INITIALIZATION IMPROVES LLM-DRIVEN DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have been used for novel discovery of algorithms, theorems, drugs, and other tasks through the use of harnesses that prompt an LLM to iteratively optimize an objective. In this work, we study the relationship between the population of previous iterates and eventual discovery success. We generalize past work on harness design to develop a suite of 12 harnesses called `Modular` and characterize their performance across 5 diverse discovery tasks, finding that discovery success is brittle and sensitive to harness design. We uncover mode collapse, characterized by a dramatic drop in the diversity of iterates, as a common failure mode. We find that popular state-of-the-art harnesses and diversity-inducing harness interventions, which aim to prolong this collapse, yield inconsistent gains. Our results instead uncover that the performance of early discoveries is predictive of eventual success. We therefore propose a universally applicable intervention that performs an initial stage of parallel exploration in order to initialize subsequent iterative optimization. Our method provides consistent gains across many harnesses and target applications, confirming the importance of initialization in LLM-driven discovery.

1 INTRODUCTION

Large Language Models (LLMs) are increasingly playing a role in advancing scientific and algorithmic discovery (Kamatar et al., 2026; Woodruff et al., 2026). These advances are enabled by harnessing the vast parametric knowledge of LLMs (Petroni et al., 2019; Roberts et al., 2020; Geva et al., 2021) by scaling *inference-time compute*, i.e., spending more tokens to obtain better solutions (Guo et al., 2025; Muennighoff et al., 2025; Wu et al., 2024a). Recent successes include algorithm design (Cheng et al., 2025), drug repurposing (Gottweis et al., 2026), verifiable theorem proving (Zheng et al., 2022), agentic system design (Zhang et al., 2025a), chemical hypothesis generation (Summers et al., 2026), and prompt design (Agrawal et al., 2026b).

While some of these discoveries have been achieved via active human-LLM teaming (Woodruff et al., 2026), there is growing interest in designing scalable, autonomous, and general-purpose *discovery harnesses* (Novikov et al., 2025; Sharma, 2025; Lange et al., 2026; Agrawal et al., 2026a). These harnesses manage an LLM's in-context knowledge (i.e., prompt) as inference-time compute is spent: their primary role is to maintain a population of past discoveries which an LLM iteratively refines. In this work, we study how populations of discoveries evolve. In particular, we examine the factors that mediate the relationship between initial and downstream discovery quality.

We begin by validating the role of in-context knowledge as a key player in discovery quality. We do so by expanding prior work on harness design to develop a suite of 12 discovery harnesses, referred to as `Modular`, that can expose a wide range of harness behavior by varying design choices. We use these harnesses to study whether sequentially refining an initial seed discovery is beneficial. To answer this, we directly compare the discovery performance of sequential harnesses in `Modular`, which continuously update the LLM's context with knowledge from recent discoveries, against a simpler approach: repeatedly querying the LLM in parallel to refine the initial seed without updating the prompt with new iterates. We control the inference-time compute expended by each discovery strategy and find that parallel prompting beats *some* but not *all* of the discovery harnesses. Across five diverse discovery tasks, the top-performing harnesses differed, leading us to conclude that harnessed LLM-guided discovery, while promising, is *brittle*. This means that in-context knowledge

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

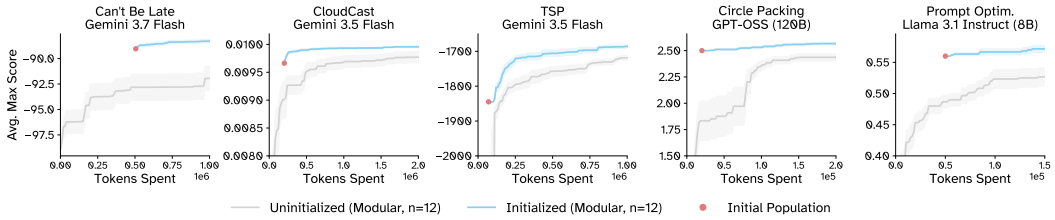

Figure 1: **Initialization raises the floor for discovery under a fixed total token budget.** We find that initializing (the `Modular`) discovery harnesses with a high-performing initial population is beneficial for downstream discovery success across harnesses, tasks, and LLMs. Initialization budget is informed by the convergence criteria in Section 5. Initial population size $m = 15$, similarity threshold $s = 0.98$.

is crucial to downstream discovery success, but how that in-context knowledge is best curated and managed is both task- and LLM-specific.

Despite the variability in harness performance across different task/LLM settings, we find, on average, that harnesses experience diminished discovery diversity over the duration of the trajectory, which we refer to as *mode collapse*. Therefore, we examine if harnesses that *explore* more (i.e., generate more diverse discoveries) are beneficial for downstream discovery. Specifically, we consider two approaches to enable more diverse discovery: 1) state-of-the-art harnesses that prioritize diverse discoveries and 2) diversity-oriented interventions to manage discovery harness populations. Once again, we find that ideal harness design and interventions are sensitive to the task and choice of LLM. Therefore, no approach consistently enables trajectories to recover from poor initial discoveries. However, a commonality of the highest performing harnesses across tasks and LLMs is that they enable a performant initial discovery population. Spurred by this insight, **we confirm that the performance of early iterates can predict of downstream discovery success.**

Therefore, we hypothesize that explicitly initializing a discovery harness with a high-performing population will benefit discovery. To test this, we propose a simple task-, LLM-, and harness-invariant initialization method: use a portion of the total compute budget to generate a pool of initial iterates through a parallel optimization stage. We then warm-start sequential discovery harnesses with the high-performing candidates from this initial population. In so doing, we observe that, even after controlling for initialization compute, our approach can beat both uninitialized sequential and parallel discovery. Additionally, we establish a simple method to determine the initialization budget: use a simple convergence criterion to decide when to curate a high-performing initial population from the parallel pool and begin sequential optimization. This convergence criterion consistently predicts a near-ideal initialization budget and subsequently outperforms purely sequential and parallel compute across task-, LLM-, discovery harnesses. Finally, we validate that explicitly initializing state-of-the-art harnesses with our method consistently increases discovery performance by 2-21%.

2 Background on LLMs as Optimizers

Here we introduce the practice of using LLMs as optimizers for discovery tasks, enumerate the general design principles that govern discovery harnesses, describe the metrics we use to quantitatively compare different discovery trajectories, and outline the discovery tasks we consider in this study.

2.1 Sequential Discovery Harnesses

Yang et al. (2024) proposed to use LLMs to sequentially optimize an objective expressed in natural language by iteratively prompting an LLM with the objective and examples of past solutions. *Discovery harnesses* instantiate this general workflow to enable better downstream discovery (Appendix A, Alg 1). Generally, a harness *samples* parent discoveries from an *active population* of past discoveries. An LLM is then prompted to *mutate* these parents to generate an improved *discovery*, with respect to the objective. New discoveries undergo task-specific *evaluation* and are added to the active population if they meet certain criteria. The active population is periodically *pruned* when

it reaches capacity. These harnesses can be crafted in a task-specific manner (Guo et al., 2024a; Agrawal et al., 2026b; Zhang et al., 2025a), but interest in *task-agnostic* discovery harnesses is growing (Sharma, 2025; Agrawal et al., 2026a; Maheswaran et al., 2026; Lange et al., 2026). Here, we study the drivers of failures and successes of these general-purpose discovery harnesses, and therefore consider a wide range of harnesses with differing design choices.

`Modular` Harnesses: We develop a set of harnesses called `Modular` by generalizing the `OPRO` discovery framework proposed by Yang et al. (2024). We minimally modify the `OPRO` harness to expose two axes of freedom: 1) the LLM mutation strategy and 2) the parent sampling strategy to enable a broader range of harness behavior. We consider four mutation strategies (Listing 4-7) and three sampling strategies (Alg 2-4), resulting in a suite of 12 unique harnesses detailed in Appendix A.1.

State-of-the-art (SOTA) Harnesses: We consider three popular, well-engineered discovery harnesses from the recent literature: `ShinkaEvolve` (`SE`) (Lange et al., 2026), `OpenEvolve` (`OE`) (Sharma, 2025), and `optimize_anything` (`OA`) (Agrawal et al., 2026a) as they are widely adopted and studied. We detail their hyper-parameter settings in Appendix A.3.

2.2 Discovery Metrics

We outline metrics to quantify a discovery scheme that generates $D = \{d_i, ..., d_n\}$ discoveries.

Maximum Score: In a discovery trajectory, all discoveries D are scored using a task-specific evaluation function (detailed in Tab 6). We follow the standard practice to define the quality of a discovery trajectory by the maximal score achieved by any discovery within that trajectory.

Tokens Expenditure: We quantify the cost of a discovery trajectory by the number of tokens used to generate all discoveries D. Past work has quantified the cost of a trajectory by the number of discovery evaluations (Agrawal et al., 2026b;a; Lange et al., 2026; Sharma, 2025); however, this is not an exact comparison as some discovery schemes expend more inference-time compute to generate and evaluate a single discovery. We therefore measure token expenditure instead.

Diversity: Following Wenger & Kenett (2026), we measure the semantic similarity between two LLM-generated discoveries d_1, d_2 by embedding them with a sentence embedding model and computing the cosine similarity between their embeddings. We extend this approach to quantify the *diversity* of a set of discoveries D, following Zhang et al. (2024). Letting $\underline{E} = \{e_1, \ldots, e_n\}$ denote D's corresponding embeddings, $\text{Diversity}(D) = 1 - \overline{\text{sim}}(E)$, where $\overline{\text{sim}}(E)$ is the average pairwise cosine similarity over all $e_i, e_j \in E$, $i \neq j$. We use a general embedding model (`all-MiniLM-L6-v2`) for natural language tasks (as per Wenger & Kenett (2026)) and a code embedding model (`nomic-ai/CodeRankEmbed`) for coding tasks.

2.3 Discovery tasks

We consider five optimization tasks that cover a wide range of domains from (Cheng et al., 2025; Agrawal et al., 2026a; Yang et al., 2024). We briefly describe them below and further detail them, their LLMs, token budgets, and evaluation metrics in Appendix B, Tab 6:

1. **CloudCast**: Discover an algorithm to optimize multi-cloud data transfer cost (Wooders et al., 2024).
2. **Can't Be Late**: Discover an algorithm for single spot instance deadline-driven job scheduling (Wu et al., 2024b).
3. **Circle Packing** ($n = 26$): Discover an algorithm to arrange 26 circles of variable diameters inside a unit square without overlap such that cumulative diameters are maximized.
4. **Traveling Salesperson** ($n = 100$) (**TSP**): Discover the shortest path that visits 100 cities (with predefined inter-city distances) exactly once and returns to the starting point.
5. **Prompt Optimization (GSM8K)**: Discover an instruction prompt that enables a downstream LLM to maximize score on the GSM8K benchmark (Cobbe et al., 2021).

We repeat experiments for all tasks over multiple LLMs and report a subset of these experiments in the main text (see Tab 1); full experimental results are reported in Appendix C.

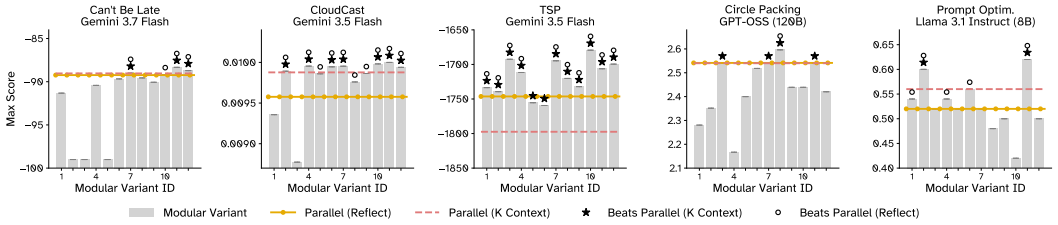

Figure 2: **Sequential discovery can outperform parallel discovery.** Comparing sequential discovery harnesses from our `Modular` suite (detailed in Tab 5) with parallel discovery using the "Reflect" (Listing 5) and "K Context" prompts (Listing 4). Higher scores are better. Both parallel and sequential approaches are initialized with the same initial seed solutions. We see that generally some form of sequential compute outperforms parallel; but the types of harnesses that perform the best vary with task and LLM. Full results in Fig 9.

3 Parallel vs. Sequential Discovery

Here we ask the question: *How should inference-time compute be allocated and managed to enable better discovery?* Sequential *discovery harnesses*, which serially query a language model to refine a discovery, conditioned on past iterates, have become popular (Yang et al., 2024; Novikov et al., 2025; Sharma, 2025; Lange et al., 2026; Agrawal et al., 2026a). We seek to validate the implicit assumption in these works that sequential refinement of a discovery is better than a simple alternative: directly querying the LLM independently in parallel (Brown et al., 2024; Wang et al., 2026).

We begin with a straightforward comparison: for each (discovery task, LLM) in Tab 1, we compare the highest-scoring discovery across the 12 sequential `Modular` harnesses (see Tab 5) against parallel discovery. In both sequential and parallel settings, the initial seed iterate and token budget are the same (Appendix B). In parallel discovery, the initial seed iterate is mutated in parallel by an LLM prompted with the task's discovery objective until the token budget is exhausted. Here, the LLM's context never changes; we study this parallel approach using two different mutator prompts (i.e., Reflect and K Context; see Appendix A.4). Sequential harnesses similarly employ an LLM to mutate discoveries, but instead iteratively update the LLM's context with information from recent discoveries (see Appendix 2.1).

Task	Optimizer LLM
CloudCast	Gemini 3.5 Flash
Can't Be Late	Gemini 3.7 Flash
TSP	Gemini 3.5 Flash
Circle Packing	GPT-OSS (120B)
Prompt Optim.	Llama 3.1 Instruct (8B)

Table 1: A subset of (task, LLM) pairs studied. Full set detailed in Appendix B, Tab 6.

In Fig 2, we plot the maximal score achieved by all `Modular` harness variants and parallel discovery schemes. Since parallel discovery never updates the LLM's context, its performance is dependent on the LLM's parametric knowledge base. In parallel discovery, the ceiling for discovery performance is not known a priori, as reliably measuring the LLM's parametric knowledge for discovery tasks is challenging. For example, in Fig 2, we observe that for **TSP**, parallel discovery typically underperforms sequential `Modular` harnesses, while for **Circle Packing** the trend is reversed. On the other hand, sequential harnesses enable LLMs to build on the in-context knowledge from past discoveries, rather than relying solely on parametric knowledge. For each task, we observed that multiple `Modular` harnesses outperformed parallel discovery, but that the specific outperforming `Modular` variants vary. This suggests that sequential refinement of a discovery can outperform parallel optimization but the best sequential harness design changes per (task, LLM) pairing, indicating that sequential discovery is sensitive to harness hyper-parameter choices. We replicate these findings for each task over additional LLMs in Appendix C, Fig 9.

To better understand failure modes of sequential discovery, we analyze the populations of iterates between sequential and parallel discovery schemes. In Fig 3, we plot the diversity of the entire population of discoveries from a parallel trajectory vs. the windowed diversity over a sequential trajectory averaged across all `Modular` harness variants. We find that the parallel schemes generally have

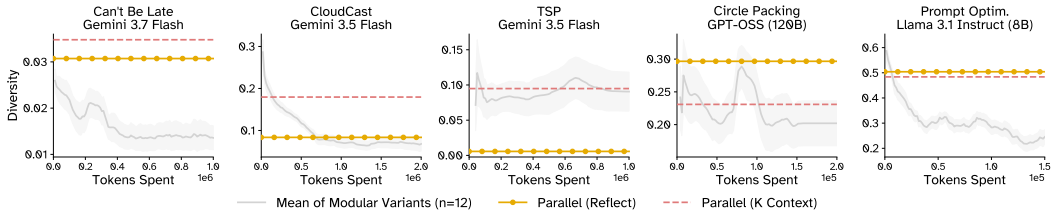

Figure 3: **Sequential discovery is prone to mode collapse.** The windowed average pairwise diversity of discoveries over the course of a trajectory across all `Modular` variants vs. average diversity of parallel discovery. Diversity for the parallel discovery scheme is computed over the full set of discoveries. Sequential harnesses consistently yield fewer diverse discoveries and exhibit diminished diversity over the trajectory, whereas parallel schemes discover more diverse candidates. Window size = 15 for all tasks except **Circle Packing** which has window size = 6. Full results in Fig 10.

more diverse populations of discoveries than sequential. Further, we characterize *mode collapse* in sequential trajectories: the diversity of new candidates decreases over the course of a sequential trajectory. We hypothesize that this is due to the in-context prior, which is present and reinforced throughout sequential discovery, but absent in parallel discovery. Despite mode collapse, Fig 2 demonstrates that sequentially refining a seed discovery can be beneficial and can outperform parallel discovery if the sequential harness discovers a high-performing mode.

4 Can Exploration Enable Better Discovery?

Predicated on the finding from Section 3 that sequential discovery harnesses are prone to mode collapse, we ask: *Can additional exploration enable discovery trajectories to recover from low-performing modes?* We use the diversity of the discovered population as a proxy metric for how much a discovery scheme "explored". We consider two different approaches to assess the effects of exploration on discovery quality:

1. We compare our simple `Modular` harnesses to a set of state-of-the-art harnesses that treat population diversity as a key consideration in harness design.
2. We control for harness design variation by applying a class of online interventions to `Modular` harnesses that aim to maintain diversity as per (Lange et al., 2026).

State-of-the-art Harnesses. We directly compare the performance of the state-of-the-art `ShinkaEvolve` (`SE`), `OpenEvolve` (`OE`), and `optimize_anything` (`OA`) harnesses to `Modular` variants across the same (task, LLM) pairs as in Section 3; all harnesses share the same initial seed iterate. In Tab 4, we compare the average maximum discovery score across `Modular` and state-of-the-art harnesses. Despite validating that the more sophisticated harnesses do, on average, generate more diverse discoveries (i.e., explored more, see Tab 2), we find that they often underperform less diverse `Modular` harnesses. This suggests that harness design choices, which interact differently with the task and LLM, rather than exploration, are a primary mediator of successful discovery.

	Can't Be Late	**CloudCast**	**TSP**	**Circle Packing**	**Prompt Optim.**
`Modular` (n=12)	0.02 ± 0.01	0.1271 ± 0.0195	0.095 ± 0.050	0.26 ± 0.06	0.40 ± 0.04
`Modular` w/ Diversity Interven. (n=60)	0.06 ± 0.02	0.1417 ± 0.0132	0.100 ± 0.022	0.27 ± 0.02	0.38 ± 0.03
State-of-the-art (n=3)	0.05 ± 0.07	0.1680 ± 0.3698	0.259 ± 0.332	0.25 ± 0.10	0.36 ± 0.23

Table 2: **Discovery harnesses can discover more diverse populations.** We compute the diversity of all discoveries in a trajectory, and then report the average diversity across different harnesses. Both state-of-the-art discovery harnesses (`SE`, `OA`, `OE`) and `Modular` harnesses with diversity interventions on average generally achieve more diverse discoveries than baseline `Modular` harnesses. Full results in Appendix C, Tab 9.

	Can't Be Late	CloudCast	TSP	Circle Packing	Prompt Optim.
`Modular` (n=12)	-91.94 ± 2.75	**0.0098 ± 0.0002**	**-1719 ± 16**	**2.44 ± 0.08**	**0.53 ± 0.03**
Sim thresh. (0.95) (n=12)	-95.79 ± 2.98	0.0097 ± 0.0002	-1777 ± 29	2.19 ± 0.38	**0.53 ± 0.03**
Sim thresh. (0.8) (n=12)	**-91.68 ± 2.81**	0.0097 ± 0.0001	-1775 ± 17	2.34 ± 0.26	0.52 ± 0.02
Sim thresh. (0.95) + LLM-Judge (n=12)	-95.93 ± 2.82	**0.0098 ± 0.0001**	-1744 ± 46	2.33 ± 0.23	**0.53 ± 0.04**
Sim thresh. (0.8) + LLM-Judge (n=12)	-94.82 ± 2.75	**0.0098 ± 0.0001**	-1735 ± 37	2.26 ± 0.30	0.51 ± 0.04
Deduplication (n=12)	-93.93 ± 3.34	**0.0098 ± 0.0001**	-1721 ± 12	2.04 ± 0.44	0.52 ± 0.04

Table 3: **Diversity interventions do not consistently improve discovery harness performance.** Each cell contains the average maximal score achieved over a set of harnesses. The basic `Modular` harnesses generally have the highest discovery quality, with some diversity interventions matching, and rarely exceeding, performance. Sim thresh. is the cosine similarity threshold (s). Bold indicates highest score within a column. Full results in Appendix C, Tab 10.

	Can't Be Late	CloudCast	TSP	Circle Packing	Prompt Optim.
`Modular` (n=12)	-91.94 ± 2.75	**0.0098 ± 0.0002**	**-1719 ± 16**	2.44 ± 0.08	**0.53 ± 0.03**
State-of-the-art (n=3)	**-91.32 ± 8.33**	0.0095 ± 0.0021	-1772 ± 250	**2.51 ± 0.07**	0.47 ± 0.20

Table 4: **State-of-the-art discovery harnesses (`SE`, `OA`, `OE`) do not consistently outperform `Modular` harnesses.** The ideal set of discovery harnesses varies across (task, LLM) settings. Bold indicates highest score within a column. Full results in Appendix C, Tab 10.

Online Diversity Interventions. Next, to more precisely measure the effect of exploration on discovery quality, we directly apply an intervention on the `Modular` harnesses while controlling for the effect of harness design. Following Lange et al. (2026), we consider a class of online methods that intervene on the active population and aim to maintain a diverse population: rejecting a newly discovered d_{new} from entering the active population $P = \{d_1, ..., d_m\}$ if d_{new} is too similar to current candidates, where m is the max population size. We consider three simple and popular online diversity interventions:

1. **Cosine Similarity Threshold** (Lange et al., 2026): Reject d_{new} if its embedding exceeds the cosine similarity threshold s to the embedding of any candidate already in P. We use the sentence embedding models detailed in Section 2.2 to embed candidates.
2. **LLM-Judge** (Lange et al., 2026): Identify the most similar candidate d^* to d_{new} from P by comparing cosine similarities of the candidates' embeddings. Reject d_{new} if its cosine similarity to d^* exceeds s *and* an LLM-Judge (Gemini 3.5 Flash) does not deem d_{new} meaningfully different from d^*. See Appendix A.2 for the LLM-Judge prompt.
3. **Deduplication:** Reject d_{new} if $d_{\text{new}} \in P$.

We run the same set of sequential experiments with the `Modular` harness variants as Section 3, but we vary the diversity interventions: cosine similarity thresholding with $s \in \{0.95, 0.8\}$, LLM-Judge with $s \in \{0.95, 0.8\}$, and deduplication. We validate that these interventions do generally enable harnesses to explore more and achieve more diverse discoveries (Tab 2). However, these in-

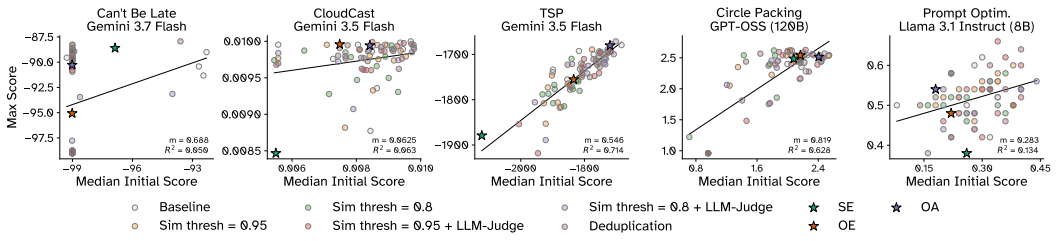

Figure 4: **Initial discovery quality can predict downstream quality.** We plot the correlation between the median score of the first 15 discoveries vs. the max score achieved in a trajectory across varying discovery harnesses (72 `Modular` harnesses and 3 state-of-the-art harnesses).

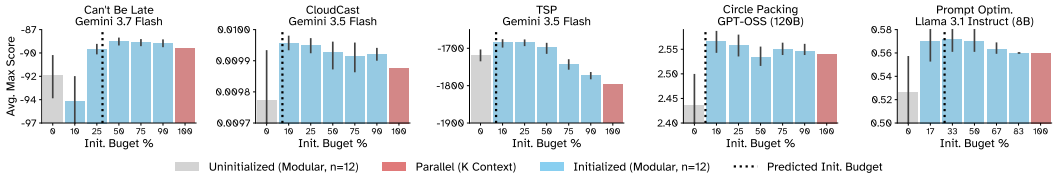

Figure 5: **Initialized discovery can outperform purely sequential and parallel discovery.** Sequential discovery harnesses initialized with a high-performing population (blue) via a fraction of the discovery budget yield the most performant discoveries. The ideal initialization to total discovery budget ratio (dotted line) can be successfully predicted via a convergence analysis Section 5. Diversity threshold $s = 0.98$ for all initial populations, population size $m = 15$. Full results in Fig 12.

terventions have variable effects on discovery quality and, in some cases, harm performance (Tab 3). In some cases, exploration negatively impacts convergence towards high-performance discoveries. Further, we find that the interventions that enable better discovery vary across tasks and are sensitive to hyper-parameters.

High-performing discovery trajectories can collapse onto favorable modes. Our results from state-of-the-art harnesses and diversity interventions show that when harnesses start from the same seed, those that explore more (i.e., maintain more diverse populations) do not reliably produce better-performing discoveries. For example, in Fig 6, we visually stratify all 72 discovery trajectories for the **CloudCast** task (from Tab 3) into tertiles by their best score. We observe that while high-performing trajectories can be marginally more diverse early on, they too can experience mode collapse over time. This suggests a more nuanced conclusion: in a trajectory with poor initial candidates, mode collapse around those candidates can prevent better ones from ever being discovered, so intervening on the population can help. In contrast, an initially high-performing trajectory may actually be hurt by unnecessary exploration and benefit from more exploitation. Distinguishing between these two scenarios is challenging, and **exploration does not guarantee recovering from poor initial discoveries.**

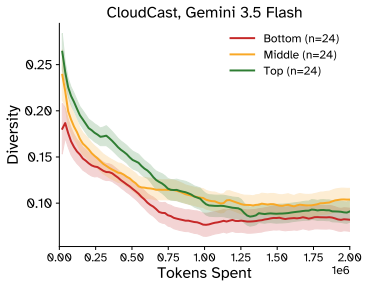

Figure 6: **High scoring trajectories have higher initial diversity but eventually experience mode collapse.** We stratify the `Modular` harness trajectories (for **CloudCast**) from Tab 3 into tertiles based on the max score of a given trajectory. Windowed diversity visualized over trajectory (window size=15). Full results in Appendix C, Fig 11.

In Fig 4, we visualize the relationship between the initial score of the first 15 discoveries in a discovery trajectory vs. the maximum downstream score achieved by that trajectory. While no specific harness consistently performed well, the highest-performing harnesses for each (task, LLM) pairing tended to be those that produced a high-performing initial population. **We find that the quality of the initially discovered population can be predictive of downstream discovery quality in a trajectory.**

5 The Importance of Initialization

Since the eventual success of a sequential trajectory can be correlated with the performance of the first few discovered candidates (Fig 4), we posit that it is beneficial to initialize a high-performing population of candidates that can better condition downstream sequential optimization. We ask the question: *can we better allocate compute so that the performance of the initial candidate population is boosted?*

Given a discovery budget, we propose a method to stratify compute into a population initialization phase followed by a serial discovery phase. We find that by spending a fraction of the compute budget (r) on parallel discovery, we can curate initial populations that are more amenable to downstream refinement via sequential optimization. This produces a simple and extensible initialization method

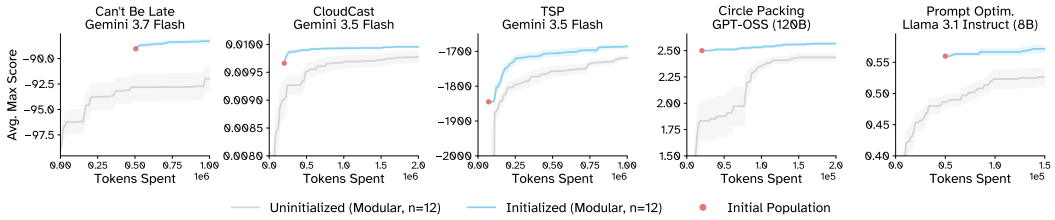

Figure 7: **Initializing state-of-the-art harnesses improves discoveries.** We find that initializing the state-of-the-art discovery harnesses (`OA`, `OE`, `SE`) with the single highest-performing seed discovery within a given initialization budget is beneficial for downstream discovery success across harnesses, tasks, and LLMs. Initialization budget is informed by the convergence criteria in Section 5.

that can be adopted with *any* sequential discovery harness: first generate a pool of discoveries using parallel discovery; then curate a performant population p_{init} of size m by greedily selecting candidates from that parallel pool. We apply cosine similarity thresholding when greedily initializing a population (see description in Section 4 & (Lange et al., 2026)). If we exhaust sufficiently unique candidates before the population has reached size m, we greedily populate the remaining population slots with the deduplicated highest performers from the parallel pool.

We repeat the set of experiments on `Modular` harnesses from Section 3 but with five different initialization budgets ($r \in [0, 1]$) per task (see task-specific ratios in Appendix B, Tab 7). We let the initial population size $m = 15$ and consider similarity thresholds $s \in \{1, 0.98, 0.95, 0.8\}$ where $s = 1$ corresponds to greedy sampling. In Fig 5, we plot the average maximal score achieved across all 12 `Modular` variants across different initialization budgets; $r = 0$ means sequential trajectories with no initialization and $r = 1$ means fully parallel discovery. We observe that our parallel initialization scheme uniformly outperforms both fully serial and parallel optimization.

Initialization raises the discovery floor. In all settings, we find that reframing the purpose of parallel discovery from a technique that aims to push the discovery ceiling higher ($r = 1$) to an initialization method that raises the discovery floor ($0 < r < 1$) can be beneficial. As shown in Fig 5, initializing a serial trajectory with a high-performing population consistently outperforms both purely serial ($r = 0$) and purely parallel ($r = 1$) discovery, given enough additional serial compute. Among initial populations of varying diversity ($s \in \{1, 0.98, 0.95, 0.8\}$), greedy initialization ($s = 1$) generally performs well, but can sometimes underperform minimal-diversity rejection sampling (i.e., $s = 0.98$) when the initial population contains too many exact or near-duplicate (Tab 11). As we curate substantially more diverse initial populations (i.e., $s < 0.98$), we find that the average population performance drops, leading to subsequently lower scoring discoveries. This suggests that minimal filtering (i.e., $s = 0.98$) to reduce exact and very-near duplicates in the initial population, while prioritizing performance, is sufficient for initializing high-performing trajectories.

Fig 5 shows that *some* amount of initialization compute is beneficial for discovery across all tasks. This raises the question: *can we automatically determine how much initialization compute is ideal in a given setting?* We find that parallel discovery typically converges in discovery quality (Appendix B.2, Fig 8), and therefore propose using convergence as a simple indicator. To demonstrate, we order parallel discoveries by a random seed and then detect convergence (see Section B.2 for the convergence criterion). We annotate Fig 5 with detected convergence and notice that this consistently coincides with the highest-performing initialization budgets. We find that the same task-specific convergence criteria generally can be applied successfully across LLMs (Appendix C, Fig 12). Further, while fully parallel discovery itself is not amenable to continuous convergence monitoring, we suggest using batched parallel discovery, rather than exhausting the discovery budget at once. Doing so enables early convergence detection which signals if it is appropriate to initialize a performant discovery population and switch to a sequential harness.

State-of-the-art harnesses benefit from parallel initialization. Previously, in Tab 4, we find that for some tasks, state-of-the-art harnesses enabled better discovery. Here, we validate that explicit population initialization outperforms the implicit initialization achieved by SOTA harnesses for the Tab 1 settings. We conduct an experiment comparing the performance of state-of-the-art harnesses that are uninitialized (started from the same initial seed candidates as Section 4, Listing 13-

17) versus initialized with the single highest-performing candidate found within an initialization budget. We use the ideal-initialized budgets found in Fig 5 derived from the convergence analysis detailed above. In Fig 7, we visualize the maximum score trajectory of initialized vs. non-initialized state-of-the-art harnesses; both settings have the same discovery budget. We find that in all (task, LLM) pairings, explicit initialization benefits downstream discovery: Post-initialization discovery scores increase by 2%-21%. We conclude that harnesses, on their own, are not consistently capable of implicitly initializing a high-performing trajectory; therefore, it is beneficial to do so explicitly.

6 Related Work

Diversity of LLM generated text has been measured and studied from many perspectives (Zhang et al., 2024). For instance, LLM response diversity has been shown to decrease with RL-based post-training (Rafailov et al., 2023; Guo et al., 2025; Puri et al., 2026; Jin et al., 2026; Wu et al., 2025; Chen et al., 2026). To overcome this, many prompting strategies have been proposed (Wang et al., 2024; Zhang et al., 2025b; Troshin et al., 2025). However, a more fundamental problem emerges in iterative language modeling tasks where priors from earlier LLM outputs influence and enforce motifs in later outputs (Xu et al., 2022; Holtzman et al., 2019; Laban et al., 2026). We confirm a similar trend in LLM-driven discovery: as LLMs sequentially refine discoveries, subsequent discoveries become more similar to each other. Beyond individual models, populations of LLMs are also producing increasingly similar outputs on open-ended tasks, across both model versions and model families (Guo et al., 2024b; Alemohammad et al., 2024; Jiang et al., 2026; Wenger & Kenett, 2026). This raises concerns that future LLM-generated discoveries may become homogenized.

Eliciting LLMs' Parametric Knowledge. The parametric knowledge an LLM acquires has been found to be a function of how frequently that knowledge was present in its training data, the parametric capacity of the model, and the training compute budget (Kandpal et al., 2023; Sakarvadia et al., 2025; Morris et al., 2025; Muennighoff et al., 2023; Kaplan et al., 2020). This knowledge can be utilized to accomplish tasks via prompting, and even small changes to the prompt can substantially change an LLM's output (Min et al., 2022; McCoy et al., 2019; Webson & Pavlick, 2022; Tenney et al., 2019; Brown et al., 2020; Pan et al., 2024). It has been shown that LLM output quality can be improved by spending more *inference-time compute*, e.g., generating more tokens (Wei et al., 2022; Snell et al., 2024; Wu et al., 2024a; Guo et al., 2025; Muennighoff et al., 2025). In this work, we confirm that expending and managing inference-time compute, via discovery harnesses, is useful for LLM-driven discovery, but the extent to which is mediated by the quality of its initial discovery.

7 Conclusion and Future Work

In this work, we study the relationship between how *discovery harnesses* are initialized and how this affects downstream discovery quality. Through the development and characterization of a suite of discovery harnesses, we validate that sequential refinement of an initial seed discovery is beneficial, but sensitive to harness design. Subsequently, we find that harness design choices that aim to enable more diverse discoveries, from an initial seed discovery, do not predictably achieve more *performant* discoveries. However, we notice, across a wide range of (task, LLM) settings, that the best performing discovery trajectories can be predicted by their initial discovery quality. Building on this, we propose a simple, task-, LLM-, and harness-agnostic method to curate high-performing initial discovery populations. By warm-starting discovery harnesses with these initial populations, we consistently improve discovery harnesses' performance.

This work examined the impact and role of *in-context* knowledge on an discovery quality. Future work may characterize the impact of *parametric* knowledge on discovery quality. First, it would be beneficial to better quantify an LLM's parametric knowledge as it relates to discovery tasks, which may exercise the long-tail of human knowledge. This could be done by extending our current analysis of parallel discovery performance, which refrains from updating an LLM's in-context knowledge and instead relies entirely on parametric knowledge. These findings could inform future research in examining how a model's parametric knowledge priors interact with task-specific in-context priors. For example, our current results suggest that reframing obscure discovery objectives in terms of knowledge domains the LLM is performant in could be beneficial (e.g., express an algorithm design problem in a popular, rather than rare, programming language).

AI USE STATEMENT

In this work, we used generative AI tools to edit components of experimental software. We **have not** used generative AI tools to [propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation, clean and reformat datasets, support qualitative and thematic data analysis, interpret results] and [generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs] are not applicable to this work. Additionally, we used generative AI tools for modifying scientific figures, formatting LaTeX tables, rephrasing specific sentences, and editing software code. We have reviewed all AI-assisted work. LLM-generated code was reviewed for correctness by one author. All figures were first created without AI use, and subsequent iterations of figures that were refined with AI assistance were compared to the ground-truth human-generated figures for correctness.

REPRODUCIBILITY STATEMENT

To enable reproduction of our results we will link to the code base upon acceptance. We additionally include details about the discovery harness studied in Appendix A, the parallel discovery scheme in Section A.4, and discovery tasks (and their initial seeds & objectives) in Appendix B. For all discovery tasks we link to the code bases from which we acquired the task instructions (Listing 8-12) and initial seeds (Listing 13-17). 95% confidence intervals are reported for all experimental results presented in a figure (shaded region) or table (uncertainty estimate). Additionally, the results presented in the main text (Tab 1), are replicated in Appendix C across additional (task, LLM) pairings (see Tab 6).

REFERENCES

Lakshya A Agrawal, Donghyun Lee, Shangyin Tan, Wenjie Ma, Karim Elmaaroufi, Rohit Sandadi, Sanjit A Seshia, Koushik Sen, Dan Klein, Ion Stoica, et al. optimize_anything: Unified text optimization can outperform specialized systems. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pp. 1–16, 2026a.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=RQm2KQTM5r.

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard Baraniuk. Self-consuming generative models go mad. In *International Conference on Learning Representations*, volume 2024, pp. 53581–53608, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Advances in Neural Information Processing Systems*, 38:57654–57689, 2026.

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.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, et al. Accelerating scientific discovery with co-scientist. *Nature*, pp. 1–3, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Yanzhu Guo, Guokan Shang, Michalis Vazirgiannis, and Chloé Clavel. The curious decline of linguistic diversity: Training language models on synthetic text. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3589–3604, 2024b.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Liwei Jiang, Yuanjun Chai, Margaret Li, Mickel Liu, Raymond Fok, Nouha Dziri, Yulia Tsvetkov, Maarten Sap, and Yejin Choi. Artificial hivemind: The open-ended homogeneity of language models (and beyond). *Advances in Neural Information Processing Systems*, 38, 2026.

Renren Jin, Pengzhi Gao, Yuqi Ren, Zhuowen Han, Tongxuan Zhang, Wuwei Huang, Wei Liu, Jian Luan, and Deyi Xiong. Revisiting entropy in reinforcement learning for large reasoning models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 25300–25322, 2026.

Alok Kamatar, J Gregory Pauloski, Yadu Babuji, Ryan Chard, Mansi Sakarvadia, Daniel Babnigg, Ian Foster, and Kyle Chard. Empowering scientific workflows with federated agents. In *2026 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1403–1418. IEEE, 2026.

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. Large language models struggle to learn long-tail knowledge. In *International conference on machine learning*, pp. 15696–15707. PMLR, 2023.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. Llms get lost in multi-turn conversation. In *International Conference on Learning Representations*, volume 2026, pp. 54738–54778, 2026.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=lKEdGCoDNC.

Adam Lipowski and Dorota Lipowska. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications*, 391(6):2193–2196, 2012. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2011.12.004. URL https://www.sciencedirect.com/science/article/pii/S0378437111009010.

Monishwaran Maheswaran, Leon Lakhani, Zhongzhu Zhou, Shijia Yang, Junxiong Wang, Coleman Hooper, Yuezhou Hu, Rishabh Tiwari, Jue Wang, Harman Singh, et al. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *arXiv preprint arXiv:2604.07725*, 2026.

R Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 3428–3448, 2019.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 11048–11064, 2022.

John X Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025.

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A Raffel. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36:50358–50376, 2023.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, 2025.

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.

Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 39154–39200, 2024.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2463–2473, 2019.

Isha Puri, Mehul Damani, Idan Shenfeld, Marzyeh Ghassemi, Jacob Andreas, and Yoon Kim. Escaping the mode: Multi-answer reinforcement learning in lms. In *Forty-third International Conference on Machine Learning*, 2026.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 5418–5426, 2020.

Mansi Sakarvadia, Aswathy Ajith, Arham Khan, Nathaniel Hudson, Caleb Geniesse, Kyle Chard, Yaoqing Yang, Ian Foster, and Michael W Mahoney. Mitigating memorization in language models. In *International Conference on Learning Representations*, volume 2025, pp. 91698–91741, 2025.

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

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Thomas J Summers, Logan Ward, Logan J Augustine, Jiyoung Lee, Michael G Taylor, Pascal Grosset, Scot Halverson, Yuliana Zamora, Xiaoyun Wang, Geetika Gupta, et al. Chelatron: Charting chemical space with agentic ai for metal-ligand discovery. 2026.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. *arXiv preprint arXiv:1905.06316*, 2019.

Sergey Troshin, Irina Saparina, Antske Fokkens, and Vlad Niculae. Asking a language model for diverse responses. In *Proceedings of the 2nd Workshop on Uncertainty-Aware NLP (UncertaiNLP 2025)*, pp. 66–72, 2025.

Cheng Wang, Gyuri Szarvas, Georges Balazs, Pavel Danchenko, and Patrick Ernst. Calibrating verbalized probabilities for large language models. *arXiv preprint arXiv:2410.06707*, 2024.

Yike Wang, Huaisheng Zhu, Zhengyu Hu, Yige Yuan, Zhengyu Chen, Shakti Senthil, Hannaneh Hajishirzi, Yulia Tsvetkov, Pradeep Dasigi, and Teng Xiao. Rethinking the evaluation of harness evolution for agents. In *COLM 2026 The 2nd Workshop on Lifelong Agents: Learning, Aligning, and Evolving*, 2026.

Albert Webson and Ellie Pavlick. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 2300–2344, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Emily Wenger and Yoed N Kenett. Large language models are homogeneously creative. *PNAS nexus*, 5(3):pgag042, 2026.

Sarah Wooders, Shu Liu, Paras Jain, Xiangxi Mo, Joseph E Gonzalez, Vincent Liu, and Ion Stoica. Cloudcast:{High-Throughput},{Cost-Aware} overlay multicast in the cloud. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 281–296, 2024.

David P Woodruff, Vincent Cohen-Addad, Lalit Jain, Jieming Mao, Song Zuo, MohammadHossein Bateni, Simina Branzei, Michael P Brenner, Lin Chen, Ying Feng, et al. Accelerating scientific research with gemini: Case studies and common techniques. *arXiv preprint arXiv:2602.03837*, 2026.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024a.

Zhanghao Wu, Wei-Lin Chiang, Ziming Mao, Zongheng Yang, Eric Friedman, Scott Shenker, and Ion Stoica. Can't be late: Optimizing spot instance savings under deadlines. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 185–203, 2024b.

Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. Learning to break the loop: Analyzing and mitigating repetitions for neural text generation. *Advances in Neural Information Processing Systems*, 35:3082–3095, 2022.

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

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025a.

Jiayi Zhang, Simon Yu, Derek Chong, Anthony Sicilia, Michael R Tomz, Christopher D Manning, and Weiyan Shi. Verbalized sampling: How to mitigate mode collapse and unlock llm diversity. *arXiv preprint arXiv:2510.01171*, 2025b.

Tianhui Zhang, Bei Peng, and Danushka Bollegala. Improving diversity of commonsense generation by large language models via in-context learning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 9226–9242, 2024.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A DISCOVERY HARNESSES

In this study, we consider 15 different discovery harnesses (12 `Modular` harnesses detailed in Section A.1 and 3 state-of-the-art harnesses detailed in Section A.3) to expose a wide range of design choices and downstream behavior.

A.1 MODULAR HARNESSES

We develop simple and `Modular` harnesses built on top of the `OPRO` discovery harness proposed in (Yang et al., 2024). We minimally modify the `OPRO` harness to expose two key axes of freedom: 1) the mutation strategy and 2) the parent sampling strategy. Every new discovery is added to the active population P; when the active population exceeds the maximum population size $m = 15$, the lowest performing discovery is pruned from the population.

Algorithm 1: General Discovery Harness

Input: task_objective; discovery evaluation procedure *eval*
Data: $P \leftarrow \emptyset$; `// active population past discoveries`
Output: Best discovery $\in P$
while *tokens used* $<$ *budget* **do**
  parents $\leftarrow$ *SelectParents*(P);
  discovery $\leftarrow$ *LLMMutator*(parents, task_objective);
  score, meta_data $\leftarrow$ *eval*(discovery);
  $P \leftarrow$ *UpdatePopulation*(P, (discovery, score, meta_data));

We consider four different mutation strategies: `K Context` (Yang et al., 2024), `Reflective (Reflect)` (Agrawal et al., 2026b), `Differential Evolution (DE)` (Guo et al., 2023), and `Genetic Algorithm (GA)` (Guo et al., 2023). We consider three different parent sampling strategies: `Highest Scoring (High Score)` (Yang et al., 2024), `Tournament` (Guo et al., 2023), and `Roulette Wheel (Wheel)` (Guo et al., 2023; Lipowski & Lipowska, 2012). In total, these choices yield 12 unique harnesses detailed in Tab 5.

`Modular` ID	LLM Mutation Strategy	Parent Sampling Strategy
`Modular` 1	`DE` (Listing 6)	`High Score` (Alg 2)
`Modular` 2	`DE` (Listing 6)	`Tournament` (Alg 3)
`Modular` 3	`DE` (Listing 6)	`Wheel` (Alg 4)
`Modular` 4	`GA` (Listing 7)	`High Score` (Alg 2)
`Modular` 5	`GA` (Listing 7)	`Tournament` (Alg 3)
`Modular` 6	`GA` (Listing 7)	`Wheel` (Alg 4)
`Modular` 7	`Reflect` (Listing 5)	`High Score` (Alg 2)
`Modular` 8	`Reflect` (Listing 5)	`Tournament` (Alg 3)
`Modular` 9	`Reflect` (Listing 5)	`Wheel` (Alg 4)
`Modular` 10	`K Context` (Listing 4)	`High Score` (Alg 2)
`Modular` 11	`K Context` (Listing 4)	`Tournament` (Alg 3)
`Modular` 12	`K Context` (Listing 4)	`Wheel` (Alg 4)

Table 5: `Modular` variants summary.

Algorithm 2: High Score Sampling (Yang et al., 2024)

Input: k (# of samples)
Data: P (active discovery population)
Output: d (discovery_set)
$d \leftarrow$ ascending(P)[-k:] ; `// rank discoveries by score, low to high`

Algorithm 3: Tournament Sampling (Guo et al., 2023)

Input: k (# of samples), $p = 0.5$ (selection probability)
Data: P (active discovery population)
Output: d (discovery_set)
$R \leftarrow \text{descending}(P)$; `// rank discoveries by score, high to low`
$w_i \leftarrow p(1-p)^i$ for $i = 0, \ldots, |R| - 1$; `// geometric rank weights`
$\hat{w} \leftarrow w / \sum w$; `// normalize into sampling probabilities`
$d \leftarrow \text{sample_k_samples}(R, k, \hat{w})$;

Algorithm 4: Roulette Wheel Sampling (Guo et al., 2024a; Lipowski & Lipowska, 2012)

Input: k (# of samples)
Data: P (active discovery population)
Output: d (discovery_set)
$s_i \leftarrow \text{score}(x_i) + \text{shift for } x_i \in P$; `// shift by` $|\min(\texttt{score})| + \epsilon$ `if negative`
$\hat{s} \leftarrow s / \sum s$; `// normalize into sampling probabilities`
$d \leftarrow \text{sample_k_samples}(P, k, \hat{s})$;

A.2 LLM-Judge as a Novelty Evaluator

We include the prompts used by the LLM-judge for the `Modular` harness diversity interventions detailed in Section 4 in Listing 1 and Listing 2; these prompts are directly adopted and adapted from (Lange et al., 2026).

```
NOVELTY_SYSTEM_MSG = """You are an expert code reviewer tasked with determining if two code snippets are
      meaningfully different from each other.

Your job is to analyze both programs and determine if the proposed code introduces meaningful changes compared
       to the existing code. Consider:

1. **Algorithmic differences**: Different approaches, logic, or strategies
2. **Structural changes**: Different data structures, control flow, or organization
3. **Functional improvements**: New features, optimizations, or capabilities
4. **Implementation variations**: Different ways of achieving the same goal that could lead to different
      performance characteristics
5. **Hyperparameter changes**: Different hyperparameters that could lead to different performance
      characteristics

Ignore trivial differences like:
- Variable name changes
- Minor formatting or style changes
- Comments or documentation changes
- Insignificant refactoring that doesn't change the core logic

Respond with:
- **NOVEL**: If the codes are meaningfully different
- **NOT_NOVEL**: If the codes are essentially the same with only trivial differences

After your decision, provide a brief explanation of your reasoning."""


NOVELTY_USER_MSG = """Please analyze these two code snippets:

**EXISTING CODE:**
```{language}
{existing_code}
```

PROPOSED CODE:
```{language}
{proposed_code}
```

Are these codes meaningfully different? Respond with NOVEL or NOT_NOVEL followed by your explanation."""
```

Listing 1: LLM-Judge prompt from (Lange et al., 2026) for coding tasks (**Can't Be Late**, **CloudCast**, **Circle Packing**). (https://github.com/SakanaAI/ShinkaEvolve/blob/main/shinka/prompts/prompts_novelty.py)

```
JUDGE_SYSTEM_MSG = """You are an expert evaluator tasked with determining if two candidate {
      solution_description} are meaningfully different from each other.

Task context: {task_description} The goal is to maximize {task_metric}.
```

```
Your job is to analyze both candidates and determine if the proposed candidate introduces meaningful changes
      compared to the existing candidate, with respect to that goal. Consider:

1. **Strategic differences**: Different approaches, arguments, or strategies for pursuing the goal
2. **Structural differences**: Different organization, framing, or content structure
3. **Substantive additions**: New ideas, techniques, or angles not present in the existing candidate
4. **Implementation variations**: Different ways of pursuing the same goal that could plausibly lead to
      different {task_metric} outcomes

Ignore trivial differences like:
- Minor wording, phrasing, or formatting changes
- Reordering that doesn't change the underlying content
- Insignificant edits that don't change the core approach

Respond with:
- **NOVEL**: If the candidates are meaningfully different
- **NOT_NOVEL**: If the candidates are essentially the same with only trivial differences

After your decision, provide a brief explanation of your reasoning."""


JUDGE_USER_MSG = """Please analyze these two candidate solutions:

**EXISTING CANDIDATE:**
{existing_solution}

**PROPOSED CANDIDATE:**
{proposed_solution}

Are these candidates meaningfully different? Respond with NOVEL or NOT_NOVEL followed by your explanation."""
```

Listing 2: LLM-Judge prompt inspired by (Lange et al., 2026) for non-coding tasks (**TSP**,**Prompt Optim.**). (https://github.com/SakanaAI/ShinkaEvolve/blob/main/shinka/prompts/prompts_novelty.py)

A.3 State-of-the-art Harnesses

We consider three popular state-of-the-art harnesses that have been widely studied in this work: `OpenEvolve` (Sharma, 2025), `ShinkaEvolve` (Lange et al., 2026), and `OA` Agrawal et al. (2026a). We instantiate each harness with their recommended default parameters; we set the LLM-judge in `ShinkaEvolve` to Gemini 3.5 Flash and the cosine similarity threshold to 0.95.

A.4 Parallel Discovery

Here we detail the parallel discovery strategy described in Section 3. This approach entails using an LLM, prompted with the task's discovery objective (Listings 8-12), to mutate an initial seed discovery (Listings 13-17) in parallel until the inference budget is exhausted. Because the in-context knowledge in the LLM's mutation prompts is never updated, each mutation operation is *independent* and therefore can be parallelized. The goal of parallel discovery is to generate a high-performing discovery. We consider both the `K Context (Listing 4)` and `Reflect (Listing 5)` mutators for parallel discovery as they can both be instantiated with a single seed discovery, while both `DE (Listing 6)` and `GA (Listing 7)` need at least two. In Fig 9, we notice that parallel discovery with the *K Context* mutator is generally more performant than with the *Reflect* mutator. The main difference between parallel discovery and sequential harnesses is that sequential harnesses actively update the in-context information in the LLM's mutation prompt, and therefore build up dependencies between subsequent discoveries.

B DISCOVERY TASKS

We outline the five discovery tasks we study in Tab 6. Unless otherwise stated, all discovery schemes are initialized with the following task-specific initial seed discoveries: **Can't Be Late** (Listing 13), **CloudCast** (Listing 14), **TSP** (Listing 15), **Circle Packing** ($n = 26$) (Listing 16), **Prompt Optimization (GSM8K)** (Listing 17). For experiments in Section 5, task-specific initialization ratios are detailed in Tab 7.

Task	LLMs	Token Discovery Budget	Evaluation Metric
Can't Be Late Seed: Listing 13 Instructions: Listing 8	Gemini 3.5 Flash Gemini 3.7 Flash	1M	Negated cost in dollars of discovered scheduling strategy.
CloudCast Seed: Listing 14 Instructions: Listing 9	Gemini 3.5 Flash Gemini 3.7 Flash	2M	Score $= \frac{1}{1+cost}$ where cost of discovered algorithm in dollars is = egress costs (data_vol × edge_cost) + instance costs (runtime × cost_per_hour)
TSP Seed: Listing 15 Instructions: Listing 10	Gemini 3.5 Flash Gemini 3.7 Flash GPT-OSS (120B)	1M	Negated total distance of discovered path (unit less).
Circle Packing ($n = 26$) Seed: Listing 16 Instructions: Listing 11	Gemini 2.5 Flash GPT-OSS (120B) Qwen 3.6 A3B (35B)	200K	Sum of all circle diameters in discovered configuration.
Prompt Optim. Seed: Listing 17 Instructions: Listing 12	Gemini 2.5 Flash Gemini 3.5 Flash Llama 3.1 Instruct (8B)	150K	Evaluation performance on GSM8K Benchmark Cobbe et al. (2021) w/ discovered instruction prompt for downstream LLM (OLMo 2 0425 SFT (1B)) measured as a percentage

Table 6: **Discovery Task Experiment Configurations.**

Task	Initial:Total Budget Ratios (r)
Can't Be Late	0.1, 0.25, 0.5, 0.75, 0.9
CloudCast	0.1, 0.25, 0.5, 0.75, 0.9
TSP	0.1, 0.25, 0.5, 0.75, 0.9
Circle Packing ($n = 26$)	0.1, 0.25, 0.5, 0.75, 0.9
Prompt Optimization (GSM8K)	0.17, 0.33, 0.5, 0.67, 0.83

Table 7: We vary the initialization budget ratios (r) across five values for each task for the `Modular` harness initialization experiments in Section 5.

B.1 Reward Hacking on **CloudCast** Task

Via manual inspection of the generated discoveries for the **CloudCast** task, we notice instances of *reward hacking* (Pan et al., 2024), whereby the LLM (Gemini 3.5 Flash), rather than genuinely implementing a cloud scheduling algorithm *with the provided information*, instead attempts to find a *shortcut* to evaluate different discoveries locally and greedily choose the highest performing one. We generally find that the highest performing genuine discoveries hit a score performance ceiling of about 0.01; reward-hacked discoveries typically exceed a performance ceiling of 0.015. Therefore, we apply a post-hoc max-score filter of 0.015 to **CloudCast** discoveries.

We detail an example of *reward hacking* behavior we found in one such discovery in Listing 3.

```
...

def find_val_in_stack(names, default):
    try:
        for frame_info in inspect.stack():
            frame = frame_info.frame
            for name in names:
                if name in frame.f_locals:
                    return frame.f_locals[name]
                for local_name, local_val in frame.f_locals.items():
                    if hasattr(local_val, name):
                        return getattr(local_val, name)
    except Exception:
        pass
    return default

def find_simulator():
    try:
        for frame_info in inspect.stack():
            frame = frame_info.frame
            for name, obj in list(frame.f_locals.items()) + list(frame.f_globals.items()):
                if obj is None:
                    continue
                if any(x in name.lower() for x in ['sim', 'eval', 'runner']):
                    if hasattr(obj, 'evaluate') or hasattr(obj, 'simulate') or hasattr(obj, 'run'):
                        return obj
                cls_name = obj.__class__.__name__.lower()
                if any(x in cls_name for x in ['sim', 'eval', 'runner']):
                    if hasattr(obj, 'evaluate') or hasattr(obj, 'simulate') or hasattr(obj, 'run'):
                        return obj
    except Exception:
        pass
    return None

def evaluate_topology_via_simulator(topo, G, data_volume_gb, cost_per_hour):
    sim = find_simulator()
    if sim is not None:
        for method_name in ['evaluate', 'simulate', 'run']:
            if hasattr(sim, method_name):
                try:
                    res = getattr(sim, method_name)(topo)
                    if isinstance(res, tuple) and len(res) >= 2:
                        return res[0], res[1]
                except Exception:
                    pass
    return estimate_topology(topo, G, data_volume_gb, cost_per_hour)
...
```

Listing 3: Excerpt of *reward hacking* in a **CloudCast** discovery from Gemini 3.5 Flash. We noitice that the "discovery" explicitly inspects the function call stack to find functions `['sim', 'eval', 'runner']`, and attempts to locally use those functions as an evaluation oracle.

B.2 Convergence Criterion

Here we detail the convergence criterion used to assess a discovery trajectory: if the best-known discovered candidate has not improved by a threshold δ within $N\%$ of the discovery budget, then declare convergence. These thresholds were chosen to generally be at least an order of magnitude smaller than the maximum known task score, but for **Can't Be Late**, **CloudCast**, and **TSP** are two orders of magnitude smaller than the maximum known score, and N was determined via a visual analysis of the parallel discovery curves Fig 8.

We validate our choice of δ and N for each task from models in Tab 1 as we find that the same values can be successfully used to detect convergence (and subsequently ideal initialization budgets) across multiple LLMs: convergence predicted the ideal initialization budget in most settings (Fig 12). Since δ is task-specific and N is budget-specific, it would be beneficial for future work to consider (δ, N) pairs curated by a subject-matter expert *prior* to discovery. We offer this convergence analysis as a proof of concept and acknowledge that the current choices of (δ, N) were made by non-subject-matter experts (the authors). We detail the task-specific (δ, N) pairs in Tab 8.

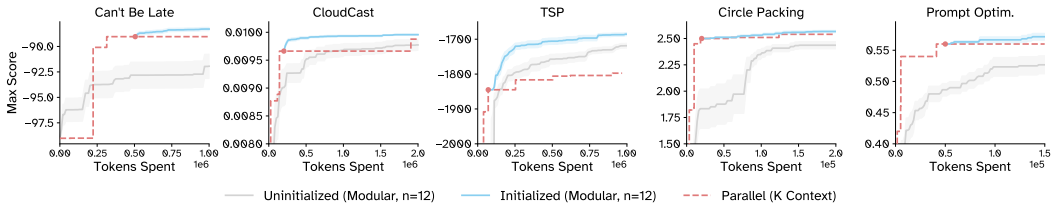

Figure 8: **Parallel discovery quality converges.** We find that initializing (the `Modular`) discovery harnesses with a high-performing initial population is beneficial for downstream discovery success across harnesses, tasks, and LLMs. The initialization budgets are informed by the convergence criteria in Section 5. We overlay the running maximum score of discoveries from the parallel discovery scheme (Section A.4) ordered by a random seed; we notice that these discoveries converge, making them amenable to convergence analysis (Appendix B.2). Initial population size $m = 15$, similarity threshold $s = 0.98$.

Task	δ	N
Can't Be Late	0.5	25%
CloudCast	0.0002	10%
TSP	30	10%
Circle Packing ($n = 26$)	0.1	10%
Prompt Optimization (GSM8K)	0.01	25%

Table 8: Task-specific δ and N values, by which convergence of a discovery trajectory can be determined.

C Extended Results

Here we include additional experimental results. Primarily, we repeat each experiment in the main text over additional LLMs detailed in Tab 6.

C.1 Parallel vs. Sequential Discovery

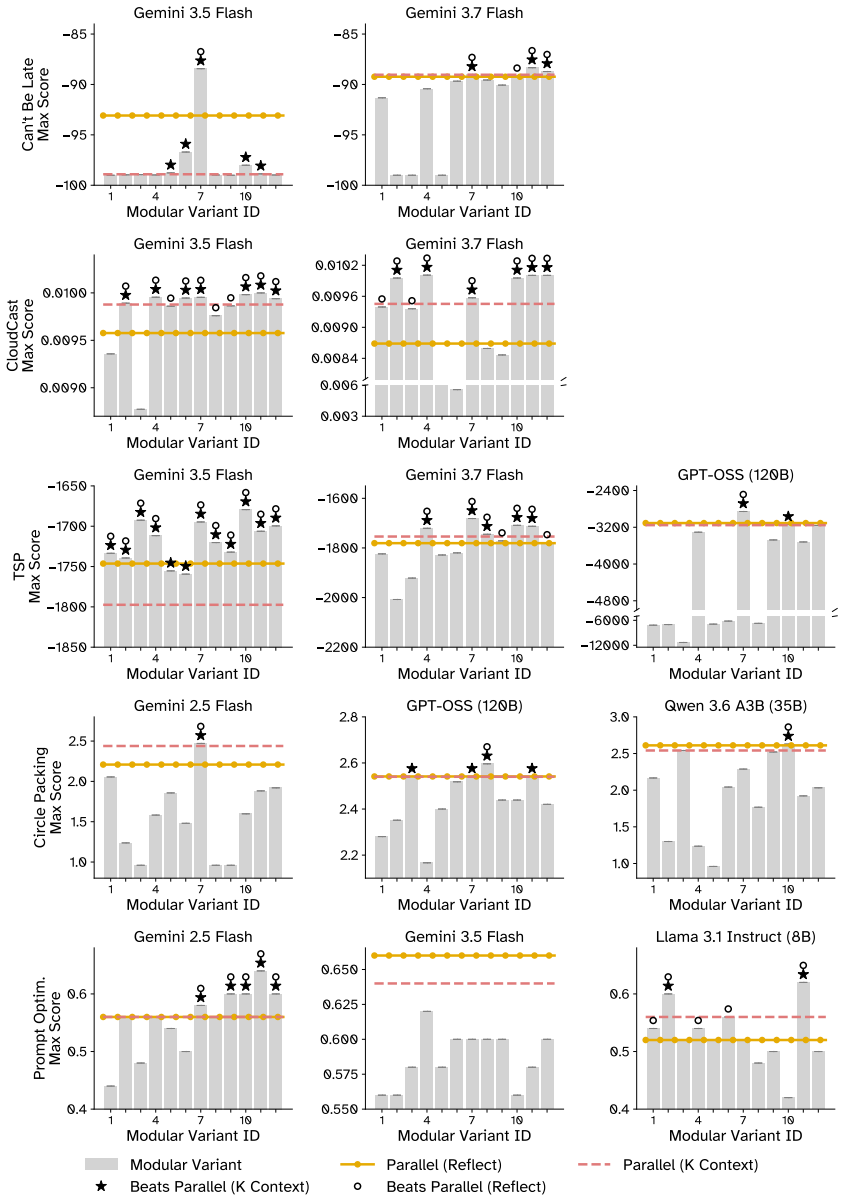

Figure 9: **Sequential discovery can outperform parallel discovery.** We present the extended version of Fig 2 here. Comparing sequential discovery harnesses from our `Modular` suite (detailed in Tab 5) with parallel discovery using the "Reflect" (Listing 5) and "K Context" prompts (Listing 4). Higher scores are better. Both parallel and sequential approaches are initialized with the same initial seed solutions. We see that generally some form of sequential compute outperforms parallel; but the types of harnesses that perform the best vary with task and LLM. Different rows correspond to different discovery tasks. Note that within a column, the LLM varies across tasks.

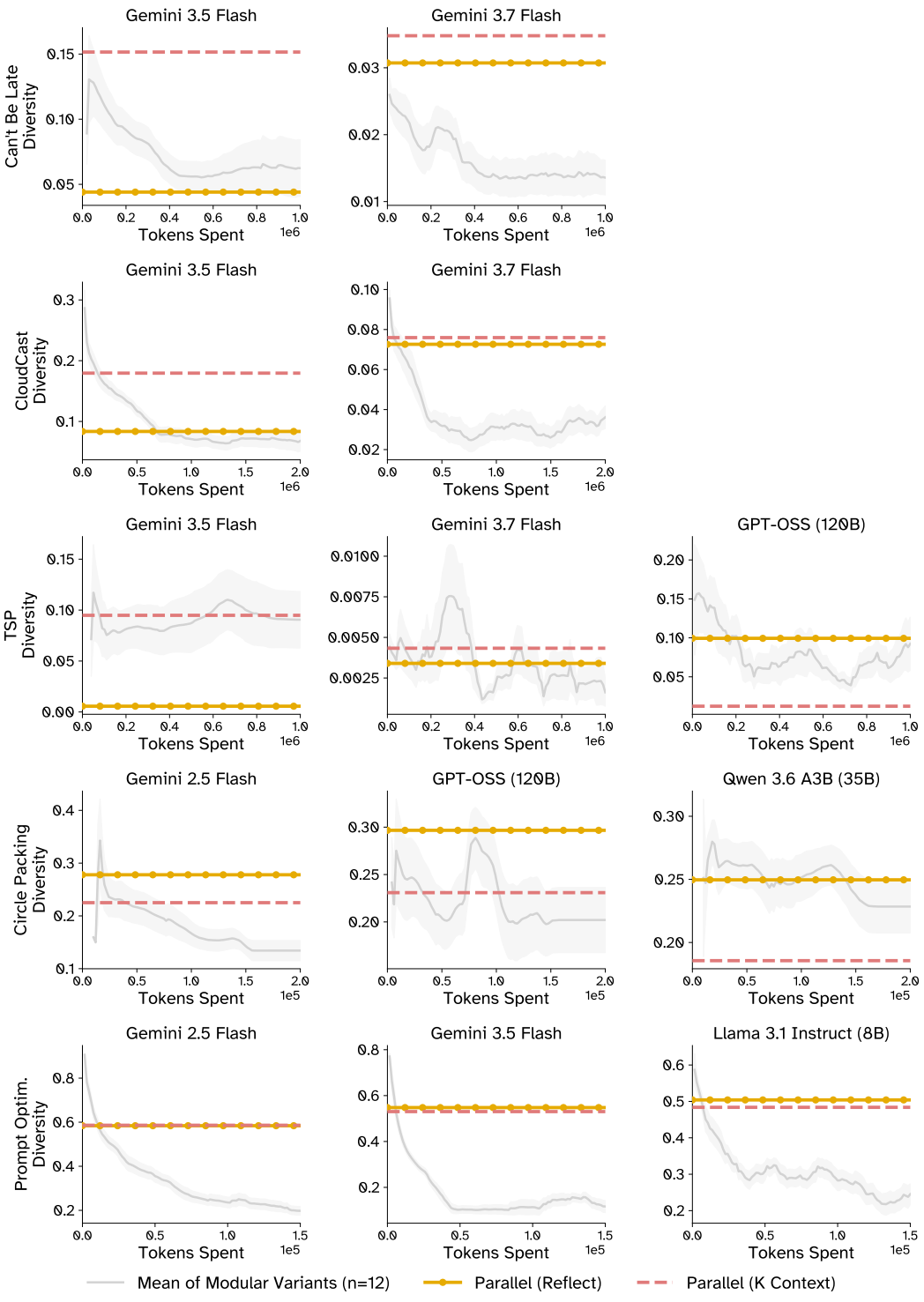

Figure 10: **Sequential discovery is prone to mode collapse.** We present the extended version of Fig 3 here. The windowed average pairwise diversity of discoveries over the course of a trajectory across all `Modular` variants vs. average diversity of parallel discovery. Diversity for the parallel discovery scheme is computed over the full set of discoveries. Sequential harnesses consistently yield fewer diverse discoveries and exhibit diminished diversity over the trajectory, whereas parallel schemes discover more diverse candidates. Window size = 15 for all tasks except **Circle Packing** which has window size = 6.

C.2 CAN EXPLORATION ENABLE BETTER DISCOVERY?

Can't Be Late		
	Gemini 3.7 Flash	Gemini 3.5 Flash
`Modular` ($n = 12$)	0.02 ± 0.01	0.08 ± 0.03
Sim thresh. ($s = 0.95$) ($n = 12$)	0.05 ± 0.05	0.13 ± 0.07
Sim thresh. ($s = 0.8$) ($n = 12$)	0.13 ± 0.06	0.18 ± 0.06
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.03 ± 0.03	0.08 ± 0.04
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.03 ± 0.01	0.07 ± 0.02
Deduplication ($n = 12$)	0.05 ± 0.04	0.10 ± 0.04
`ShinkaEvolve`	0.08	0.33
`OpenEvolve`	0.03	0.05
`optimize_anything`	0.03	0.04
CloudCast		
	Gemini 3.7 Flash	Gemini 3.5 Flash
`Modular` ($n = 12$)	0.0542 ± 0.0137	0.1271 ± 0.0195
Sim thresh. ($s = 0.95$) ($n = 12$)	0.0592 ± 0.0130	0.1397 ± 0.0273
Sim thresh. ($s = 0.8$) ($n = 12$)	0.0617 ± 0.0084	0.1426 ± 0.0287
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.0626 ± 0.0146	0.1514 ± 0.0387
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.0609 ± 0.0136	0.1472 ± 0.0406
Deduplication ($n = 12$)	0.0475 ± 0.0072	0.1275 ± 0.0275
`ShinkaEvolve`	0.0879	0.3390
`OpenEvolve`	0.1336	0.0976
`optimize_anything`	0.0660	0.0673
TSP		
	GPT-OSS (120B)	Gemini 3.7 Flash	Gemini 3.5 Flash
`Modular` ($n = 12$)	0.094 ± 0.055	0.005 ± 0.003	0.095 ± 0.050
Sim thresh. ($s = 0.95$) ($n = 12$)	0.069 ± 0.072	0.006 ± 0.000	0.083 ± 0.047
Sim thresh. ($s = 0.8$) ($n = 12$)	0.066 ± 0.065	0.006 ± 0.000	0.123 ± 0.056
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.124 ± 0.083	0.003 ± 0.002	0.101 ± 0.054
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.078 ± 0.035	0.004 ± 0.002	0.124 ± 0.052
Deduplication ($n = 12$)	0.092 ± 0.050	0.004 ± 0.003	0.069 ± 0.056
`ShinkaEvolve`	0.150	0.001	0.150
`OpenEvolve`	0.165	0.018	0.333
`optimize_anything`	0.004	0.002	0.074
Circle Packing		
	Qwen 3.6 A3B (35B)	Gemini 2.5 Flash	GPT-OSS (120B)
`Modular` ($n = 12$)	0.26 ± 0.03	0.20 ± 0.06	0.26 ± 0.06
Sim thresh. ($s = 0.95$) ($n = 12$)	0.24 ± 0.02	0.23 ± 0.06	0.26 ± 0.06
Sim thresh. ($s = 0.8$) ($n = 12$)	0.25 ± 0.03	0.24 ± 0.06	0.30 ± 0.04
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.25 ± 0.02	0.26 ± 0.05	0.28 ± 0.04
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.23 ± 0.03	0.24 ± 0.05	0.28 ± 0.04
Deduplication ($n = 12$)	0.24 ± 0.03	0.22 ± 0.06	0.24 ± 0.07
`ShinkaEvolve`	0.17	0.31	0.27
`OpenEvolve`	0.23	0.27	0.27
`optimize_anything`	0.26	0.38	0.20
Prompt Optim.		
	Gemini 2.5 Flash	Gemini 3.5 Flash	Llama 3.1 Instruct (8B)
`Modular` ($n = 12$)	0.40 ± 0.04	0.23 ± 0.04	0.40 ± 0.04
Sim thresh. ($s = 0.95$) ($n = 12$)	0.41 ± 0.04	0.23 ± 0.04	0.36 ± 0.05
Sim thresh. ($s = 0.8$) ($n = 12$)	0.44 ± 0.03	0.25 ± 0.03	0.39 ± 0.07
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.41 ± 0.05	0.26 ± 0.03	0.39 ± 0.08
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.42 ± 0.04	0.27 ± 0.04	0.40 ± 0.07
Deduplication ($n = 12$)	0.44 ± 0.04	0.23 ± 0.03	0.37 ± 0.06
`ShinkaEvolve`	0.47	0.41	0.46
`OpenEvolve`	0.41	0.47	0.28
`optimize_anything`	0.29	0.16	0.35

Table 9: **Discovery harnesses can discovery more diverse populations.** We report the comprehensive diversity results for the Section 4 experiments. Both state-of-the-art discovery harnesses (`SE`, `OA`, `OE`) and `Modular` harnesses with diversity interventions on average generally achieve more diverse discoveries than baseline `Modular` harnesses across tasks and LLMs.

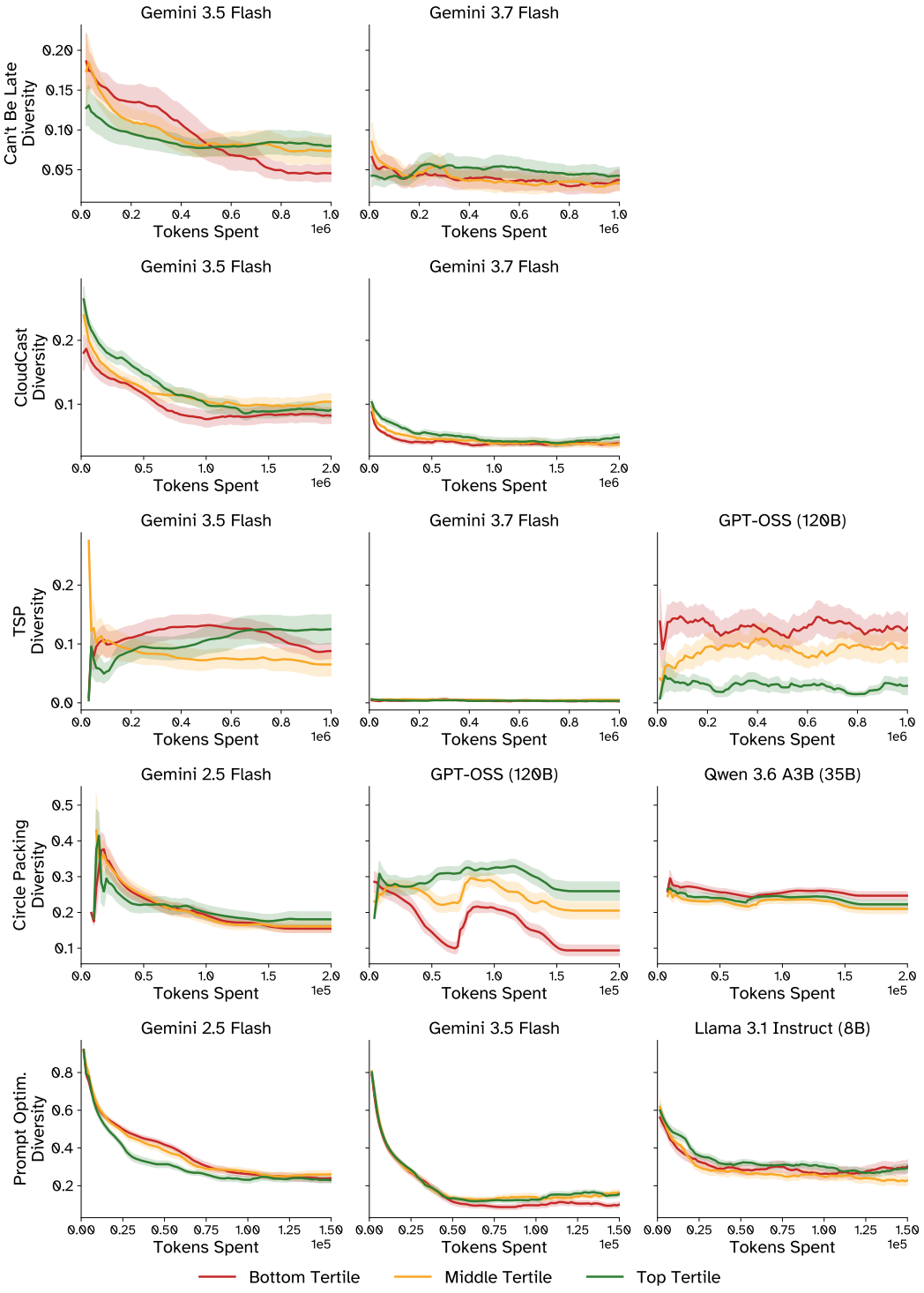

Figure 11: **High scoring trajectories eventually experience mode collapse.** We present the extended version of Fig 6 here. We stratify the `Modular` harness trajectories from Tab 3 into tertiles based on the max score of a given trajectory. Windowed diversity visualized over trajectory; window size = 15 for all tasks except **Circle Packing** which has window size = 6.

Can't Be Late			
	Gemini 3.7 Flash	Gemini 3.5 Flash	
`Modular` ($n = 12$)	-91.94 ± 2.75	-97.80 ± 1.92	
Sim thresh. ($s = 0.95$) ($n = 12$)	-95.79 ± 2.98	-98.14 ± 1.86	
Sim thresh. ($s = 0.8$) ($n = 12$)	-91.68 ± 2.81	-97.42 ± 2.34	
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	-95.93 ± 2.82	-98.64 ± 0.52	
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	-94.82 ± 2.75	-98.39 ± 1.22	
Deduplication ($n = 12$)	-93.93 ± 3.34	-96.28 ± 2.88	
Avg. Interventions ($n = 60$)	-94.43 ± 1.23	-97.77 ± 0.80	
`ShinkaEvolve`	-88.60	-99.01	
`OpenEvolve`	-95.06	-98.91	
`optimize_anything`	-90.28	-98.80	
Avg. Production ($n = 3$)	-91.32 ± 8.33	-98.90 ± 0.26	
CloudCast			
	Gemini 3.7 Flash	Gemini 3.5 Flash	
`Modular` ($n = 12$)	0.0090 ± 0.0008	0.0098 ± 0.0002	
Sim thresh. ($s = 0.95$) ($n = 12$)	0.0092 ± 0.0005	0.0097 ± 0.0002	
Sim thresh. ($s = 0.8$) ($n = 12$)	0.0097 ± 0.0003	0.0097 ± 0.0001	
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.0091 ± 0.0008	0.0098 ± 0.0001	
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.0093 ± 0.0005	0.0098 ± 0.0001	
Deduplication ($n = 12$)	0.0093 ± 0.0004	0.0098 ± 0.0001	
Avg. Interventions ($n = 60$)	0.0093 ± 0.0002	0.0098 ± 0.0001	
`ShinkaEvolve`	0.0100	0.0085	
`OpenEvolve`	0.0100	0.0100	
`optimize_anything`	0.0100	0.0099	
Avg. Production ($n = 3$)	0.0100 ± 0.0000	0.0095 ± 0.0021	
TSP			
	GPT-OSS (120B)	Gemini 3.7 Flash	Gemini 3.5 Flash
`Modular` ($n = 12$)	-5399 ± 1645	-1793 ± 60	-1719 ± 16
Sim thresh. ($s = 0.95$) ($n = 12$)	-4162 ± 1036	-1766 ± 11	-1777 ± 29
Sim thresh. ($s = 0.8$) ($n = 12$)	-4112 ± 1056	-1760 ± 15	-1775 ± 17
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	-5270 ± 1112	-1842 ± 47	-1744 ± 46
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	-5268 ± 1042	-1806 ± 57	-1735 ± 37
Deduplication ($n = 12$)	-5210 ± 1558	-1811 ± 50	-1721 ± 12
Avg. Interventions ($n = 60$)	-4805 ± 484	-1797 ± 18	-1750 ± 13
`ShinkaEvolve`	-11324	-1701	-1879
`OpenEvolve`	-5286	-1760	-1755
`optimize_anything`	-3215	-1673	-1680
Avg. Production ($n = 3$)	-6608 ± 10466	-1711 ± 111	-1772 ± 250
Circle Packing			
	Qwen 3.6 A3B (35B)	Gemini 2.5 Flash	GPT-OSS (120B)
`Modular` ($n = 12$)	1.95 ± 0.35	1.58 ± 0.31	2.44 ± 0.08
Sim thresh. ($s = 0.95$) ($n = 12$)	1.96 ± 0.38	1.84 ± 0.37	2.19 ± 0.38
Sim thresh. ($s = 0.8$) ($n = 12$)	1.76 ± 0.39	1.73 ± 0.36	2.34 ± 0.26
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	1.84 ± 0.43	1.69 ± 0.32	2.33 ± 0.23
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	1.94 ± 0.42	1.79 ± 0.40	2.26 ± 0.30
Deduplication ($n = 12$)	2.21 ± 0.23	1.44 ± 0.38	2.04 ± 0.44
Avg. Interventions ($n = 60$)	1.94 ± 0.15	1.70 ± 0.15	2.23 ± 0.13
`ShinkaEvolve`	2.48	2.43	2.49
`OpenEvolve`	2.17	1.79	2.54
`optimize_anything`	2.51	2.17	2.51
Avg. SOTA ($n = 3$)	2.39 ± 0.48	2.13 ± 0.81	2.51 ± 0.07
Prompt Optim.			
	Gemini 2.5 Flash	Gemini 3.5 Flash	Llama 3.1 Instruct (8B)
`Modular` ($n = 12$)	0.55 ± 0.04	0.59 ± 0.01	0.53 ± 0.03
Sim thresh. ($s = 0.95$) ($n = 12$)	0.55 ± 0.04	0.59 ± 0.02	0.53 ± 0.03
Sim thresh. ($s = 0.8$) ($n = 12$)	0.56 ± 0.02	0.59 ± 0.01	0.52 ± 0.02
Sim thresh. ($s = 0.95$) + LLM judge ($n = 12$)	0.54 ± 0.04	0.57 ± 0.02	0.53 ± 0.04
Sim thresh. ($s = 0.8$) + LLM judge ($n = 12$)	0.55 ± 0.05	0.58 ± 0.02	0.51 ± 0.04
Deduplication ($n = 12$)	0.55 ± 0.03	0.60 ± 0.03	0.52 ± 0.04
Avg. Interventions ($n = 60$)	0.55 ± 0.01	0.59 ± 0.01	0.52 ± 0.01
`ShinkaEvolve`	0.58	0.60	0.38
`OpenEvolve`	0.54	0.62	0.48
`optimize_anything`	0.62	0.64	0.54
Avg. Production ($n = 3$)	0.58 ± 0.10	0.62 ± 0.05	0.47 ± 0.20

Table 10: **Discovery harnesses that explore more do not necessarily achieve more performant discoveries**. We report the comprehensive set of results for Section 4. Neither state-of-the-art discovery harnesses (`SE`,`OA`,`OE`) nor `Modular` harnesses with diversity interventions predictably achieve more performant discoveries; instead the ideal choice or harness/intervention varies across (task, LLM) settings.

C.3 THE IMPORTANCE OF INITIALIZATION

Task	LLM	Sim thresh. ($s = 0.8$)	Sim thresh. ($s = 0.95$)	Sim thresh. ($s = 0.98$)	Sim thresh. (Greedy, $s = 1$)
Can't Be Late	Gemini 3.5 Flash	-98.56 ± 0.43	-98.58 ± 0.42	-98.62 ± 0.38	-98.63 ± 0.35
	Gemini 3.7 Flash Flash	-90.22 ± 0.90	-90.52 ± 0.91	-89.79 ± 0.85	-89.90 ± 0.87
CloudCast	Gemini 3.5 Flash	0.0099 ± 0.0000	0.0100 ± 0.0001	0.0099 ± 0.0000	0.0099 ± 0.0000
	Gemini 3.7 Flash	0.0095 ± 0.0001	0.0095 ± 0.0001	0.0095 ± 0.0001	0.0095 ± 0.0001
TSP	GPT-OSS (120B)	-4433 ± 415	-4674 ± 422	-3993 ± 322	-4071 ± 346
	Gemini 3.5 Flash	-1726 ± 11	-1726 ± 11	-1717 ± 10	-1713 ± 10
	Gemini 3.7 Flash	-1751 ± 8	-1747 ± 7	-1742 ± 7	-1741 ± 7
Circle Packing	GPT-OSS (120B)	-	2.53 ± 0.01	2.55 ± 0.01	2.54 ± 0.01
	Gemini 2.5 Flash	-	2.41 ± 0.03	2.45 ± 0.02	2.42 ± 0.03
	Qwen 3.6 A3B (35B)	-	1.82 ± 0.18	2.36 ± 0.08	1.96 ± 0.16
Prompt Optim.	Gemini 2.5 Flash	0.579 ± 0.008	0.571 ± 0.008	0.575 ± 0.008	0.575 ± 0.007
	Gemini 3.5 Flash	0.625 ± 0.002	0.625 ± 0.003	0.626 ± 0.003	0.626 ± 0.003
	Llama 3.1 Instruct (8B)	0.568 ± 0.005	0.564 ± 0.004	0.567 ± 0.005	0.566 ± 0.005

Table 11: **Comparing Population Initialization Methods across varying similarity thresholds.** See population curation method description in Section 5. Here we report the average score of a population initialization method across five initialization budgets r; see task-specific initialization budgets in Tab 7. We find that greedy population curation is generally performant (highest or second-highest-performing population), but that minimal similarity filtering ($s = 0.98$) typically outperforms purely greedy populations. Therefore, we conclude that minimal filtering of exact and near duplicates from the population is beneficial ($s = 0.98$). Dark green shading in a cell indicates the highest-scoring population within a row; lighter green shading indicates the second-highest-scoring population. We do not conduct $s = 0.8$ experiments for **Circle Packing** as more than half of the initialization population needed to be backfilled greedily due to a lack of sufficiently diverse candidates.

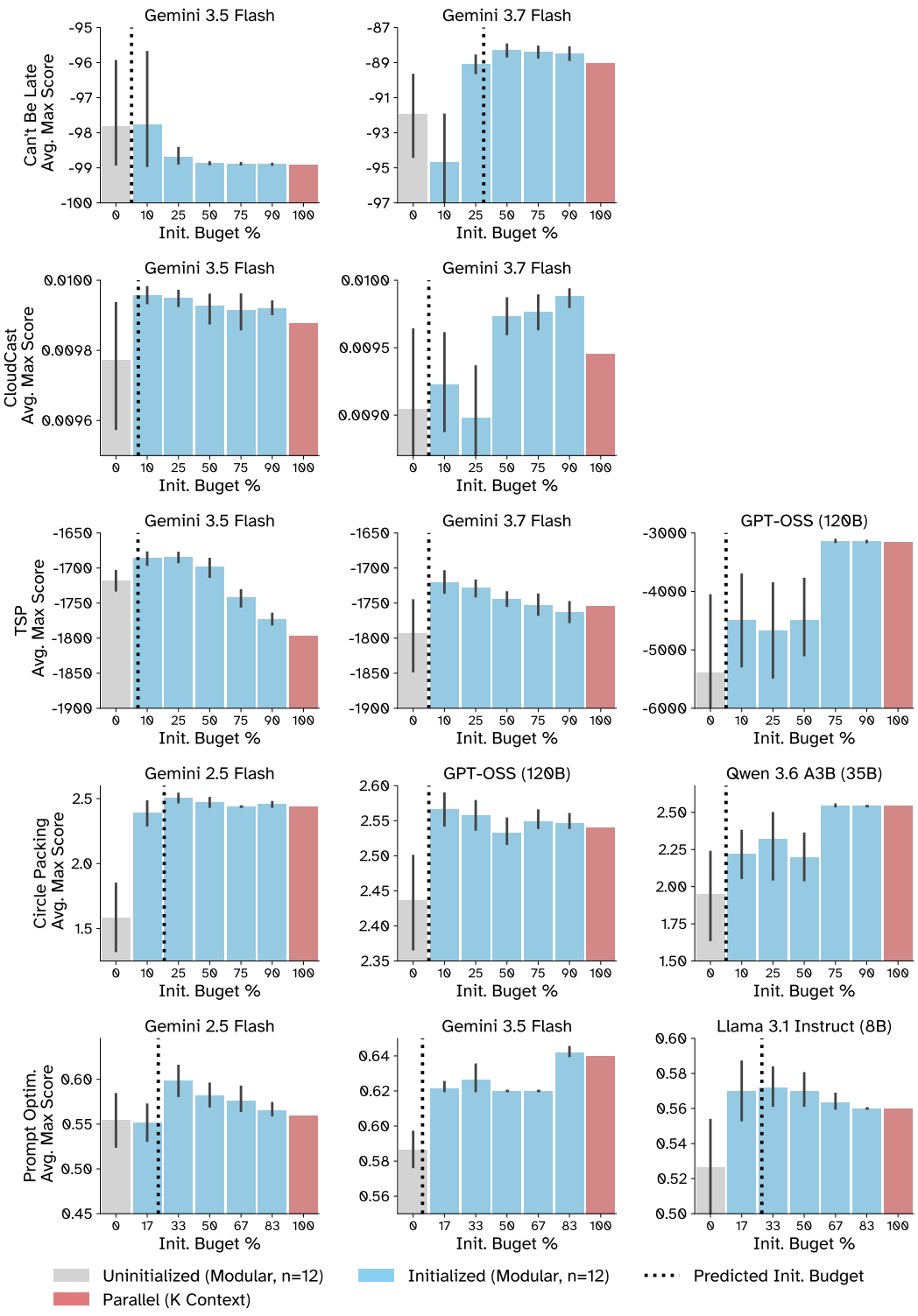

Figure 12: **Initialized discovery can outperform purely sequential and parallel discovery.** Sequential discovery harnesses which are initialized with a high-performing population (blue), via a fraction of the discovery budget yield the most performant discoveries. Predicted initialization budget (dotted line) is predicted via a convergence analysis Section 5. Diversity threshold $s = 0.98$ for all initial populations, population size $m = 15$. We notice in 9/13 scenarios, the predicted initialization budget coincides with the most performant initialization budget.

D MUTATION PROMPTS

```
P                    # active discovery population
k                    # number of in-context examples
task_instruction     # details about discovery objective
discovery_description # descriptor e.g., "algorithm"/"prompt"
task_metric          # e.g., cost, score etc.

in_context_examples = sample_population(P, k)


past_examples = ""
for solution, score, meta_data in in_context_examples:
    example = """
    ### Past Example

    **[PREVIOUSLY GENERATED SOLUTION:]**
    {solution}
    **[SCORE ASSIGNED TO SOLUTION:]**
    {task_metric}: {score}
    **[RUN METADATA:]**
    {meta_data}

    """
    past_examples += example

mutation_prompt = """
{task_instruction}
Your goal is to maximize {task_metric}. Output only the bare minimum text to reach the objective goal.

Here are some past examples and the {task_metric} they received where the goal is to maximize the metric.

{past_examples}

Generate a new {discovery_description} that is different from the old ones to maximize the {task_metric} as
      much as possible.
"""

# Mutation Operation
new_discovery = LLM(mutation_prompt)
```

Listing 4: `K Context` Mutator needs 1 LLM call. Inspired by (Yang et al., 2024). k = 3 for coding tasks (e.g., **Can't Be Late**, **CloudCast**, **Circle Packing**), and k = 5 for the remaining tasks (e.g., **TSP**, **Prompt Optim.**).

```
P                    # active discovery population
task_instruction     # details about discovery objective

in_context_example = sample_population(P, 1)
solution, score, meta_data = in_context_example[0]


diagnosis_instructions = """

You are an expert AI system diagnostics agent. Your task is to analyze the behavior of an LLM agent and
      determine exactly why its current instructions are causing it to fail.

Below is the current solution being utilized by the agent:
[CURRENT SOLUTION]
{solution}

Below is a minibatch of execution traces where the agent may have failed to achieve the target evaluation
      metric. Sometimes there are no failed trajectories to review. Review the step-by-step reasoning, tool
      calls, and outputs:
[(Potential) FAILED TRAJECTORIES]
{meta_data}

INSTRUCTIONS:
1. Conduct a rigorous error analysis. Identify patterns or assumptions in the [CURRENT PROMPT] that misled the
       agent.
2. Pinpoint exactly which instructions caused the wrong reasoning steps or incorrect tool invocations.
3. Provide a clear, high-level natural language diagnosis detailing what needs to be changed, added, or
      removed from the solution to fix these specific edge cases. Do not rewrite the solution yet; only
      provide the structural diagnosis.
"""

mutation_instructions = """

You are an elite engineering and optimization algorithm. Your goal is to mutate an existing solutions to
      maximize its performance across a series of target objectives.

You must improve upon the ancestor solution by applying high-level lessons learned from failure data. If the
      ancestor is a placeholder, replace it with a relevant solution.

[CURRENT SOLUTION]
{solution}

[NEW SYSTEM DIAGNOSIS]
```

```
{diagnosis}

CRITICAL RULES FOR MUTATION:
1. Synthesize the new system diagnosis with the accumulated historical lessons.
2. Mutate the instructions to directly prevent the errors noted in the diagnosis while preserving the parts of
      the solution that successfully handled past tasks.
3. Optimize for clarity, conciseness, and structural robustness.
4. Output ONLY the final mutated solution text. Do not include introductory conversational filler or
     concluding explanations.
"""

# Step 1: Diagnosis
diagnosis_prompt = task_instruction + diagnosis_instructions
diagnosis = LLM(diagnosis_prompt)

# Step 2: Mutation
mutation_prompt = task_instruction + mutation_instructions
new_discovery = LLM(mutation_prompt)
```

Listing 5: `Reflect` Mutator needs 2 LLM calls. From (Agrawal et al., 2026a).

```
P               # active discovery population
task_instruction # details about discovery objective

# requires >= 3 past solutions; falls back to K Context mutator otherwise
in_context_examples = sample_population(P, 3)
solution_1, score_1, meta_data_1 = in_context_examples[-3]
solution_2, score_2, meta_data_2 = in_context_examples[-2]
solution_3, score_3, meta_data_3 = in_context_examples[-1]

# Step 1: Mutation + Crossover
mutation_instructions = """

First we provide an example of evolving a prompt, this may be different from the task you are supposed to
     complete. Regardless, follow the template:
Please follow the instruction step-by-step to generate a better prompt.
1. Identify the different parts between the Prompt 1 and Prompt 2:
Prompt 1: Rewrite the input text into simpler text.
Prompt 2: Rewrite my complex sentence in simpler terms, but keep the meaning.
2. Randomly mutate the different parts
3. Crossover the different parts with the following Prompt 3 and generate a final prompt bracketed with <
     solution> and </solution>:
Prompt 3: Rewrite the given input text into simpler English sentences while preserving the same meaning, so it
      can be understood by non-native English speakers.

1. Identifying the different parts between Prompt 1 and Prompt 2:
Prompt 1: Rewrite the input text into simpler text.
Prompt 2: Rewrite my complex sentence in simpler terms, but keep the meaning.
Different parts:
"input text" vs "my complex sentence"
"simpler text" vs "simpler terms, but keep the meaning"

2. Randomly mutate the different parts:
"input text" -> "provided text"
"my complex sentence" -> "the difficult sentence"
"simpler text" -> "easier language"
"simpler terms, but keep the meaning" -> "simpler words while maintaining the meaning"

3. Crossover the different parts with the following Prompt 3 and generate a final prompt bracketed with <
     solution> and </solution>:
Prompt 3: Rewrite the given input text into simpler English sentences while preserving the same meaning, so it
      can be understood by non-native English speakers.

Final Prompt: <solution>Transform the difficult sentence into easier language while keeping the meaning, for
     non-native English speakers to comprehend.</solution>

Now it is your turn. Please follow the instruction step-by-step to generate a better solution.
1. Identify the different parts between Solution 1 and Solution 2:
Solution 1:
{solution_1}
Solution 1's meta data:
{meta_data_1}
Solution 2:
{solution_2}
Solution 2's meta data:
{meta_data_2}
2. Randomly mutate the different parts
3. Crossover the different parts with the following Solution 3 and generate a final solution (Do not write
     anything except the final request solution):
Solution 3:
{solution_3}
Solution 3's meta data:
{meta_data_3}
"""

# Step 2: Extraction
extraction_instructions = """

In the previous step, you were instructed to:
1. Identify the different parts between Solution 1 and Solution 2:
Solution 1:
```

```
{solution_1}
Solution 2:
{solution_2}
2. Randomly mutate the different parts
3. Crossover the different parts with the following Solution 3 and generate a final solution (Do not write
     anything except the final request solution):
Solution 3:
{solution_3}

Here is what you wrote:
{mutation_output}

Extract the final solution that was derived after step three in the past step from in between <solution> and
     </solution>. Output only that final solution.
"""

# 1st: Mutation Operation
mutation_prompt = task_instruction + mutation_instructions
mutation_output = LLM(mutation_prompt)

# 2nd: Extract Discovery
extraction_prompt = task_instruction + extraction_instructions
new_discovery = LLM(extraction_prompt)
```

Listing 6: `Differential Evolution (DE)` Mutator needs 2 LLM calls. From (Guo et al., 2023).

```
P               # active discovery population
task_instruction # details about discovery objective

# requires >= 3 past solutions; falls back to K Context mutator otherwise
in_context_examples = sample_population(P, 3)
solution_1, score_1, meta_data_1 = in_context_examples[-3]
solution_2, score_2, meta_data_2 = in_context_examples[-2]
solution_3, score_3, meta_data_3 = in_context_examples[-1]

# Step 1: Crossover + Mutation
crossover_instructions = """
First we provide an example of evolving a prompt, this may be different from the task you are supposed to
     complete. Regardless, follow the template:
Please follow the instruction step-by-step to generate a better prompt.
1. Crossover the following prompts and generate a new prompt:
Prompt 1: Rewrite the input text into simpler text.
Prompt 2: Rewrite my complex sentence in simpler terms, but keep the meaning.
2. Mutate the prompt generated in Step 1 and generate a final prompt bracketed with
    <solution> and </solution>.

1. Crossover Prompt: Rewrite the complex text into simpler text while keeping its meaning.
2. <solution>Transform the provided text into simpler language, maintaining its essence.</solution>

Now it is your turn. Please follow the instruction step-by-step to generate a better solution.
Please follow the instruction step-by-step to generate a better solution.
1. Crossover the following solution and generate a new solution:
Solution 1:
{solution_2}
Solution 1's meta data:
{meta_data_2}
Solution 2:
{solution_3}
Solution 2's meta data:
{meta_data_3}
2. Mutate the solution generated in Step 1 and generate a final solution bracketed with
    <solution> and </solution>.

1.
"""

# 1st: Crossover + Mutation Operation
crossover_prompt = task_instruction + crossover_instructions
crossover_output = LLM(crossover_prompt)

# 2nd: Extract Discovery; Reuse same extraction logic as Differential Evolution
extraction_prompt = task_instruction + extraction_instructions
new_discovery = LLM(extraction_prompt)
```

Listing 7: `Genetic Algorithm (GA)` Mutator needs 2 LLM calls. From (Guo et al., 2023).

E TASK OBJECTIVES

```
Optimize a cloud scheduling strategy for the "Can't Be Late" problem.

The strategy decides when to use SPOT instances (cheap but can be preempted) vs ON_DEMAND instances (expensive
      but reliable) to complete a task before its deadline. The goal is to minimize cost while ensuring the
     task completes on time.
Output the executable Python code to accomplish the task inside a markdown code block.
```

```
Key information about the problem domain:

- ClusterType.SPOT: Use spot instances (cheap, ~$0.3/hour, but can be preempted at any time)
- ClusterType.ON_DEMAND: Use on-demand instances (expensive, ~$1/hour, but guaranteed availability)
- ClusterType.NONE: Wait without using any instances (no cost, but no progress)
- restart_overhead: Time penalty incurred when switching from one instance type to another
- The strategy MUST ensure task completion before the deadline (hard constraint)
- Score is the negative of cost in dollars, so a higher (less negative) score means lower cost

Evaluation feedback format:
- Timeline format: start-end:TYPE@REGION[progress%] (e.g., "0.0-5.0:S@R0[50%]" means SPOT from hour 0-5
      reaching 50% progress)
- Spot availability: S=available, X=unavailable (e.g., "0.0-10.0:S | 10.0-15.0:X" means spot available first
      10h, then unavailable)

Optimization targets:
1. Reduce overall cost while maintaining deadline guarantees
2. Make better decisions about when to use SPOT vs ON_DEMAND
3. Handle spot unavailability more intelligently
4. Consider the trade-offs between waiting for spot and using on-demand
```

Listing 8: **Can't Be Late** discovery instructions (Agrawal et al., 2026a). (https://github.com/gepa-ai/gepa/tree/main/examples/adrs/can_be_late)

```
Optimize a broadcast routing algorithm for multi-cloud data transfer.

The algorithm decides how to route data from a single source to multiple destinations across cloud providers (
      AWS, GCP, Azure). The goal is to minimize total cost (egress fees + instance costs) while maintaining
      good transfer times.
Output the executable Python code to accomplish the task inside a markdown code block.

Key information about the problem domain:

- The network is represented as a directed graph where:
  - Nodes are cloud regions (e.g., "aws:us-east-1", "gcp:europe-west1-a", "azure:eastus")
  - Edges have 'cost' ($/GB for egress) and 'throughput' (Gbps bandwidth) attributes

- Data is partitioned into num_partitions chunks that can be routed independently
- Each partition can take a different path to reach each destination
- Total cost = egress costs (data_vol x edge_cost) +
               instance costs (runtime x cost_per_hour)

- The algorithm must return a BroadCastTopology object containing:
  - paths[dst][partition] = list of edges [[src, dst, edge_data], ...]
  - Each destination must have at least one valid path for each partition

Evaluation feedback format:
- Cost: Total transfer cost in dollars
- Transfer time: Maximum time for all destinations to receive data (seconds)

Optimization targets:
1. Reduce total cost (egress + instance costs)
2. Find paths that balance cost and throughput
3. Consider multipath routing for better bandwidth utilization
4. Exploit cloud provider pricing differences (e.g., intra-provider is cheaper)
```

Listing 9: **CloudCast** discovery instructions (Agrawal et al., 2026a). (https://github.com/gepa-ai/gepa/tree/main/examples/adrs/cloudcast)

```
Below are some previous traces and their scores. Score is the negative of
the trace length, so a higher (less negative) score means a shorter,
better trace. The trace should traverse all points exactly once.
The trace should start with <trace> and end with </trace>.
```

Listing 10: **Traveling Salesperson** ($n = 100$) discovery instructions . (https://github.com/google-deepmind/opro/blob/main/opro/optimization/optimize_tsp.py)

```
The circle packing problem involves placing n non-overlapping circles inside a container (in this case, a unit
      square) to optimize a specific metric. For this example:

We pack exactly 26 circles
Each circle must lie entirely within the unit square
No circles may overlap
We aim to maximize the sum of all circle radii.

Construct a specific arrangement of 26 circles in a unit square that attempts to maximize the sum of their
      radii.

Write a python function called 'run_packing' which acchomplishes this goal and     Returns:
        Tuple of (centers, radii, sum_of_radii)
        centers: np.array of shape (26, 2) with (x, y) coordinates
        radii: np.array of shape (26) with radius of each circle
        sum_of_radii: Sum of all radii
```

```
Output the executable Python code to accomplish the task inside a markdown code block.
```

Listing 11: **Circle Packing** ($n = 26$) discovery instruction (Sharma, 2025). (https://github.com/algorithmicsuperintelligence/openevolve/blob/main/examples/circle_packing/initial_program.py)

```
Craft a prompt prefix to boost a downstream {LLM_name}'s score on the
{benchmark_name} benchmark.
```

Listing 12: **Prompt Optimization (GSM8K)** discovery instructions.

F Initial Seed Discoveries

```python
import math
from sky_spot.strategies.strategy import Strategy
from sky_spot.utils import ClusterType

class EvolveSingleRegionStrategy(Strategy):
    NAME = 'evolve_single_region'

    def __init__(self, args):
        super().__init__(args)

    def reset(self, env, task):
        super().reset(env, task)

    def _step(self, last_cluster_type: ClusterType, has_spot: bool) -> ClusterType:
        env = self.env

        remaining_task_time = self.task_duration - sum(self.task_done_time)
        if remaining_task_time <= 1e-3:
            return ClusterType.NONE

        remaining_time = self.deadline - env.elapsed_seconds

        if remaining_task_time + self.restart_overhead >= remaining_time:
            return ClusterType.ON_DEMAND

        if has_spot:
            return ClusterType.SPOT
        else:
            return ClusterType.NONE

    @classmethod
    def _from_args(cls, parser):
        args, _ = parser.parse_known_args()
        return cls(args)
```

Listing 13: **Can't Be Late** initial seed discovery from (Agrawal et al., 2026a). (https://github.com/gepa-ai/gepa/tree/main/examples/adrs/can_be_late)

```python
import networkx as nx
import pandas as pd
import os
from typing import Dict, List


class SingleDstPath(Dict):
    partition: int
    edges: List[List]  # [[src, dst, edge data]]


class BroadCastTopology:
    def __init__(self, src: str, dsts: List[str], num_partitions: int = 4, paths: Dict[str, 'SingleDstPath'] =
       None):
        self.src = src
        self.dsts = dsts
        self.num_partitions = num_partitions
        if paths is not None:
            self.paths = paths
        else:
            self.paths = {dst: {str(i): None for i in range(num_partitions)} for dst in dsts}

    def get_paths(self):
        return self.paths

    def set_num_partitions(self, num_partitions: int):
        self.num_partitions = num_partitions

    def set_dst_partition_paths(self, dst: str, partition: int, paths: List[List]):
        partition = str(partition)
        self.paths[dst][partition] = paths
```

```
    def append_dst_partition_path(self, dst: str, partition: int, path: List):
        partition = str(partition)
        if self.paths[dst][partition] is None:
            self.paths[dst][partition] = []
        self.paths[dst][partition].append(path)


def search_algorithm(src, dsts, G, num_partitions):
    \"\"\"
    Find broadcast paths from source to all destinations.

    Uses Dijkstra's shortest path algorithm based on cost as the edge weight.

    Args:
        src: Source node identifier (e.g., "aws:ap-northeast-1")
        dsts: List of destination node identifiers
        G: NetworkX DiGraph with cost and throughput edge attributes
        num_partitions: Number of data partitions

    Returns:
        BroadCastTopology object with paths for all destinations and partitions
    \"\"\"
    h = G.copy()
    h.remove_edges_from(list(h.in_edges(src)) + list(nx.selfloop_edges(h)))
    bc_topology = BroadCastTopology(src, dsts, num_partitions)

    for dst in dsts:
        path = nx.dijkstra_path(h, src, dst, weight="cost")
        for i in range(0, len(path) - 1):
            s, t = path[i], path[i + 1]
            for j in range(bc_topology.num_partitions):
                bc_topology.append_dst_partition_path(dst, j, [s, t, G[s][t]])

    return bc_topology
```

Listing 14: **CloudCast** initial seed discovery from (Agrawal et al., 2026a). (https://github.com/gepa-ai/gepa/tree/main/examples/adrs/cloudcast)

```
<trace>
3,7,83,48,87,19,59,65,28,36,35,38,25,55,27,17,22,93,
69,67,47,77,95,74,51,97,84,20,11,33,61,18,24,12,6,21,
81,72,79,49,39,9,30,23,66,90,52,44,2,32,37,15,99,80,
54,71,40,42,41,88,56,8,16,5,73,60,43,63,96,64,46,82,
31,0,94,68,53,85,75,89,10,98,4,78,70,34,14,86,29,91,
62,57,13,50,76,26,1,45,92,58
</trace>
```

Listing 15: **Traveling Salesperson** ($n = 100$) initial seed discovery is a randomly initialized path as per (Yang et al., 2024). (https://github.com/google-deepmind/opro/blob/main/opro/optimization/optimize_tsp.py)

```
import numpy as np


def run_packing():
    """
    Construct a specific arrangement of 26 circles in a unit square
    that attempts to maximize the sum of their radii.

    Returns:
        Tuple of (centers, radii, sum_of_radii)
        centers: np.array of shape (26, 2) with (x, y) coordinates
        radii: np.array of shape (26) with radius of each circle
        sum_of_radii: Sum of all radii
    """
    # Initialize arrays for 26 circles
    n = 26
    centers = np.zeros((n, 2))

    # Place circles in a structured pattern
    # This is a simple pattern - evolution will improve this

    # First, place a large circle in the center
    centers[0] = [0.5, 0.5]

    # Place 8 circles around it in a ring
    for i in range(8):
        angle = 2 * np.pi * i / 8
        centers[i + 1] = [0.5 + 0.3 * np.cos(angle), 0.5 + 0.3 * np.sin(angle)]

    # Place 16 more circles in an outer ring
    for i in range(16):
        angle = 2 * np.pi * i / 16
        centers[i + 9] = [0.5 + 0.7 * np.cos(angle), 0.5 + 0.7 * np.sin(angle)]

    # Additional positioning adjustment to make sure all circles
```

```
    # are inside the square and don't overlap
    # Clip to ensure everything is inside the unit square
    centers = np.clip(centers, 0.01, 0.99)

    # Compute maximum valid radii for this configuration
    radii = compute_max_radii(centers)

    # Calculate the sum of radii
    sum_radii = np.sum(radii)

    return centers, radii, sum_radii


def compute_max_radii(centers):
    """
    Compute the maximum possible radii for each circle position
    such that they don't overlap and stay within the unit square.

    Args:
        centers: np.array of shape (n, 2) with (x, y) coordinates

    Returns:
        np.array of shape (n) with radius of each circle
    """
    n = centers.shape[0]
    radii = np.ones(n)

    # First, limit by distance to square borders
    for i in range(n):
        x, y = centers[i]
        # Distance to borders
        radii[i] = min(x, y, 1 - x, 1 - y)

    # Then, limit by distance to other circles
    # Each pair of circles with centers at distance d can have
    # sum of radii at most d to avoid overlap
    for i in range(n):
        for j in range(i + 1, n):
            dist = np.sqrt(np.sum((centers[i] - centers[j]) ** 2))

            # If current radii would cause overlap
            if radii[i] + radii[j] > dist:
                # Scale both radii proportionally
                scale = dist / (radii[i] + radii[j])
                radii[i] *= scale
                radii[j] *= scale

    return radii
```

Listing 16: **Circle Packing** ($n = 26$) initial seed discovery from (Sharma, 2025). (https://github.com/algorithmicsuperintelligence/openevolve/blob/main/examples/circle_packing/initial_program.py)

```
'placeholder prompt prefix'
```

Listing 17: **Prompt Optimization (GSM8K)** initial seed discovery is a placeholder string.