karmesh Yadav, Kartik Khandelwal, Khyathi Raghavi Chandu, Léonard Blier, Lucile Saulnier, Matthieu Dinot, Maxime Darrin, Neha Gupta, Roman Soletskyi, Sagar Vaze, and 82 others. 2025. Magistral. *Preprint*, arXiv:2506.10910.

Melanie Mitchell and Charles E Taylor. 1999. Evolutionary computation: an overview. *Annual Review of Ecology and Systematics*, pages 593–616.

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

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb,

Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Anne Ouyang, Simon Guo, Simran Arora, Alex L. Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. 2025. Kernelbench: Can llms write efficient gpu kernels? *Preprint*, arXiv:2502.10517.

Maxime Peyrard, Martin Josifoski, and Robert West. 2025. Agentic ai: The era of semantic decoding. *Preprint*, arXiv:2403.14562.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, Jiacheng Zhu, Xuan Jiang, Sirui Li, Cathy Wu, Bryan Kian Hsiang Low, Jinhua Zhao, and Paul Pu Liang. 2026. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *Preprint*, arXiv:2604.01658.

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

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. 2025a. Llm-sr: Scientific equation discovery via programming with large language models. *Preprint*, arXiv:2404.18400.

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

Jianyong Sun, Hu Zhang, Qingfu Zhang, and Huanhuan Chen. 2018. Balancing exploration and exploitation in multiobjective evolutionary optimization. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 199–200.

Anja Šurina, Amin Mansouri, Lars C.P.M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. 2025. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Second Conference on Language Modeling*.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *Preprint*, arXiv:2404.14387.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, Soroosh Mariooryad, Yifan Ding, Xinyang Geng, Fred Alcober, Roy Frostig, Mark Omernick, Lexi Walker, Cosmin Paduraru, Christina Sorokin, and 1118 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Niki van Stein, Haoran Yin, Anna V. Kononova, Thomas Bäck, and Gabriela Ochoa. 2025. Behaviour space analysis of llm-driven meta-heuristic discovery. *Preprint*, arXiv:2507.03605.

Xingyu Wu, Sheng-Hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. 2025. Evolutionary computation in the era of large language model: Survey and roadmap. *IEEE Transactions on Evolutionary Computation*, 29(2):534–554.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Xianliang Yang, Ling Zhang, Haolong Qian, Lei Song, and Jiang Bian. 2025a. Heuragenix: Leveraging llms for solving complex combinatorial optimization challenges. *Preprint*, arXiv:2506.15196.

Zonglin Yang, Wanhao Liu, Ben Gao, Tong Xie, Yuqiang Li, Wanli Ouyang, Soujanya Poria, Erik Cambria, and Dongzhan Zhou. 2025b. Moosechem: Large language models for rediscovering unseen chemistry scientific hypotheses. *Preprint*, arXiv:2410.07076.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, volume 37, pages 43571–43608. Curran Associates, Inc.

Shaohua Yu, Tianyu Chen, and Linyan Liu. 2026. Large language model-driven full-component evolution of adaptive large neighborhood search. *Preprint*, arXiv:2603.06996.

Leyi Zhao, Weijie Huang, Yitong Guo, Jiang Bian, Chenghong Wang, and Xuhong Zhang. 2026. Large language model-powered evolutionary code optimization on a phylogenetic tree. *Preprint*, arXiv:2601.14523.

Zhikai Zhao, Chuanbo Hua, Federico Berto, Kanghoon Lee, Zihan Ma, Jiachen Li, and Jinkyoo Park. 2025. Trajevo: Designing trajectory prediction heuristics via llm-driven evolution. *Preprint*, arXiv:2505.04480.

Yangqiaoyu Zhou, Haokun Liu, Tejes Srivastava, Hongyuan Mei, and Chenhao Tan. 2024. Hypothesis generation with large language models. In *Proceedings of the 1st Workshop on NLP for Science (NLP4Science)*, page 117–139. Association for Computational Linguistics.

Yongxin Zhou, Changshun Wu, Philippe Mulhem, Didier Schwab, and Maxime Peyrard. 2026. What matters to an LLM? behavioral and computational evidences from summarization. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 5712–5737, Rabat, Morocco. Association for Computational Linguistics.

A Complete Experimental Result

See Table 2.

B Use of AI Assistant

AI tools were used to assist in writing, editing, and code development. The authors provided all content, ideas, and decisions, and the AI was used solely to improve clarity, readability, and efficiency.

C Task-Specific Experimental Details

Here are task-family-specific details of (i) EA parameters, (ii) genome validity checks, (iii) fitness evaluation, (iv) novelty distance, (v) population initialization, and (vi) prompts used for zero-shot and evolutionary search. Unless stated otherwise, the global evolutionary loop (selection–mutation–evaluation–pool update) follows Section 3.

C.1 Route Optimization

EA Parameters: For both TSP-30 and TSP-60, we use: $n_{\text{init}} = 40$ (initial population size), $q = 0.2$ (elite fraction), $p_{\text{parent}} = 3$ (parents sampled per generation), $p_{\text{child}} = 10$ (offspring per generation), $N = 40$ (capacity-limited pool), $G = 30$ (generations), and seed $= 21$.

Genome and Validity: A genome is a permutation $\pi \in S_n$ where $n \in \{30, 60\}$ is the number of cities. The LLM generates a genome as a JSON array of integers. Invalid genomes (non-permutations, unparsable outputs) receive fitness $f = 0$ and are excluded from parent sampling.

Fitness Evaluation: Given a distance matrix $\text{DIST} \in \mathbb{R}^{n \times n}$, the tour length is $L(\pi) = \sum_{i=1}^{n-1} \text{DIST}_{\pi_i, \pi_{i+1}} + \text{DIST}_{\pi_n, \pi_1}$. Fitness is the inverted length: $f_{\text{TSP}}(\pi) = -L(\pi)$ (normalized post-hoc by task-level min/max).

Novelty Distance: We use edge-set distance after canonization:

$$D_{\text{edge}}(\pi, \sigma) = 1 - \frac{|E(\pi) \cap E(\sigma)|}{|E(\pi)|},$$

where $E(\pi)$ denotes the set of undirected edges in tour π. This metric captures structural differences regardless of rotation or starting city.

Population Initialization: The initial population $\mathcal{P}_0$ consists of $n_{\text{init}} = 40$ random permutations (using the same random seed across all models). Each permutation is obtained via `random.sample(range(n), n)`.

Zero-shot Prompt: The zero-shot prompt provides the complete distance matrix as JSON and asks the model to return an optimal tour. See Table 3.

Evolution Prompt: During evolution, the prompt provides the distance matrix and up to 3 parent tours with their scores (path lengths). The LLM is asked to generate a better child tour. See Table 4.

C.2 Prompt Optimization

EA Parameters: $n_{\text{init}} = 10$, $q = 0.2$, $p_{\text{parent}} = 2$, $p_{\text{child}} = 5$, $N = 10$ (capacity), $G = 30$, and two task variants: **SAMSum** (dialogue summarization) and **ASSET** (text simplification).

Genome and Validity: A genome is a natural language instruction prompt (string). Any non-empty string output is considered valid.

Fitness Evaluation: For each prompt genome p, we condition a frozen evaluator LLM (gpt-4o-mini) on p to generate outputs on a held-out 25% evaluation set. Fitness is the task-specific metric score (ROUGE-L for SAMSum, SARI for ASSET).

Novelty Distance: We use semantic cosine distance in a fixed embedding space (OpenAI's `text-embedding-ada-002`):

$$D_{\cos}(p_1, p_2) = 1 - \frac{E(p_1) \cdot E(p_2)}{\|E(p_1)\| \|E(p_2)\|},$$

where $E(\cdot)$ is the sentence embedding.

Population Initialization: Initial prompts are loaded from a curated set of baseline prompts for each task (SAMSum or ASSET). Duplicates are removed, and the first 10 unique prompts are selected.

LLM	Zero-Shot					First Generation					Last Generation				
					Avg					Avg					Avg
4o	**47.4**	51.9	**82.3**	31.5	**53.3**	37.2	41.3	75.7	31.7	46.5	85.4	70.9	77.7	75.4	77.4
1.5-Pro	47.3	43.0	70.4	30.7	47.8	43.0	38.3	84.0	32.4	49.4	**89.0**	72.4	85.5	58.5	76.4
V3	39.0	41.2	71.5	31.5	45.8	**50.8**	56.6	**87.2**	33.0	**56.9**	70.4	77.1	91.1	62.7	75.3
Large	19.5	49.1	79.7	31.5	45.0	34.5	56.6	74.1	33.0	49.5	58.4	84.7	78.7	81.1	75.7
3.1-70B-Instruct	15.9	55.2	75.2	31.5	44.5	34.0	45.1	69.9	33.0	45.5	59.6	69.5	78.0	69.8	69.2
Magistral-Small	29.0	49.9	66.6	31.5	44.3	34.0	61.8	70.5	31.7	49.5	68.0	73.7	75.4	64.4	70.4
24B-Instruct	11.6	**55.8**	72.2	31.5	42.8	34.0	**66.5**	70.2	33.1	51.0	75.0	84.3	75.2	**92.0**	**81.6**
4o-mini	13.9	38.3	70.1	31.5	38.4	34.0	55.8	66.7	31.7	47.1	60.2	71.2	82.9	66.2	70.1
3.1-8B-Instruct	2.9	55.7	48.7	**32.9**	35.1	34.3	44.5	67.4	31.7	44.5	63.8	73.7	77.1	74.5	72.2
7B-Instruct	18.5	46.9	49.7	23.6	34.7	35.1	47.7	67.7	31.7	45.5	46.9	73.7	**91.8**	67.7	70.0
1.5-Flash	6.6	47.5	60.9	3.4	29.6	34.0	49.8	73.5	31.7	47.2	57.1	**95.9**	75.1	44.6	68.2
3.2-3B-Instruct	22.2	36.3	28.4	29.8	29.1	34.6	47.7	66.7	32.3	45.3	55.1	70.6	85.9	47.5	64.8
3.2-1B-Instruct	14.8	47.8	0.0	31.5	23.5	34.0	53.1	69.4	31.7	47.0	54.1	68.5	80.9	48.5	63.0
3n-4B	0.0	55.6	0.0	22.9	19.6	35.6	53.2	66.7	31.7	46.8	42.7	80.4	67.8	52.2	60.8
3.5-turbo	16.5	27.2	0.0	28.1	18.0	34.0	40.5	75.6	**33.2**	45.8	49.0	65.8	81.4	41.0	59.3

Table 2: Fitness performance of LLMs on four task families (Route Optimization, Prompt Optimization, Equation Discovery, Heuristic Design). Cells report averaged normalized fitness across two sub-tasks and two seeds within the same family, making scores comparable over three settings. Those cells are background-shaded by a normalized improvement (darker = larger improvement) computed per `(model,task)` pair relative to their initial-population best value of each sub-task. Models are sorted in descending order of their average Zero-Shot score. Best scores per column are **bold**.

TSP Zero-shot Prompt

System: You are an optimization expert helping to solve a hard problem. You will be shown several candidate solutions with their scores. Your goal is to propose better solutions.

User: TASK DESC: The traveling salesman problem (TSP) aims to find the shortest route visiting all cities exactly once. You must return a valid tour as a list of city indices.

QUESTION: [Distance matrix provided as JSON]

Please return the optimal solution as JSON without extra explanation: `{ "genome": "[list of n unique integers 0 to n-1]" }`.

Table 3: TSP Zero-shot Prompt Template

Zero-shot Prompt: See Table 5.

Evolution Prompt: See Table 6.

C.3 Equation Discovery

EA Parameters: $n_{\text{init}} = 7$, $q = 0.2$, $p_{\text{parent}} = 2$, $p_{\text{child}} = 10$, $N = 40$ (capacity), $G = 30$, and seed $= 21$. We evaluate on two nonlinear oscillator datasets with different dimensionalities (Oscillator-1: 3 variables; Oscillator-2: 4 variables with time).

Genome and Validity: A genome is a Python function string defining $a = f(x, v)$ (Oscillator-1) or $a = f(t, x, v)$ (Oscillator-2). Genomes are validated by attempting to parse and execute them; non-executable or divergent outputs receive fitness $f = 1 \times 10^6$ (high loss).

Fitness Evaluation: Given training data (X, y_{true}), fitness is computed as:

$$f_{\text{SymReg}} = 1 - \text{norm}\left(\text{MSE}\left(y_{\text{pred}}, y_{\text{true}}\right)\right),$$

where normalization is per-task instance.

Novelty Distance: We use functional behavior distance over a fixed input grid:

$$D_{\text{sem}}(f, g) = 1 - \frac{1}{m} \sum_{j=1}^{m} \cos\left(f(x_j), g(x_j)\right),$$

where x_j are uniformly sampled input points and $\cos(\cdot)$ is cosine similarity. This captures semantic divergence in output behavior.

TSP Evolution Prompt

System: You are an optimization expert helping to solve a hard problem. You will be shown several candidate solutions with their scores. Your goal is to propose better solutions.

User: TASK DESC: The traveling salesman problem (TSP) aims to find the shortest route visiting all cities exactly once.

QUESTION: [Distance matrix provided as JSON]

Here are [num_parents] previous solutions and their scores (lower is better):

```
{"genome": [parent_1], "score": 1234.5}
{"genome": [parent_2], "score": 1256.3}
...
```

Please return one BETTER child genome as JSON: `{ "genome": "[full-new]" }`. The genome must be a list of [n] unique integers from 0 to [n-1].

Table 4: TSP Evolution Prompt Template

Prompt Optimization Zero-shot Prompt

System: You are a prompt optimization expert. Your goal is to design effective prompts for language models on text summarization/simplification tasks.

User: TASK DESC: Your task is to create a high-quality instruction prompt that will guide a language model to perform [summarization/simplification] with high fidelity to the original content.

Please return ONLY the prompt text (plain text, not JSON or wrapped in quotes) that will be used to instruct the language model.

Example expected output:

```
Summarize the following dialogue in a concise manner...
```

Table 5: Prompt Optimization Zero-shot Prompt Template

Population Initialization: Initial genomes are randomly sampled expressions combining input variables, constants, and allowed functions (e.g., `np.sin`, `np.cos`, `np.exp`). We maintain the same seed for all models. All initial genomes are evaluated on the training set.

Zero-shot Prompt: See Table 7.

Evolution Prompt: See Table 8.

C.4 Heuristic Design

EA Parameters: For bin packing heuristic design, we use: $n_{\text{init}} = 7$, $q = 0.2$, $p_{\text{parent}} = 2$, $p_{\text{child}} = 10$, $N = 40$ (capacity), $G = 30$, and seed $= 42$.

Genome and Validity: A genome is a Python function string implementing a priority heuristic `def priority(item, bins): ...`. The function takes item size and bin residual capacities as input, and returns priority scores. Invalid or non-executable functions receive fitness $f = 1 \times 10^6$.

Fitness Evaluation: For each genome, we run online bin packing on **all** instances of the active dataset (OR3 or Weibull) using the heuristic function. Fitness is the inverted number of bins used, averaged over all instances:

$$f_{\text{BinPack}} = -\frac{1}{|\text{instances}|} \sum_i \text{bins\_used}(i).$$

Novelty Distance: We measure behavioral/strategy distance as well. We define a set of fixed "probe scenarios" (random combinations of item sizes and bin capacities), compute the priority/score vectors for each candidate function under these scenarios, and measure distance via:

$$D_{\text{behav}}(h_1, h_2) = 1 - \frac{1}{K} \sum_{k=1}^{K} \cos\left(\mathbf{s}_{h_1}^{(k)}, \mathbf{s}_{h_2}^{(k)}\right),$$

where $\mathbf{s}_h^{(k)}$ is the priority vector returned by function h on scenario k. This metric captures functional divergence in heuristic strategy independent of implementation style. If a function returns probability distributions, distribution distance (Jensen-Shannon or Earth Mover Distance) can alternatively be used.

Prompt Optimization Evolution Prompt

System: You are a prompt optimization expert. Your goal is to design effective prompts for language models.
User: TASK DESC: Create a better instruction prompt for [summarization/simplification].
Here are previous prompts and their performance scores (higher fitness = better):

```
Prompt: "Summarize the following..."
Fitness: 0.1234
Prompt: "Create a brief summary..."
Fitness: 0.1156
```

Analyze the patterns in successful prompts. Combine successful elements and create a new, improved prompt. Return ONLY the plain text prompt without explanation.

Table 6: Prompt Optimization Evolution Prompt Template

Symbolic Regression Zero-shot Prompt

System: You are a scientific equation discovery expert. Your goal is to propose symbolic expressions that fit the data well.
User: TASK DESC: This is a symbolic regression task for a damped nonlinear oscillator. Given (x, v) and acceleration a, find a mathematical expression $a = f(x, v)$ that fits the data as accurately as possible.
QUESTION: Here are some data points [100 samples as JSON]:

```
[{"x": 0.123, "v": 0.456, "a": 0.789}, ...]
```

Please return a Python function as a code block without explanation:

```
def equation(x, v):
    return 1.2*x + 0.8*v + np.sin(x)
```

Table 7: Symbolic Regression Zero-shot Prompt Template

Population Initialization: Initial heuristics are sampled from a set of canonical bin packing rules (e.g., best-fit, worst-fit, first-fit, combinations thereof). More details can be found in our code repository. Each is evaluated on all instances.

Zero-shot Prompt: See Table 9.

Evolution Prompt: See Table 10.

C.4.1 Zero-shot Evaluation Details

For each model–task pair, we sampled outputs under six temperature settings ($T \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$), with two runs per temperature.

C.4.2 Task-agnostic Novelty Computation

Algorithm 1 details the task-agnostic novelty computation procedure used across all experiments. Novelty is defined as the minimum semantic distance to prior candidates within the same problem instance and normalized to ensure comparability across generations.

Algorithm 1: Task-agnostic novelty computation for each genome

Require: task T, problem instance p, generation g, candidate $a_{p,g}$, prior set $\mathcal{A}_{p,g}^{\text{prior}}$, task-specific distance metric D_T, all diversity values $\mathcal{N}_p$ for instance p, constant ε
Distance Computation: Compute raw point diversity (nearest neighbor): $n \leftarrow \min_{b \in \mathcal{A}_{p,g}^{\text{prior}}} D_T(a_{p,g}, b)$
Normalization: Normalize within p: $\widehat{n}(a_{p,g}) \leftarrow \frac{n - \min(\mathcal{N}_p)}{\max(\mathcal{N}_p) - \min(\mathcal{N}_p) + \varepsilon}$
Return normalized diversity $\widehat{n}(a_{p,g}) \in [0, 1]$

C.5 Multi-dimensional Scaling Parameters

We use Multi-dimensional Scaling (MDS) to project task-specific novelty distances into 2D embeddings ("trajectory landscapes") for visualization. The approach is standardized across all four task families, with variations in the distance metric:

Sampling and Fitting Strategy: For each task, we collect all genomes across generations and models. To avoid computational bottlenecks:

1. If total genomes $N \leq 4000$: fit MDS on the

Symbolic Regression Evolution Prompt

System: You are a scientific equation discovery expert. Your goal is to propose a new, better mathematical expression that fits the data (lower MSE is better).

User: TASK DESC: Symbolic regression for damped nonlinear oscillator. Find $a = f(x, v)$.

Here are previous candidate expressions and their MSE scores:

```
{"code": "def equation(x, v): return x + v", "mse_score": 0.1234}
{"code": "def equation(x, v): return np.sin(x) + v**2", "mse_score": 0.0987}
```

Please return a new, better Python function as a code block without explanation. The function should take x, v as input and may use mathematical operations and numpy functions.

Table 8: Symbolic Regression Evolution Prompt Template

Bin Packing Zero-shot Prompt

System: You are an expert in online bin packing algorithms. Your goal is to design priority functions that minimize the number of bins needed.

User: TASK DESC: Online bin packing: given items of varying sizes and bins of fixed capacity, design a priority heuristic that decides where to place each incoming item.

Your priority function will be called as:

```
priority(item, bins)
```

Where:

- `item`: float, the size of current item to pack
- `bins`: numpy array, remaining capacities of bins that can fit the item

Return: numpy array of same length as `bins`, with priority scores for each bin.

Example:

```
def priority(item, bins):
    return -(bins - item)  # Best-fit heuristic
```

Please provide a function better than the example! Return ONLY the code block.

Table 9: Bin Packing Zero-shot Prompt Template

entire distance matrix.

2. If $N > 4000$: use stratified sampling (max 60 genomes per (model, generation) bucket) to obtain $m \leq 4000$ base points, then use out-of-sample (OOS) placement for remaining points.

MDS Solver Parameters: All experiments use `sklearn.manifold.MDS` with:

- `n_components=2`: Project to 2D for visualization.
- `dissimilarity="precomputed"`: Input is precomputed distance matrix.
- `n_init=1`: Single initialization (fixed random seed ensures reproducibility).
- `max_iter=300`: Maximum solver iterations.
- `eps=1e-3`: Convergence tolerance.
- `random_state=42`: Deterministic seed.

Out-of-Sample Placement: For genomes not in the base fit set, we use k-NN Shepard interpolation in the 2D space:

1. Compute distance (using task-specific metric) from each OOS point to all m base points.
2. Find $k = 8$ nearest neighbors (smallest distances).
3. Assign weights $w_i = 1/(d_i + 10^{-8})^p$, where $p = 2.0$ and d_i is distance to neighbor i.
4. Place OOS point at weighted average of neighbor 2D coordinates.

This approach is fast (vectorized per block of 4000 points) and preserves neighborhood structure in the high-dimensional space.

Bin Packing Evolution Prompt

System: You are an expert in online bin packing algorithms. Your goal is to design a new, better priority function that minimizes bins used.

User: TASK DESC: Online bin packing heuristic design. Minimize the number of bins used across **all [dataset] instances**.

Here are previous priority functions and their performance (average bins used):

```
{"code": "def priority(item, bins): return -(bins - item)", "avg_bins": 45.3}
{"code": "def priority(item, bins): return bins**2 / (item + 1e-6)", "avg_bins": 42.1}
```

Analyze successful patterns and create a new, better heuristic. The function signature must be:

```
def priority(item, bins):
    import numpy as np
    # Your heuristic here
    return priority_scores
```

Return ONLY the code block without explanation. Try to achieve fewer bins than the parents above!

Table 10: Bin Packing Evolution Prompt Template

Prompt Optimization Preprocessing: For prompt optimization, embeddings are high-dimensional ($\sim$ 1536 dimensions for text-embedding-ada-002). To accelerate MDS on large datasets ($> 10K$ rows), we first applied PyGlimmerMDS[4] (a GPU-accelerated multilevel MDS variant) on a server, then loaded precomputed 2D coordinates for visualization.

Normalization and Scaling: Fitness values are normalized per task using robust min-max scaling (1st and 99th percentiles), ensuring visual comparability across tasks with different fitness ranges. Generation and fitness are displayed via (i) color (viridis colormap) and (ii) point size, respectively.

D Robustness Analyses

D.1 Temperature-Sensitivity Experiment

A potential concern is that our findings may depend on specific decoding hyperparameters as temperature directly affects the stochasticity of LLM-generated mutations. We also conduct a temperature-sensitivity analysis on two representative task families—route optimization (TSP) and equation discovery (Oscillator)—using two models with contrasting refinement capabilities (Mistral-7B and Mistral-24B). We vary the decoding temperature over a wide range $T \in \{0.0, 0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.3\}$, and for each configuration measure both the local refinement rate and the final optimization performance.

Overall, these additional results in Table 11 and Figure 9 that our main finding—that local refinement ability is a key driver of optimization success—is robust to substantial variations in decoding temperature. Rather than being tied to a narrow hyperparameter regime, refinement behavior turns out to be a stable property of the combined system (model, prompt, and decoding configuration).

Task	**Model**	**Pearson r**	**p-value**
Oscillator	Mistral-7B	0.49	0.209
Oscillator	Mistral-24B	0.32	0.433
TSP	Mistral-7B	0.76^{*}	0.027
TSP	Mistral-24B	0.92^{***}	0.000

Table 11: Pearson correlation between local refinement rate and final performance under temperature variation. The relationship remains consistently positive across settings.

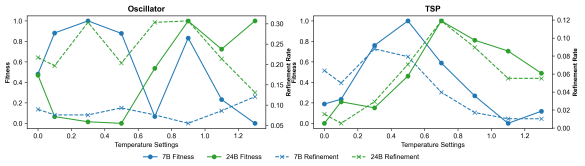


Figure 9: Temperature sensitivity of refinement–performance dynamics. While performance varies with temperature, the relationship between refinement and fitness remains stable, particularly on TSP where strong positive correlations are observed.

D.2 Perturbation Study Details

E Statistical Model Specifications

E.1 OLS Regressions

We employ ordinary least squares (OLS) regression with clustering-robust standard errors to evaluate

[4] https://github.com/hageldave/PyGlimmerMDS

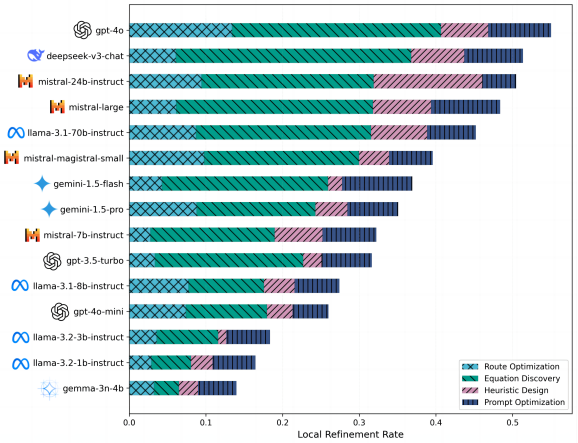


Figure 10: Average local refinement rate across models and task families. Mistral-24B exhibits consistently strong and balanced refinement behavior across tasks, aligning with its strong post-optimization performance..

Task	Good Refiner	Bad Refiner
TSP60	Mistral-24B	Mistral-7B
Summarization	DeepSeek-V3-Chat	GPT-4o-mini
Oscillator-1	GPT-3.5-Turbo	Gemma-3n-4B
Bin-Packing-OR3	Mistral-24B	LLaMA-3.2-3B-Instruct

Table 12: Models used in the mixing model experiments across tasks. We contrast strong refiners with weaker ones to analyze causal effects of refinement ability.

the predictive power of novelty and breakthrough-rate metrics on final evolutionary performance. All models include task fixed effects to account for within-task heterogeneity (8 tasks × 2 seeds = 16 task instances across 15 LLMs, $N = 119$ model-task pairs).

Data and Estimation:

- **Sample:** Aggregated at the model-task level ($N = 119$ observations: 15 models × 8 tasks, with missing cells for some model-task combinations).
- **Dependent variable:** best_final_perf$_z$: z-score-normalized best final generation fitness per (model, task) pair.
- **Covariates (all z-scored for interpretability)**:
  - avg_novelty$_z$: Mean within-generation novelty (average distance to nearest prior candidate).
  - initial_nov$_z$: Initial population novelty (diversity at generation 0).
  - avg_breakthrough_rate$_z$: Fraction of generations achieving best-so-far improvement.
  - zero_shot_perf$_z$: Average zero-shot performance under temperature-swept setting.
- **Errors:** Clustered by model (15 clusters) using Huber-White robust covariance estimator to account for within-model correlations across tasks.
- **Fixed effects:** 8 task indicators (baseline: TSP-30; reference category absorbed in intercept).

Model Specifications: We fit two sets of models to test distinct hypotheses:

Set A: Novelty as Predictor. Regression form: best_final_perf$_z$ $\sim$ predictor$_z$ + $\mathbf{1}_{\text{task}}$

M1 Predictor = avg_novelty$_z$ only; tests if exploration (novelty) predicts final performance.

M2 Predictor = initial_nov$_z$ only; tests if initial diversity is predictive.

M3 Predictor = zero_shot_perf$_z$ only; baseline model controlling for base capability.

M4 Predictors = zero_shot_perf$_z$ + avg_novelty$_z$; tests whether novelty adds explanatory power controlling for zero-shot ability.

M5 Predictors = zero_shot_perf$_z$ + initial_nov$_z$; alternative initial-diversity control.

Set B: Breakthrough-Rate as Predictor. Regression form: best_final_perf$_z$ $\sim$ predictor$_z$ + $\mathbf{1}_{\text{task}}$

M6 Predictor = avg_breakthrough_rate$_z$ only; tests if progress (breakthrough frequency) predicts success.

M7 Predictor = zero_shot_perf$_z$ only; baseline.

M8 Predictors = zero_shot_perf$_z$ + avg_breakthrough_rate$_z$; joint model.

Results Summary: See Table 13 and Table 14

Table 13: OLS Regression Results: Novelty Predictiveness (Set A)

Model	Predictor(s)	β	SE	p-value	R^2	Adj. R^2
M1	avg_novelty_z	-0.027	0.073	0.710	0.001	-0.072
M2	initial_nov_z	-0.042	0.102	0.681	0.002	-0.071
M3	zero_shot_perf_z	0.322^{*}	0.134	0.016	0.103	0.038
M4	ZS + Avg Novelty	ZS: 0.322^{*}	0.138	0.019	0.103	0.029
		Nov: 0.002	0.072	0.980		
M5	ZS + Init Novelty	ZS: 0.324^{*}	0.132	0.014	0.106	0.033
		Init: -0.055	0.063	0.387		

Table 14: OLS Regression Results: Breakthrough-Rate Predictiveness (Set B)

Model	Predictor(s)	β	SE	p-value	R^2	Adj. R^2
M6	avg_breakthrough_rate_z	0.445^{***}	0.097	< 0.001	0.198	0.139
M7	zero_shot_perf_z	0.322^{*}	0.134	0.016	0.103	0.038
M8	ZS + Breakthrough	ZS: 0.226	0.127	0.076	0.246	0.184
		BR: 0.389^{***}	0.087	< 0.001		

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$. All predictors are z-scored. Standard errors are clustered by model (robust to heteroskedasticity and within-model correlation). Task fixed effects included but not shown.

Implications: The weak explanatory power of novelty metrics (Set A) contrasts sharply with the strong predictiveness of breakthrough-rate metrics (Set B), supporting that models that generate diverse genomes do not necessarily improve faster. Instead, the ability to drive consistent fitness improvements (characterized by breakthrough frequency) is the key differentiator across LLMs.

E.2 Mixed-Effects Regression Models

We employ generalized linear mixed-effects models (GLMM) to analyze breakthrough probability at the generation level, accounting for model-level heterogeneity via random intercepts. We fit two specifications: (1) **Concurrent Model**: same-generation predictors, and (2) **Lagged Model**: current-generation predictors forecasting next-generation breakthrough probability.

Data and Estimation:

- **Concurrent sample:** $N = 3{,}570$ generation-level observations across 15 LLMs (groups), with variable group sizes (min 209, max 242, mean 238). Data from all 8 tasks × 30 generations × 15 models, subset to generations with complete novelty and entropy measurements.
- **Lagged sample:** $N = 3{,}451$ observations (omitting final generation of each model-task pair, which has no $t + 1$ outcome).
- **Dependent variable (concurrent):** $\text{prob\_breakthrough}_z$: z-scored fraction of offspring achieving best-so-far improvement in generation g (binary: 0/1 per generation, then aggregated).
- **Dependent variable (lagged):** $\text{prob\_breakthrough}_{z,\,t+1}$: next-generation breakthrough probability (lead variable).
- **Fixed effects (all z-scored for comparability)**:
  - $H_{\text{fitness},\,z}$: Fitness-weighted spatial entropy (concentration of high-fitness mass).
  - $H_{\text{spatial},\,z}$: Uniform-weighted spatial entropy (semantic dispersion).
  - $\text{mean\_novelty\_per\_gen}_z$: Average within-generation novelty.
  - $\text{max\_novelty\_per\_gen}_z$: Maximum novelty in generation.
  - $\text{mean\_novelty\_per\_gen}_z \times H_{\text{spatial},\,z}$: Interaction term capturing interference between exploration and dispersion.
  - generation_z: z-scored generation index (time control).
  - 8 task indicators (baseline: TSP-30; reference absorbed in intercept).
- **Random effects:** Model-level random intercept, $u_{\text{model}} \sim \mathcal{N}(0, \tau^2)$, allowing breakthrough propensity to vary across LLMs.
- **Estimation:** Maximum likelihood (ML, not

REML) to permit likelihood ratio testing between models.

Model Formulation: Concurrent Model. For generation g of model m on task t:

$$\begin{aligned}\text{prob\_breakthrough}_{g,m,t,z} =& \beta_0 + \\ & \beta_1 H_{\text{fitness},z} + \beta_2 H_{\text{spatial},z} + \\ & \beta_3 \overline{\text{nov}}_z + \beta_4 \text{max\_nov}_z + \\ & \beta_5 (\overline{\text{nov}}_z \times H_{\text{spatial},z}) \\ & + \gamma_t \mathbf{1}_t + \beta_6 \text{gen}_z + u_m + \epsilon_{g,m,t}. \end{aligned} \quad (1)$$

Lagged Model. Using generation g predictors to forecast generation $g+1$ outcomes:

$$\begin{aligned}\text{prob\_breakthrough}_{g+1,m,t,z} =& \beta_0^{\text{lag}} + \\ & \beta_1^{\text{lag}} H_{\text{fitness},z}(g) + \beta_2^{\text{lag}} H_{\text{spatial},z}(g) \\ & + \beta_3^{\text{lag}} \overline{\text{nov}}_z + \beta_4^{\text{lag}} \text{max\_nov}_z \\ & + \beta_5^{\text{lag}} (\overline{\text{nov}}_z(g) \times H_{\text{spatial},z}(g)) \\ & + \gamma_t^{\text{lag}} \mathbf{1}_t + \beta_6^{\text{lag}} \text{gen}_z(g) \\ & + u_m^{\text{lag}} + \epsilon_{g+1,m,t}. \end{aligned} \quad (2)$$

Results Summary: See Table 15

Temporal Dynamics: The lagged model reveals that current-generation state is a weak predictor of next-generation breakthroughs (residual variance 0.662 vs. concurrent 0.780, suggesting some temporal structure but substantial noise). The interaction term remains the strongest signal across both timescales, suggesting the interference effect is a robust mechanistic feature of LLM-guided evolution, not merely a concurrent correlation.

F Supplementary Visualizations

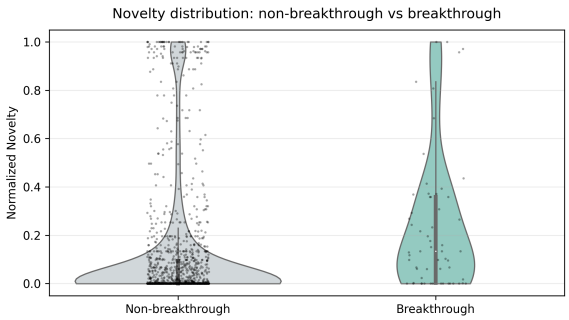


Figure 11: Breakthrough and Non Breakthrough's distribution for all collected trajectories

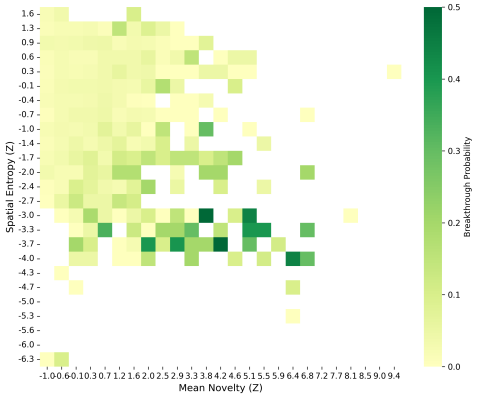


Figure 12: **Interaction between novelty and spatial entropy in breakthrough dynamics.** Each cell reports the empirical breakthrough probability aggregated over generations falling into the corresponding bins of mean novelty and spatial entropy(z-scored). Color intensity indicates higher likelihood of breakthroughs.

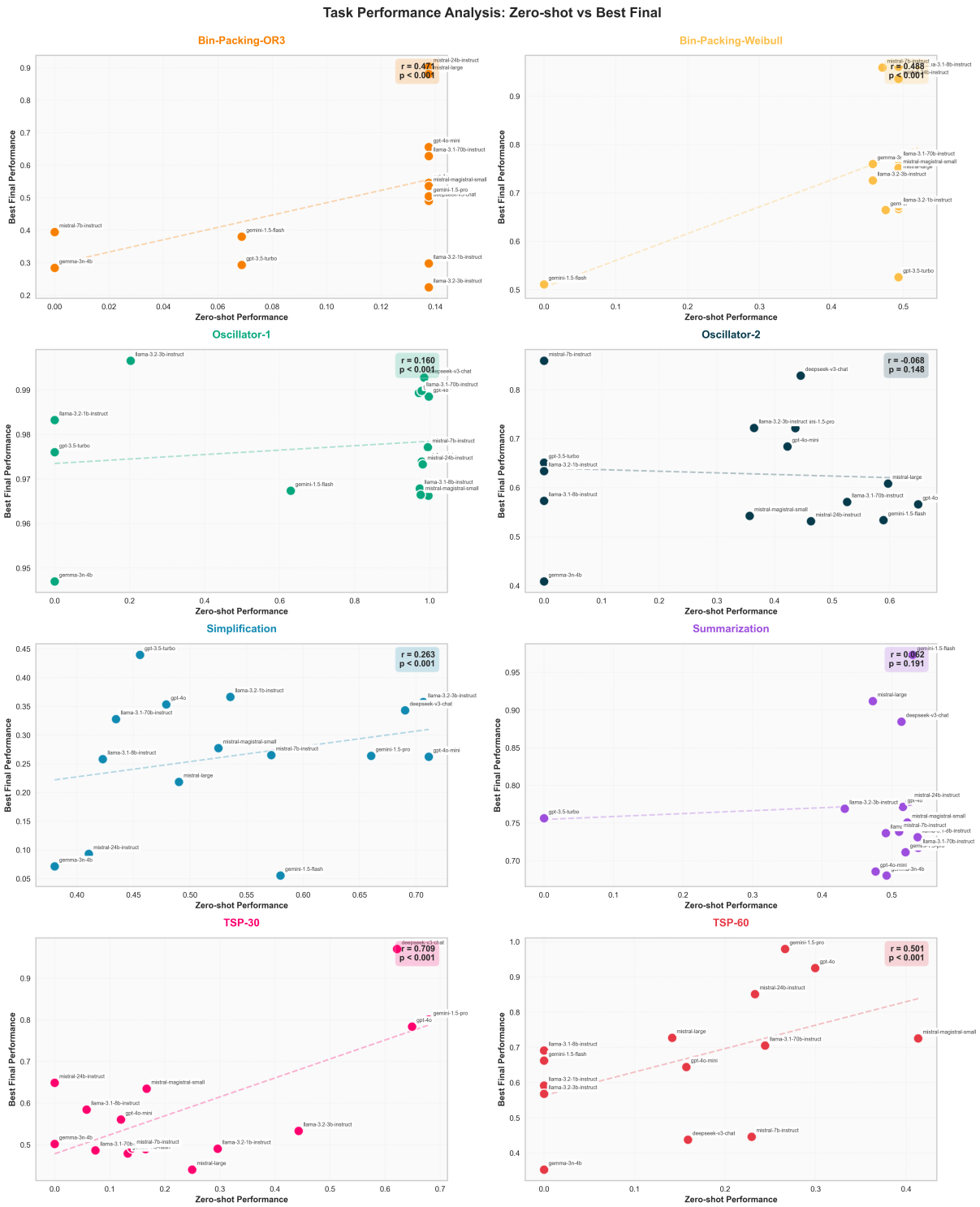


Figure 13: Zero-shot Performance Versus Post-Optimization performance for each task

Table 15: Mixed-Effects Models: Concurrent vs. Lagged Breakthrough Prediction

Fixed Effect	**Concurrent**			**Lagged (+1 Gen)**		
	$\hat{\beta}$	SE	p	$\hat{\beta}$	SE	p
Main Effects						
$H_{\text{fitness},z}$	-0.073	0.026	0.005	-0.074	0.024	0.002
$H_{\text{spatial},z}$	-0.015	0.024	0.532	0.012	0.022	0.593
mean_novelty$_z$	0.070^{**}	0.026	0.006	0.016	0.025	0.517
max_novelty$_z$	0.029	0.023	0.202	0.006	0.022	0.787
Interaction						
novelty $\times$ H_{spatial}	-0.090^{***}	0.010	< 0.001	-0.051^{***}	0.009	< 0.001
Temporal						
generation$_z$	-0.250^{***}	0.018	< 0.001	-0.193^{***}	0.017	< 0.001
Model Statistics						
No. Observations		3,570			3,451	
No. Groups (models)		15			15	
Log-Likelihood		-4639.77			-4203.99	
Residual Variance		0.7798			0.6621	
Random Intercept Var		0.034			0.033	

$^{*}p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$. All predictors z-scored. Task fixed effects included but not separately reported. SE = model-level clustered standard error. Random intercept allows LLM-specific deviation from population mean.

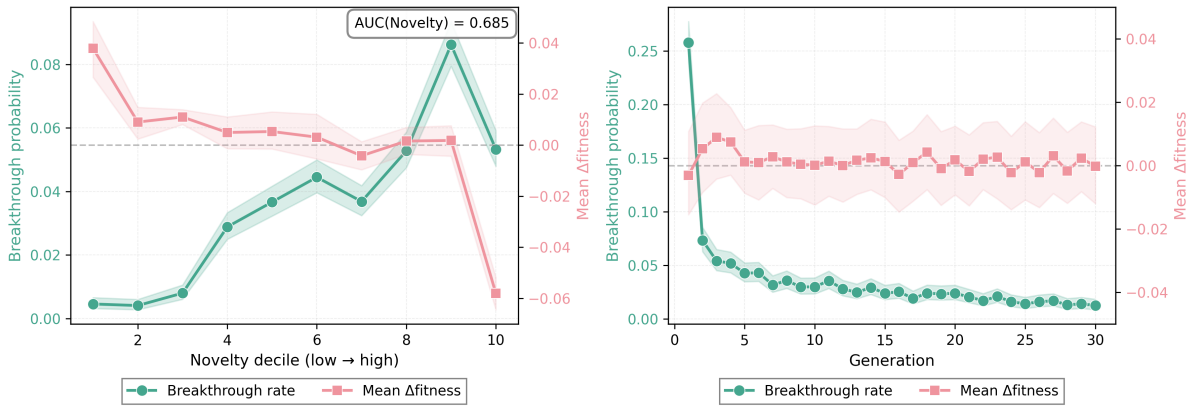


Figure 14: Interaction between breakthroughs and Novelty

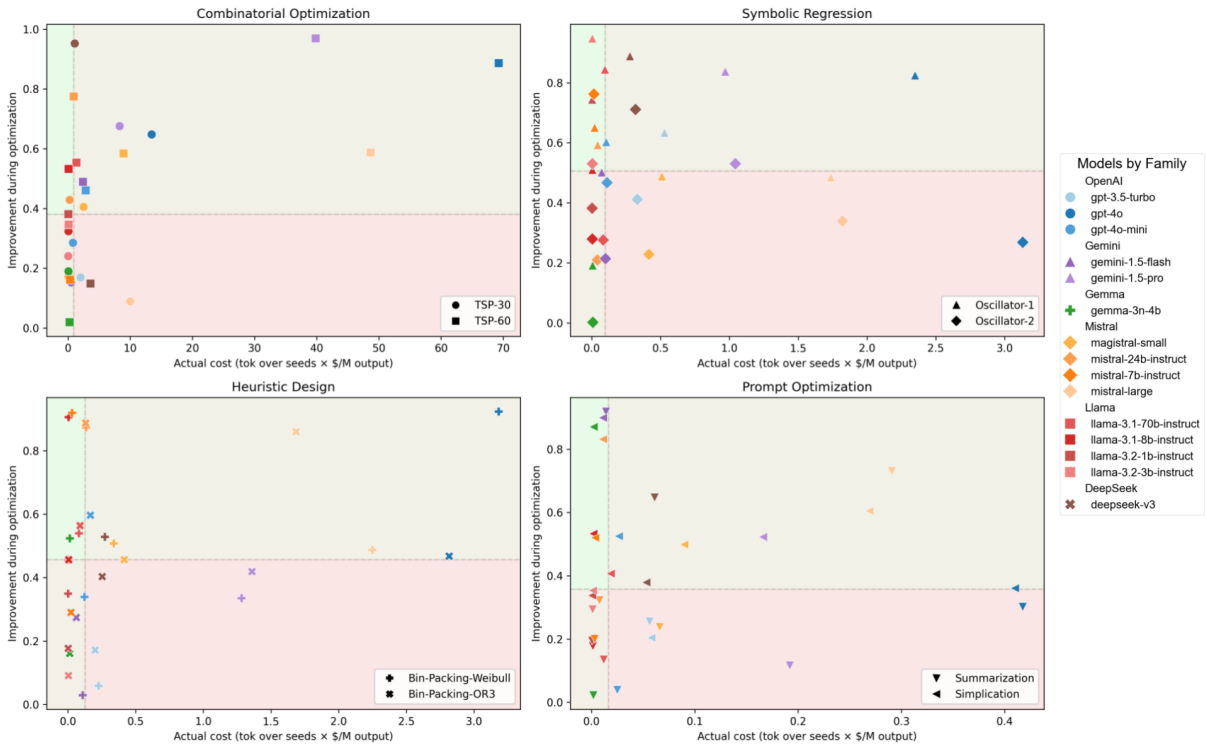


Figure 15: Cost-efficiency plots for four task families

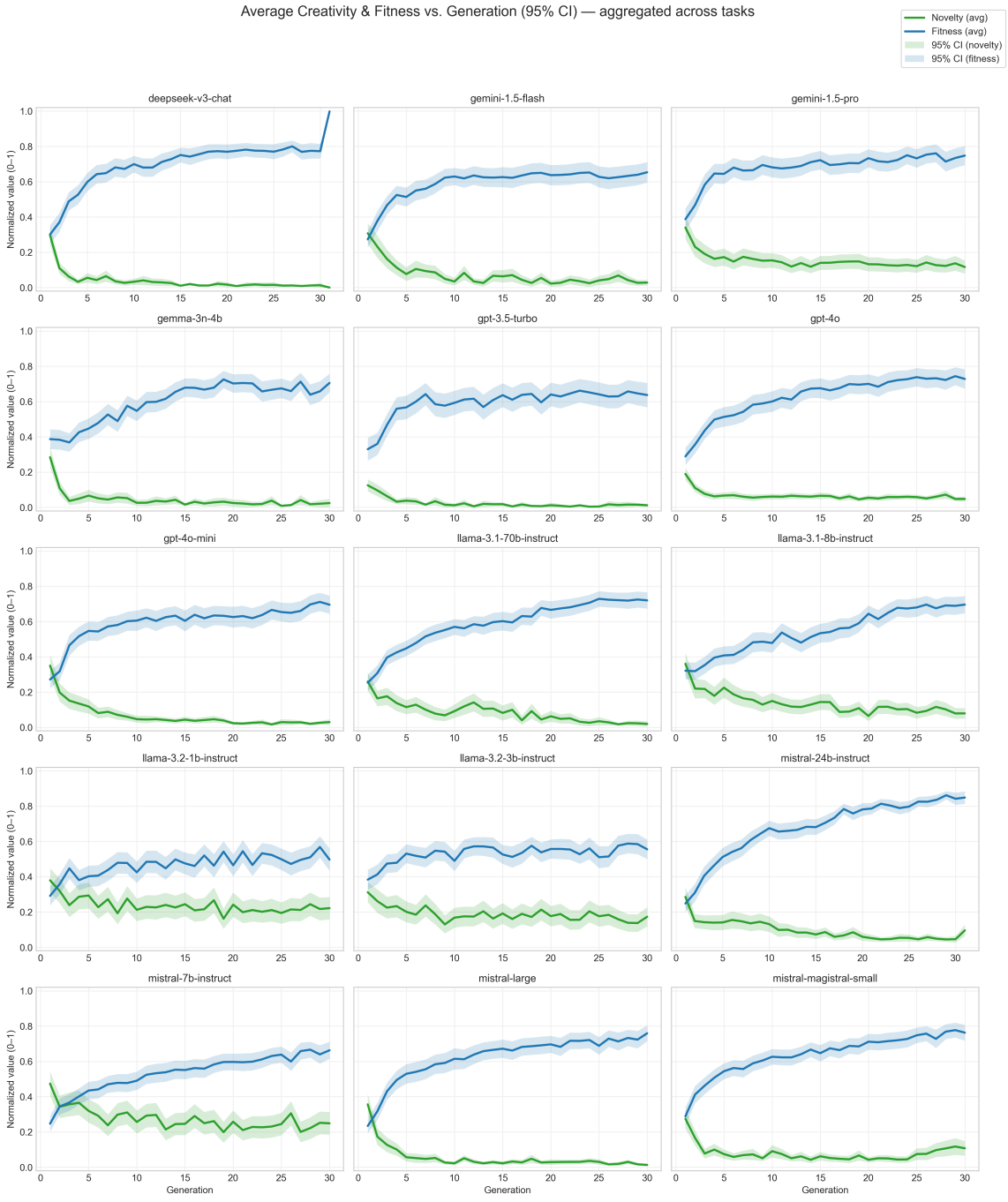


Figure 16: Novelty and fitness coevolution line-plots aggregated over tasks (exploration–exploitation tension)