Self-Improving Language Models with Bidirectional Evolutionary Search

Guowei Xu [1] **Zhenting Qi** [1] **Huangyuan Su** [1] **Weirui Ye** [2]
Himabindu Lakkaraju [1] **Sham M. Kakade** [1] **Yilun Du** [1]

Abstract

Search has been proposed as an effective method for self-improving language models and agentic systems, both for post-training sample generation and for inference. However, widely used methods such as best-of-N sampling and tree search face two fundamental limitations: they are guided by sparse verification signals, and they construct candidates primarily through autoregressive expansion, restricting exploration to regions with substantial model probability mass. To address these, we propose Bidirectional Evolutionary Search (BES), a search framework that couples forward candidate evolution with backward goal decomposition. In the forward search, BES augments standard expansion with evolution operators that recombine partial trajectories to generate candidates that are difficult to obtain from a single model rollout. In the backward search, BES recursively decomposes the original task into checkable subgoals, producing dense intermediate feedback that guides forward search. We provide theoretical motivation showing that candidates generated by expansion-only search are confined to a narrow entropy shell while evolutionary operators can escape it, and that backward search can exponentially reduce the number of required samples to find a correct answer. Experiments show that on challenging post-training tasks where mainstream post-training algorithms fail to improve, BES enables consistent gains, and on three open problem solving benchmarks at inference time, BES outperforms existing open-source frameworks in both average and best-case performance.

[1]Harvard University [2]MIT. Correspondence to: Yilun Du <ydu@seas.harvard.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

Large language models (LLMs) and agentic systems have demonstrated remarkable capabilities on complex reasoning problems (Wei et al., 2022; Qi et al., 2025; Guo et al., 2025). They can even solve open problems across mathematical and scientific domains (Novikov et al., 2025; Hubert et al., 2026) and surpass the best human performance on tasks such as code generation (Li et al., 2022; Yuksekgonul et al., 2026). In this context, the question of how to do better sampling from LLMs and agentic systems is of critical importance (Guan et al., 2025). This is particularly significant for problems at the frontier of model capability, where naive sampling methods may require too many samples to obtain a correct answer or may simply fail (Glazer et al., 2025; Yue et al., 2026). At training time, higher-quality samples enable more effective post-training and self-improvement (Zelikman et al., 2022; Yuan et al., 2024); at inference time, they serve as a natural mechanism for test-time scaling (Zelikman et al., 2024; Wu et al., 2025; Lange et al., 2025), which can further push the boundary of what models can achieve.

Currently, the two dominant sampling methods in post-training, self-improvement, and inference for LLMs and agentic systems are best-of-N sampling and tree search. Best-of-N sampling is simple and efficient. For problems of moderate difficulty, it typically suffices to find high-quality responses and is therefore widely adopted in post-training algorithms such as GRPO (Guo et al., 2025) and its variants (Yu et al., 2025), while also serving as a strong baseline for inference. Tree search methods such as beam search and Monte Carlo Tree Search (Kocsis and Szepesvari, 2006) can discover better responses more sample-efficiently than best-of-N on harder problems. For example, Tree-GRPO leverages tree search for sample generation during post-training (Ji et al., 2026), and Tree of Thoughts explores multiple reasoning paths at inference time (Yao et al., 2023).

However, both methods share two fundamental limitations. (1) The verification signal to guide the search is sparse. Effective search depends critically on the accuracy and granularity of the verifier, yet in common settings such as RLVR post-training, verifiers typically provide only coarse-grained feedback. (2) They struggle to generate candidates beyond the model's own distribution. They construct candidates by

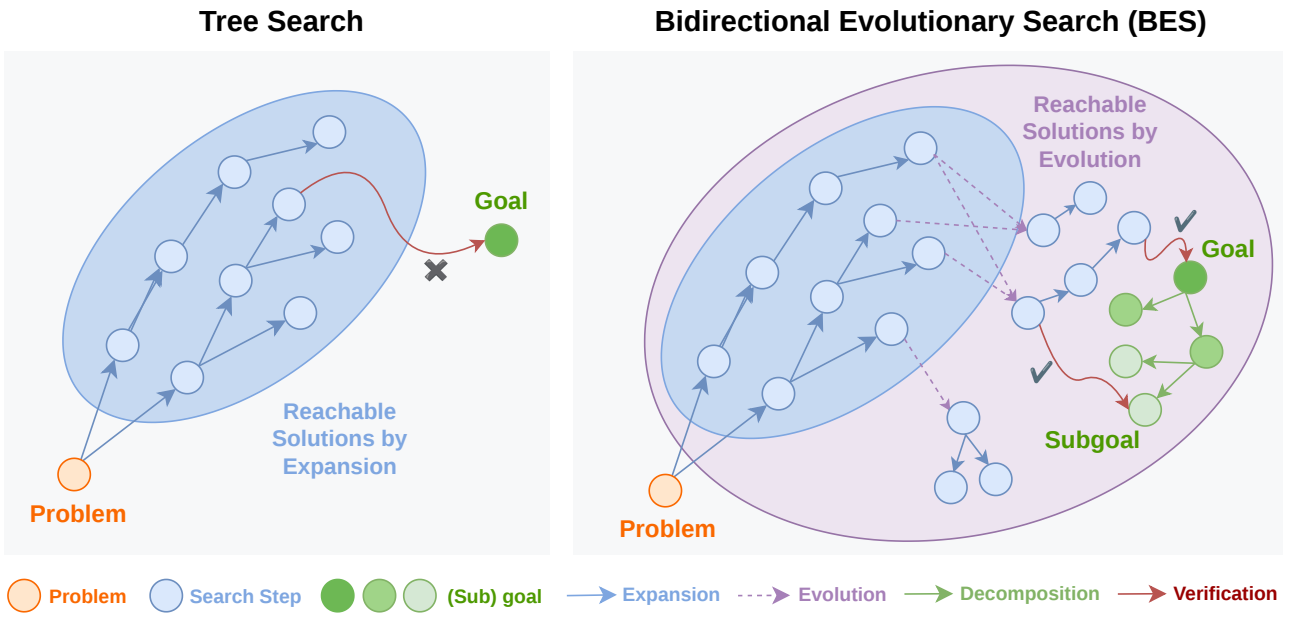

Figure 1. Comparison of tree search and Bidirectional Evolutionary Search (BES). **Left:** Tree search constructs candidates by sequentially expanding steps. We prove that all such candidates are confined to a narrow entropy shell (Theorem 4.4a), limiting exploration to a small region of the solution space. **Right:** BES escapes this shell through evolution operators that recombine parts of different trajectories, with backward search decomposing the problem into verifiable sub-goals that provide dense feedback to guide the forward search toward the final goal. ✓ and ✗ indicate whether a candidate satisfies or fails the (sub-)goal, respectively.

auto-regressively extending responses. This confines candidates to the support of the model's own distribution (Brown et al., 2024), making it difficult to reach low-probability regions where correct solutions often reside on hard problems.

To address these two limitations, we propose Bidirectional Evolutionary Search (BES). First, to tackle the sparsity of verification signals, we introduce bidirectional search: the forward search seeks better candidate solutions, while the backward search discovers finer-grained sub-goals to verify them. Second, to generate candidates beyond the model's own distribution, we draw inspiration from evolutionary biology. For much of the history of life, organisms reproduced asexually. Each offspring was a direct extension of its parent, and beneficial mutations arising independently in different individuals could never be combined (Fisher, 1999). Sexual reproduction fundamentally changed this through chromosomal recombination: gene segments from different lineages are spliced together to produce novel combinations that neither parent possessed (Muller, 1932). Analogously, in our setting, rather than only extending responses auto-regressively, we introduce four evolution operators: combination, translocation, deletion, and crossover. Combination, translocation, and crossover merge the strengths of two distinct responses in different ways to construct a new candidate, while deletion removes the least sound segment from a response. We theoretically prove that responses generated by expansion-only search are confined to a narrow entropy shell, while evolution operators can escape it.

We evaluate BES on both post-training and inference across LLM and agent settings. For post-training, we test on challenging logical reasoning and multi-hop reasoning tasks where mainstream post-training algorithms such as GRPO (Guo et al., 2025), MaxRL (Tajwar et al., 2026), and Tree-GRPO (Ji et al., 2026) struggle to find sufficient high-quality training samples and consequently fail to improve the base model. In contrast, BES consistently discovers effective training samples. For inference, we test on three open problem solving benchmarks, where BES discovers higher-quality solutions compared to existing open-source frameworks, including OpenEvolve (Sharma, 2025), GEPA (Agrawal et al., 2026), and ShinkaEvolve (Lange et al., 2025). We also provide ablation studies validating the contribution of each component, case studies visualizing search process, and cost analysis comparing wall time and API cost of BES against baselines.

2. Preliminaries

We consider reasoning problems of the form $\mathcal{T} = (x, V)$, where x is a problem description (e.g. a math question) and $V(x, y) \in [0, 1]$ is a *verifier* that assigns a score measuring how well a trajectory y solves the problem x.

Given a policy $\pi_\theta(\cdot \mid x)$ (e.g. an LLM that generates reasoning traces, or an agent that interacts with an environment through a sequence of actions), our goal is to produce a terminal response y that maximizes the verifier score. Let

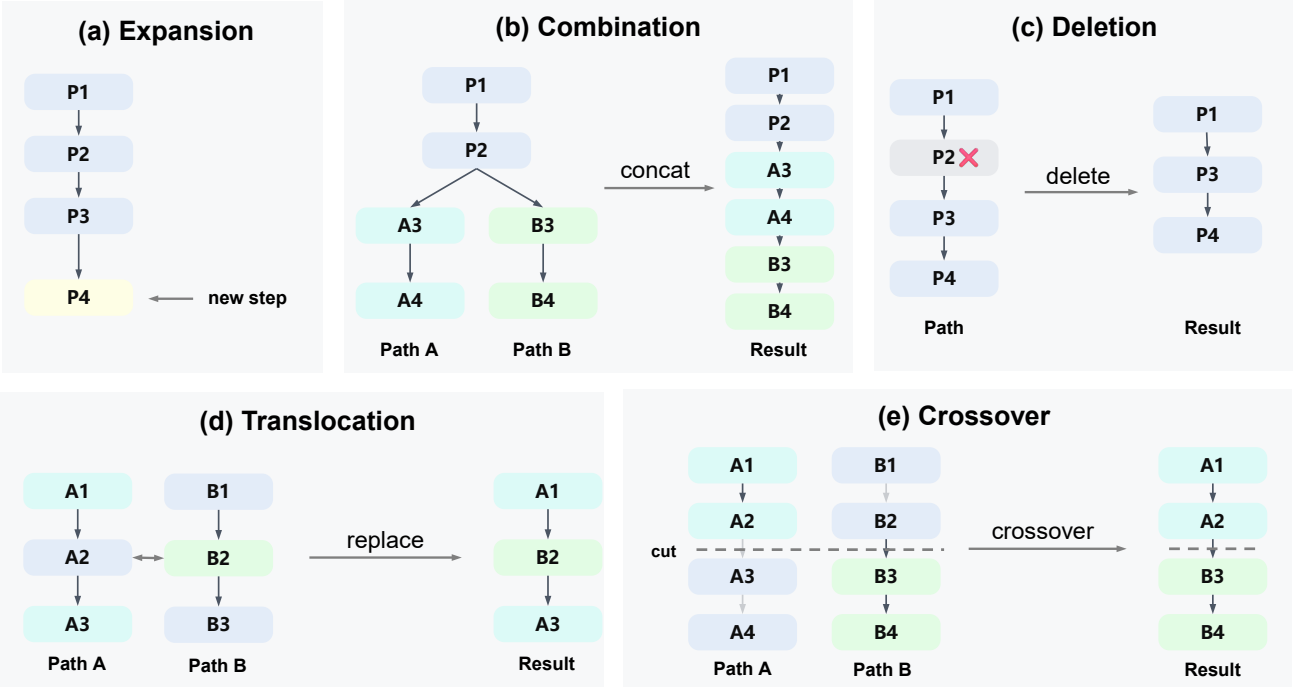

Figure 2. Forward search operators. **(a) Expansion:** the policy generates new steps (yellow). **(b) Combination:** two trajectories sharing a common prefix (P1–P2) have their distinct suffixes concatenated into a single candidate. **(c) Deletion:** an interior step (P2) is removed. **(d) Translocation:** one step in Path A (A2) is replaced by a step from Path B (B2). **(e) Crossover:** Path A is cut at a splice point and its tail is replaced by the tail of Path B.

$\mathcal{Y}_{\text{term}}(x)$ denote the set of valid terminal responses under the task specification and resource budget. We write the target as

$$y^\star(x) \in \arg\max_{y \in \mathcal{Y}_{\text{term}}(x)} V(x, y). \tag{1}$$

Finding $y^\star$ is important in both training and inference. During training, high-quality candidates can facilitate post-training or iterative self-improvement. During inference, $y^\star$ is the response we seek.

The challenge is that the maximization in Eq. (1) is intractable: on hard problems, the probability mass that π_θ assigns to correct trajectories can be extremely small. Practical algorithms therefore approximate $y^\star$ by searching over a set of candidates. We introduce two methods below.

Best-of-N sampling. The simplest approach is to draw N independent trajectories and return the one with the highest score. Formally, let $y^{(1)}, \ldots, y^{(N)} \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid x)$. The method returns $y^{\text{BoN}} = \arg\max_{y \in \{y^{(1)}, \ldots, y^{(N)}\}} V(x, y)$. Best-of-$N$ is parallel and requires no structural knowledge of the problem. Its effectiveness, however, is bounded by the finite-sample coverage of π_θ: all N trajectories are drawn from the same distribution, so if the optimal trajectory lies in a region with very small policy mass, increasing N gives only linear coverage improvement.

Tree search. Tree-search methods exploit the sequential structure of a trajectory. A trajectory is decomposed into steps (e.g. reasoning segments or agent actions), and the search maintains a tree whose nodes are partial trajectories and whose edges correspond to appending a step. Branches are selected and extended according to a heuristic value so that compute is concentrated on prefixes that look promising. Classic methods include beam search, best-first search, and Monte Carlo Tree Search (Kocsis and Szepesvari, 2006); recent work applies these ideas to LLM and agent reasoning (Yao et al., 2023; Ji et al., 2026). Tree search can be more sample-efficient than best-of-N when the per-step signal is informative, but it still materializes terminal candidates one lineage at a time through sequential extension.

3. BES: Bidirectional Evolutionary Search

BES performs a bidirectional evolutionary search that alternates between two coupled processes: a *forward* search that seeks better candidates, and a *backward* search that decomposes the problem into sub-goals to evaluate each forward node. The forward search not only extends trajectories but also recombines parts of different candidates, producing solutions that no single rollout from π_θ would likely reach. The backward search provides dense scores for partial trajectories, guiding the forward search toward promising candidates. In practice, one backward search step is performed after every several forward search steps. The full pseudocode is given in Appendix A, and a case study illustrating the search process is provided in Appendix E.

3.1. Forward Search: Expanding the Reachable Solution Space

Each candidate partial trajectory is a node $n = (y_1, \ldots, y_t)$, where y_i denotes the i-th step (e.g. a reasoning segment or an action). The search maintains a candidate set $\mathcal{P}$ of such nodes, and at each step applies one of two types of operators, *expansion* or *evolution*, to produce a child node n', scored by the backward search (Section 3.2) and added to $\mathcal{P}$.

Expansion. Expansion extends a parent node by sampling new steps. Given $n = (y_1, \ldots, y_t)$, we sample a step count $K \sim \mathrm{Uniform}\{1, \ldots, K_{\max}\}$ and draw up to K new steps from π_θ:

$$y_{t+k} \sim \pi_\theta\big(\cdot \mid x \oplus y_1 \oplus \cdots \oplus y_{t+k-1}\big), \qquad k = 1, \ldots, K, \tag{2}$$

Evolution. A key limitation of expansion alone is that each candidate is built by sequentially extending a single trajectory, so it cannot combine useful parts from different candidates. Evolution operators overcome this by editing and recombining existing trajectories. As illustrated in Figure 2, we define four operators inspired by biological evolution: *(i) Combination* merges two trajectories by concatenating their suffixes beyond a shared prefix; *(ii) Deletion* removes one interior step, producing a shorter candidate; *(iii) Translocation* transplants a single step from one trajectory into another; and *(iv) Crossover* splices the prefix of one trajectory onto the tail of another. Formal definitions are provided in Appendix B. Together, these four evolution operators allow the search to restructure, condense, and recombine existing trajectories. As with mutations in nature, evolution operations are not guaranteed to produce better candidates. However, the value of evolution lies in generating diverse candidates: ***even if only a small fraction turn out to be improvements, that is sufficient for the search to make progress.*** Here, the four evolution operators are defined as direct edits on the step sequence. In settings where direct concatenation is not well-defined, the evolution operators can alternatively be implemented by prompting the policy model.

At each forward search step, we select among the operators with fixed probabilities. All operators require selecting one or two parent nodes from $\mathcal{P}$. Let $\mathcal{C}_t$ be the set of eligible non-terminal members of $\mathcal{P}$ at step t. For single-parent operators (expansion, deletion), we sample the parent from a Boltzmann distribution over backward score $s(n)$ (Eq. 5):

$$\Pr\big[n \mid \mathcal{C}_t\big] \;=\; \frac{\exp(\tilde{s}(n)/\tau_t)}{\sum_{n' \in \mathcal{C}_t} \exp(\tilde{s}(n')/\tau_t)}, \tag{3}$$

where $\tau_t > 0$, $\deg(n)$ is the number of children of n in $\mathcal{P}$, and the indicator term $\tilde{s}(n) \;=\; s(n) + \lambda \cdot \mathbf{1}\big[\deg(n) = 0\big]$ adds a small constant bonus λ (we use $\lambda = 0.1$) to candidates that have not yet been selected as parents, giving unexplored nodes a higher chance of being expanded.

For two-parent operators (combination, translocation, crossover), we select a pair of parent nodes (n_a, n_b) from $\mathcal{C}_t$. We calculate a pair score $s(n_a, n_b)$ (Eq. 6) that favors complementary parents whose strengths cover different parts of the problem. The pair is then drawn from the analogous Boltzmann distribution:

$$\Pr[(n_a, n_b) \mid \mathcal{C}_t] \;=\; \frac{\exp(s(n_a, n_b)/\tau_t)}{\sum_{n_a, n_b \in \mathcal{C}_t} \exp(s(n_a, n_b)/\tau_t)}. \tag{4}$$

The temperature τ_t is annealed linearly from an initial value τ_0 to a final value $\tau_{\text{end}} < \tau_0$ over the search budget, gradually shifting from exploration to exploitation.

3.2. Backward Search: Better Verification through Goal Decomposition

While the verifier V provides a score for each node in the forward search, this signal is relatively sparse. Backward search addresses this by decomposing the problem into a tree of sub-goals. Each forward node is then scored against this tree: the more sub-goals a candidate trajectory has addressed, the higher its score. This gives the forward search a dense signal to select promising candidates, even when none of them have fully solved the problem yet.

Below we describe how the backward search is constructed. Starting from the top-level goal $g_{\texttt{root}}$ (i.e. solving the entire problem), the policy π_θ is prompted to break each goal into finer sub-goals, producing a rooted backward goal tree. Each goal g on the tree can be recursively split into children $\mathrm{ch}(g)$ (finer sub-goals), and every g comes with a verifier $V_g(x, n) \in [0, 1]$ that tests how well a candidate node n addresses the sub-goal g on problem x. For the top-level goal, $V_{g_{\texttt{root}}} = V$ (the verifier of the original problem). This decomposition is re-invoked every K forward search steps: at each invocation, we select a leaf sub-goal that no current candidate fully satisfies and prompt π_θ to split it into finer sub-goals. After that, all existing forward nodes are re-scored. The verifiers are task-dependent and can be instantiated as rule-based checkers, test-case code executors, embedding similarity models, or LLM judgers. We describe the verifiers used in each experiment in Appendix D.

For a candidate node n from the forward search (Section 3.1) and a sub-goal g, define the sub-goal score

$$s(n, g) \;=\; \alpha \cdot V_g(x, n) \;+\; (1-\alpha) \cdot \frac{1}{|\mathrm{ch}(g)|} \sum_{g' \in \mathrm{ch}(g)} s(n, g'), \tag{5}$$

where $\alpha \in [0, 1]$ balances the contribution of coarser parent goals and finer-grained sub-goals to the overall score. For leaf sub-goals ($\mathrm{ch}(g) = \emptyset$), we set $s(n, g) = V_g(x, n)$. If a goal is already fully satisfied ($V_g(x, n) = 1$), we short-circuit to $s(n, g) = 1$ without evaluating its children. The overall score is $s(n) \triangleq s(n, g_{\texttt{root}})$.

For two candidate nodes n_a and n_b, we further define a pair score that measures their joint coverage of the goal tree. Replacing each sub-goal verifier with the maximum of the two parents' scores gives

$$s(n_a,n_b,g) = \alpha \cdot \max\{V_g(x,n_a),\, V_g(x,n_b)\} + (1-\alpha) \cdot \frac{1}{|\mathrm{ch}(g)|} \sum_{g' \in \mathrm{ch}(g)} s(n_a,n_b,g'), \quad (6)$$

with $s(n_a,n_b) \triangleq s(n_a,n_b,g_{\texttt{root}})$. Since a sub-goal is covered if either parent solves it, pair score favors complementary parents that cover different parts of the goal tree.

Applying this to every forward search node yields a backward score for each node and a pair score for each pair of nodes, which directly drives forward node selection.

3.3. Using BES for Post-Training and Inference

Post-Training. BES replaces sample-generation of post-training. Given a problem, BES returns a set of high-quality trajectories from the forward search candidate set $\mathcal{P}$. These trajectories are then used as training data for post-training algorithms, replacing the candidates that would otherwise be obtained from i.i.d. rollouts. Because BES can find correct solutions that single rollouts rarely reach, it provides stronger training signal, especially on hard problems.

Inference. At inference time, BES runs on open problems with a fixed compute budget, and the terminal trajectory with the highest score is returned as the final answer.

4. Theoretical Motivations

4.1. Theoretical Motivation for Evolution Operators

We justify why evolution operators provide an advantage over expansion-only search. We prove that expansion-only candidates are confined to an entropy shell, while evolution operators can escape it. Full proof is given in Appendix C.1.

Fix a task x and a step horizon T. A trajectory $Y = (y_1,\dots,y_T)$ is generated by the policy with probability $P(Y) = \prod_{t=1}^{T} P(y_t \mid y_{<t})$. Let $H_T = H_P(Y)$ denote the trajectory-level entropy. We partition the trajectory into k contiguous blocks $U_1,\dots,U_k$. We make three assumptions on the policy. All of them are naturally satisfied in practice; see Appendix C.1.1 for a detailed discussion.

Assumption 4.1 (Bounded per-step surprise)**.** *There exists $L < \infty$ such that for every reachable prefix and every step with positive probability, $-\log P(v \mid y_{<t}) \le L$.*

Assumption 4.2 (Decaying step dependence)**.** *There exists a summable nonnegative sequence $(\beta_\ell)_{\ell\ge1}$ with $C_\beta := \sum_{\ell\ge1}\beta_\ell < \infty$ such that for every $s < t$, prefix $y_{<s}$, and two steps v, v' at position s,*

$$\Big|\mathbb{E}\big[h_t(Y_{<t}) \mid Y_{<s} = y_{<s}, Y_s = v\big] - \mathbb{E}\big[h_t(Y_{<t}) \mid Y_{<s} = y_{<s}, Y_s = v'\big]\Big| \le \beta_{t-s}. \quad (7)$$

Assumption 4.3 (Linear block total correlation)**.** *We partition the trajectory into $k \ge 2$ contiguous blocks $U_1,\dots,U_k$ at fixed boundaries $0 = s_0 < s_1 < \cdots < s_k = T$. The block total correlation grows linearly in the horizon: $\sum_{j=1}^{k} H_P(U_j) - H_P(U_1,\dots,U_k) \ge \gamma T$ for some constant $\gamma > 0$.*

Theorem 4.4 (Shell confinement and escape)**.** *Under Assumptions 4.1–4.3, define the typical set $A_\epsilon^{(T)} := \{y : |-\log P(y) - H_T| \le \epsilon T\}$.*

(a) Shell confinement. Every trajectory $Y \sim P$ produced by expansion-only search satisfies $\Pr[Y \notin A_\epsilon^{(T)}] \le \exp(-\Omega(T))$. That is, search is confined to a typical set of size at most $\exp(H_T + \epsilon T)$.

(b) Shell escape. Let $Q = \bigotimes_{j=1}^{k} P_j$ be the k-time evolution distribution. For any $\epsilon < \gamma$,

$$\mathbb{E}_Q[-\log P(\widetilde{Y})] \;\ge\; H_T + \gamma T \;>\; H_T + \epsilon T, \quad (8)$$

so evolution candidates have expected log-probability strictly beyond the shell boundary. Moreover,

$$\Pr_Q\Big[\widetilde{Y} \in A_\epsilon^{(T)}\Big] \;\le\; 1 - \frac{(\gamma-\epsilon)T}{LT - H_T - \epsilon T} \;<\; 1, \quad (9)$$

confirming that a positive fraction of evolution candidates escape the shell.

Proof sketch. We first establish that, with high probability, every policy rollout has log-probability close to H_T, confining expansion-only search to a thin entropy shell. We then show that evolution operators break inter-block dependence and result in increased surprise (Lemma C.6). Generalizing to k-time evolution, the expected surprise exceeds H_T by the block total correlation (Lemma C.7), which is $\Omega(T)$ under Assumption 4.3, pushing evolution candidates outside.

4.2. Theoretical Motivation for Bidirectional Search

The previous section shows that evolution operators can construct candidates beyond the policy's entropy shell. We now show that backward search makes this enlarged space efficiently searchable. The full proof is given in Appendix C.2.

Let the backward search produce m leaf sub-goals $\{g_1,\dots,g_m\}$. For a candidate trajectory n, define $C_i(n) = \mathbf{1}\{V_{g_i}(x,n) = 1\}$. We assume that terminal success requires all leaf sub-goals: $V(x,n) = 1 \Rightarrow C_i(n) = 1,\ \forall i \in [m]$. For simplicity, suppose that for a fresh candidate n, the events $\{C_i(n) = 1\}$ are independent with probabilities $\Pr[C_i(n) = 1] = p_i$.

Theorem 4.5 (Exponential advantage from backward sub–goal signals)**.** *Let N candidates be sampled independently. Terminal-only search requires $N_{\text{term}} = \Omega(1/\prod_{i=1}^{m} p_i)$ candidates to obtain constant success probability. By contrast, backward-guided bidirectional search requires only $N_{\text{bidir}} = O\big(p_{\min}^{-1}\log(m/\delta)\big)$, where $p_{\min} = \min_i p_i$, to collect evidence for all sub-goals with probability at least $1-\delta$. In the symmetric case $p_i = p$, the ratio is $N_{\text{term}}/N_{\text{bidir}} = \Omega\big(p^{-(m-1)}/\log(m/\delta)\big)$, which is exponential in the number of sub-goals m.*

5. Experiments

We evaluate BES on both post-training and inference across LLM and agent settings. For post-training, we consider Logical Reasoning (LLM) and Multi-Hop Reasoning (Agent). For inference, we consider three open problem solving benchmarks: Circle Packing (Square), Circle Packing (Rectangle), and the Heilbronn Convex problem. In each benchmark, we compare BES against commonly used baselines and show that BES consistently achieves better sampling.

5.1. Bidirectional Evolutionary Search for Post-Training

5.1.1. Logical Reasoning

For logical reasoning, we use the Knights-and-Knaves dataset (Xie et al., 2024a). In this task, each puzzle involves a group of people who are either a knight or a knave, and the solver must deduce each one's identity.

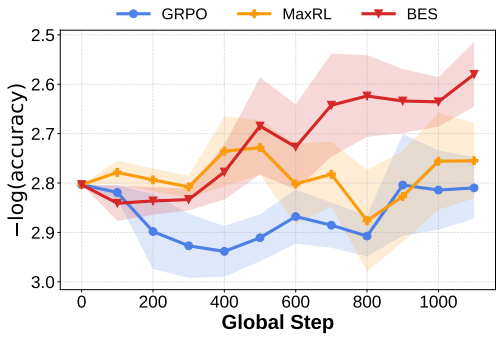

Figure 3. EMA-smoothed validation accuracy on logical reasoning.

Baselines. We select two algorithms, GRPO (Guo et al., 2025) and MaxRL (Tajwar et al., 2026), as baselines, as we apply BES on top of MaxRL. We note that BES is a sampling algorithm that is agnostic to the training method and can be applied on top of any post-training algorithm.

Training Setup. We use Gemma-3-1B-it (Team et al., 2025) as the base model. To adapt the model to the Knights-and-Knaves benchmark, we first perform a cold start by fine-tuning on 1K SFT examples generated using the official data generation pipeline for 3 epochs, followed by 4 epochs of post-training on 5K problems, during which we compare the different methods on a 1.3K validation set. Detailed experimental configurations are provided in Appendix D.1.

Results. As shown in Figure 3, due to the difficulty of training set, GRPO and MaxRL show little to no improvement during training, while BES steadily improves on the validation set throughout training, demonstrating that BES is more effective at discovering high-quality training samples.

5.1.2. Multi-Hop Reasoning

For multi-hop reasoning, we use the MuSiQue dataset (Trivedi et al., 2022). In this task, the agent must answer complex questions that require retrieving and integrating information across multiple documents, where no single document contains the complete answer.

Table 1. Multi-hop reasoning post-training results on MuSiQue with Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct. We report accuracy, number of valid searches, and finish ratio. Higher is better for all metrics. **Bold** denotes the best performing method.

Method	Accuracy (%, ↑)	# Valid Search (↑)	Finish Ratio (↑)
Llama-3.2-3B-Instruct			
Base model	4.0	–	–
+ GRPO	2.1 (-1.9)	0.84	0.64
+ Tree-GRPO	3.9 (-0.1)	1.50	0.64
+ BES	**7.0 (+3.0)**	**2.31**	**0.97**
Llama-3.1-8B-Instruct			
Base model	6.6	–	–
+ GRPO	5.6 (-1.0)	1.46	0.37
+ Tree-GRPO	7.4 (+0.8)	0.65	0.71
+ BES	**10.4 (+3.8)**	**2.11**	**0.94**

Baselines. We compare against GRPO (Guo et al., 2025) and Tree-GRPO (Ji et al., 2026), with the latter being the current state-of-the-art method for search agent post-training. We apply BES on top of GRPO.

Training Setup. We adopt the training setup of Tree-GRPO (Ji et al., 2026), using Llama-3.2-3B-Instruct and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) as base models. During the search process, the agent can perform multiple search actions before producing a final answer. Search results are provided by an offline Wikipedia server. We use the 3–4 hop solvable subset of the MuSiQue training set as our training data and train for 2 epochs, as additional epochs lead to training collapse. We evaluate on the full official MuSiQue validation set. Detailed configurations are provided in Appendix D.2.

Results. As shown in Table 1, GRPO consistently degrades from the base model on both scales, likely due to reward hacking where the model learns to skip search actions and guess directly, as reflected by its low number of valid searches. Tree-GRPO provides marginal improvement on the 8B model (+0.8%) but fails to improve the

Table 2. Open problem solving benchmarks with GPT-5 as the backbone model. We report **Mean** ± **Std** and **Best** objective values (↑). **Bold** denotes the best performing method.

Strategy	Circle Packing (Square)		Circle Packing (Rect.)		Heilbronn (Convex)	
	Avg.	Best	Avg.	Best	Avg.	Best
Human & High-compute closed-source framework						
Human	–	2.634	–	2.364	–	0.0306
AlphaEvolve	–	2.635	–	2.3658	–	0.0309
Open-source frameworks						
OpenEvolve	2.531 ± .018	2.541	2.267 ± .014	2.276	0.025 ± .005	**0.027**
GEPA	2.613 ± .022	2.628	2.326 ± .023	2.354	0.025 ± .002	**0.027**
Base framework						
ShinkaEvolve	2.464 ± .083	2.541	2.335 ± .026	2.358	0.023 ± .005	0.026
Ours						
BES	**2.623** ± .014	**2.632**	**2.349** ± .012	**2.360**	**0.026** ± .001	**0.027**

3B model. In contrast, BES achieves substantial gains on both scales (+3.0% on 3B, +3.8% on 8B), outperforming all baselines by a wide margin. Beyond accuracy, BES also produces agents with significantly more valid search actions and higher finish ratios, indicating that the trained agents learn to actively search rather than randomly guessing.

5.2. Bidirectional Evolutionary Search for Inference

At inference time, we evaluate the ability of BES to solve open problems. Specifically, we consider three benchmarks: Circle Packing (Square), which seeks to pack N circles into a unit square with maximum radius; Circle Packing (Rect), the analogous problem for a rectangular container; and Heilbronn (Convex), which seeks to place N points in the unit square to maximize the minimum area of any convex polygon formed by subsets of the points.

Baselines. We compare against three open-source frameworks: OpenEvolve (Sharma, 2025), GEPA (Agrawal et al., 2026), and ShinkaEvolve (Lange et al., 2025). All baseline results are taken from SkyDiscover (Liu et al., 2026), which uses the same backbone model, compute budget, and configuration as ours. We apply BES on top of ShinkaEvolve. For reference, we also include the results of human experts and AlphaEvolve (Novikov et al., 2025), which is closed-source and uses significantly more compute.

Setup. ShinkaEvolve maintains a population of candidate programs and iteratively proposes modifications. Since directly concatenating model outputs is not meaningful on this benchmark (where candidates are executable programs), we implement the evolution operators by prompting LLMs. We use GPT-5 as the backbone model. Detailed configurations are provided in Appendix D.3.

Results. As shown in Table 2, BES outperforms open-source frameworks on every benchmark. Notably, BES also exhibits much lower variance across runs, indicating the search is more stable and reliable. In Appendix G, we list a summary of the best programs discovered by BES.

5.3. Ablation Study

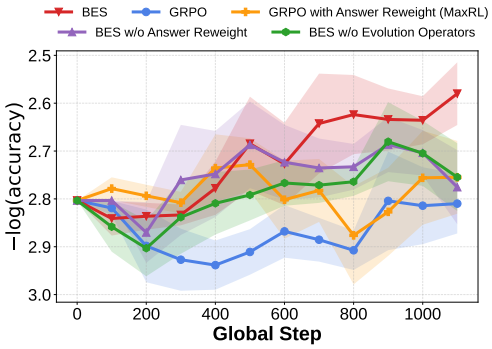

Figure 4. Ablation study on logical reasoning.

We conduct an ablation study on the Knights-and-Knaves benchmark. On this benchmark, BES combines bidirectional evolutionary search for discovering high-quality samples with MaxRL's answer reweighting for training. We therefore consider two ablations: (1) removing answer reweighting; and (2) removing the evolution operators to verify that both bidirectional search and evolution operators are necessary. As shown in Figure 4, both ablations underperform the full BES method, while still outperforming the GRPO and MaxRL baselines. This confirms the

effectiveness of each component in our approach.

5.4. Cost Analysis

Post-Training. To evaluate the cost-effectiveness of BES, we compare the wall-clock time and final accuracy across GRPO, Tree-GRPO, and BES when training Llama-3.2-3B-Instruct on the agentic search task. We report the median wall-clock time per step throughout the training process.

Table 3. Cost analysis for multi-hop reasoning post-training on Llama-3.2-3B-Instruct.

Method	Accuracy (%)	# Valid Search	Walltime (s)
GRPO	2.1 (-1.9)	0.84	64
Tree-GRPO	3.9 (-0.1)	1.50	240
BES	**7.0 (+3.0)**	**2.31**	309

Notably, the low wall-clock time of GRPO is misleading: during training, GRPO quickly exhibits reward hacking, with the model learning to skip search actions and guess answers directly, as reflected by its low number of valid searches. In contrast, BES incurs less than 30% additional overhead compared to Tree-GRPO, while achieving significantly better performance across all metrics.

Inference. We compare the API cost of BES against ShinkaEvolve across open problem solving benchmarks. As shown in Table 4, BES achieves consistently higher average values while incurring modest additional API costs.

Table 4. Cost analysis for open problem solving. We report average API cost per generation.

Method	Circle (Sq.)	Circle (Rect.)	Heilbronn
ShinkaEvolve	\$13.0	\$11.9	\$11.5
BES	\$18.6	\$14.0	\$13.7

6. Related Work

Self-Improvement for LLM and Agent. A growing line of work studies how language models and agents can improve using their own outputs to self-evolve (Huang et al., 2025; Kadlčík and Štefánik, 2024). STaR (Zelikman et al., 2022) bootstraps reasoning by filtering correct rationales for fine-tuning, while Self-Refine (Madaan et al., 2023) revises outputs at inference time through self-generated feedback. Self-Rewarding Language Models (Yuan et al., 2024) use the model itself as a judge for preference optimization. In agentic settings, Reflexion (Shinn et al., 2023) converts environmental feedback into verbal reflections, and Voyager (Wang et al., 2024) accumulates reusable skills through continual exploration. These methods typically refine individual trajectories or rely on the model to judge its own outputs. In contrast, BES treats self-improvement as a structured search problem, systematically discovering high-quality solutions that facilitate model self-improvement.

Search in LLM and Agent. Recent work applies search to both training and inference of LLMs and agents (Wang et al., 2025a; Romera-Paredes et al., 2024). On the training side, tree-structured exploration provides richer supervision and finer-grained credit assignment than standard rollout-based methods. Tree-GRPO (Ji et al., 2026) and TreeRL (Hou et al., 2025) integrate tree search directly into reinforcement learning, while ReST-MCTS* (Zhang et al., 2024), MCTS-DPO (Xie et al., 2024b), and rStar-Math (Guan et al., 2025) use search to bootstrap higher-quality training data for self-improvement. On the inference side, Tree of Thoughts (Yao et al., 2023), Graph of Thoughts (Besta et al., 2024), and RAP (Hao et al., 2023) established reasoning as an explicit search problem by expanding chain-of-thought into tree-structured exploration (Misaki et al., 2025). More recently, evolutionary approaches such as AlphaEvolve (Novikov et al., 2025), ShinkaEvolve (Lange et al., 2025), and ThetaEvolve (Wang et al., 2025b) tackle hard and even open problems (Pourcel et al., 2026) by maintaining candidate populations with LLM-driven mutations and external evaluation. However, these methods mainly rely on tree search and thus struggle to explore beyond the policy's own distribution.

Classical Search Methods. The recent surge of search methods in LLMs and agents draws on a long history of classical search. In symbolic planning and graph search, algorithms such as A* (Hart et al., 1968) and bidirectional search introduced core principles including heuristic guidance, frontier expansion, and search-space reduction. Branch-and-bound methods (Lawler and Wood, 1966), which prune subtrees whose bounds certify suboptimality, offer a natural analogy for modern methods that use verifiers to discard unpromising candidates early. Evolutionary methods, including genetic search (Holland, 1992; Banzhaf et al., 2000) and differential evolution (Storn and Price, 1997), optimize by iteratively refining candidate populations.

7. Conclusion

In this paper, we present BES, a bidirectional evolutionary search framework that addresses two fundamental limitations of existing search methods for LLMs and agents: sparse verification signals and confined candidate generation. BES couples forward search, which evolves candidates through expansion and four evolution operators, with backward search, which recursively decomposes problems into verifiable sub-goals to provide dense intermediate feedback. We provide theoretical justification showing that candidates generated by expansion-only search are confined to a narrow entropy shell while evolution operators can escape it, and that backward sub-goal decomposition exponentially reduces the number of candidates needed to find a solution.

Experiments on logical reasoning, multi-hop reasoning, and open problem solving demonstrate that BES consistently improves over strong baselines in both post-training and inference settings. We hope that BES provides useful insights for self-improving language models and agents.

References

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*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r. 2, 7

W. Banzhaf, J.R. Koza, C. Ryan, L. Spector, and C. Jacob. Genetic programming. *IEEE Intelligent Systems and their Applications*, 15(3):74–84, 2000. doi: 10.1109/5254.846288. 8

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38 (16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720. 8

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, 2024. URL https://arxiv.org/abs/2407.21787. 2

Ronald Aylmer Fisher. *The genetical theory of natural selection: a complete variorum edition*. Oxford University Press, 1999. 2

Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai, 2025. URL https://arxiv.org/abs/2411.04872. 1

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane

Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783. 6

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran

Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small LLMs can master math reasoning with self-evolved deep thinking. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=5zwF1GizFa. 1, 8

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z. 1, 2, 6

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/. 8

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi: 10.1109/TSSC.1968.300136. 8

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 04 1992. ISBN 9780262275552. doi: 10.7551/mitpress/1090.001.0001. URL https://doi.org/10.7551/mitpress/1090.001.0001. 8

Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM reinforcement learning with on-policy tree search. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12355–12369, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.604. URL https://aclanthology.org/2025.acl-long.604/. 8

Audrey Huang, Adam Block, Dylan J Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T. Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=WJaUkwci9o. 8

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z. Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, Ottavia Bertolli, Tom Zahavy, Amol Mandhane, Jessica Yung, Iuliya Beloshapka, Borja Ibarz, Vivek Veeriah, Lei Yu, Oliver Nash, Paul Lezeau, Salvatore Mercuri, Calle Sönne, Bhavik Mehta, Alex Davies, Daniel Zheng, Fabian Pedregosa, Yin Li, Ingrid von Glehn, Mark Rowland, Samuel Albanie, Ameya Velingker, Simon Schmitt, Edward Lockhart, Edward Hughes, Henryk

Michalewski, Nicolas Sonnerat, Demis Hassabis, Pushmeet Kohli, and David Silver. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, Mar 2026. ISSN 1476-4687. doi: 10.1038/s41586-025-09833-y. URL https://doi.org/10.1038/s41586-025-09833-y. 1

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for LLM agent reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ZpQwAFhU13. 1, 2, 3, 6, 8

Marek Kadlčík and Michal Štefánik. Self-training language models for arithmetic reasoning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 12378–12386, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.721. URL https://aclanthology.org/2024.findings-emnlp.721/. 8

Levente Kocsis and Csaba Szepesvari. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, 2006. URL https://api.semanticscholar.org/CorpusID:15184765. 1, 3

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349. 1, 2, 7, 8

Eugene L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Oper. Res.*, 14:699–719, 1966. URL https://api.semanticscholar.org/CorpusID:36099120. 8

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158. 1

Shu Liu, Mert Cemri, Shubham Agarwal, Alexander Krentsel, Ashwin Naren, Qiuyang Mang, Zhifei Li, Akshat Gupta, Monishwaran Maheswaran, Audrey Cheng, Melissa Pan, Ethan Boneh, Kannan Ramchandran, Koushik Sen, Alexandros G. Dimakis, Matei Zaharia, and Ion Stoica. Skydiscover: A flexible framework for ai-driven scientific and algorithmic discovery, 2026. URL https://skydiscover-ai.github.io/blog.html. 7, 24

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB. 8

Kou Misaki, Yuichi Inoue, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? scaling LLM inference-time compute with adaptive branching tree search. In *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025. URL https://openreview.net/forum?id=3HF6yogDEm. 8

Hermann Joseph Muller. Some genetic aspects of sex. *The American Naturalist*, 66(703):118–138, 1932. 2

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131. 1, 7, 8

Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on arc-agi, 2026. URL https://arxiv.org/abs/2507.14172. 8

Zhenting Qi, Mingyuan MA, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller LLMs stronger problem-solver. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=6aHUmotXaw. 1

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. 8

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve. 2, 7

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6. 8

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11 (4):341–359, December 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328. URL https://doi.org/10.1023/A:1008202821328. 8

Fahim Tajwar, Guanning Zeng, Yueer Zhou, Yuda Song, Daman Arora, Yiding Jiang, Jeff Schneider, Ruslan Salakhutdinov, Haiwen Feng, and Andrea Zanette. Maximum likelihood reinforcement learning, 2026. URL https://arxiv.org/abs/2602.02710. 2, 6

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786. 6

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/. 6

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a. 8

Haorui Wang, Marta Skreta, Cher Tian Ser, Wenhao Gao, Lingkai Kong, Felix Strieth-Kalthoff, Chenru Duan, Yuchen Zhuang, Yue Yu, Yanqiao Zhu, Yuanqi Du, Alan Aspuru-Guzik, Kirill Neklyudov, and Chao Zhang. Efficient evolutionary search over chemical space with large language models. In *The Thirteenth International Conference on Learning Representations*,

2025a. URL https://openreview.net/forum?id=awWiNvQwf3. 8

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Thetaevolve: Test-time learning on open problems, 2025b. URL https://arxiv.org/abs/2511.23473. 8

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J. 1

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VNckp7JEHn. 1

Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. Large language interpolators can learn logical reasoning: A study on knights and knaves puzzles. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024a. URL https://openreview.net/forum?id=mxX8WdPCx9. 6

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*, 2024b. URL https://openreview.net/forum?id=s004OmYP2P. 8

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h. 1, 3, 8

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476. 1

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=0NphYCmgua. 1, 8

Yang Yue, 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? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=4OsgYD7em5. 1

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175. 1

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_3ELRdg2sgI. 1, 8

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=46Zgqo4QIU. 1

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=8rcFOqEud5. 8

Appendix - Table of Contents

A Pseudo Code **16**

B Formal Definitions of Evolution Operators **16**

C Theoretical Motivations **17**

C.1 Theoretical Motivation for Evolution Operators 17

C.2 Theoretical Motivation for Bidirectional Search 21

D Detailed Experimental Setup **22**

D.1 Logical Reasoning 22

D.2 Multi-Hop Reasoning 23

D.3 Open Problem Solving 24

E Case Study **26**

F Prompts for Open Problem Solving Tasks **27**

F.1 Backward Search: Goal Tree Decomposition 27

F.2 Forward Evolution Operations 29

G Identified Programs for Open Problem Solving Tasks **32**

G.1 Circle Packing (Square) 32

G.2 Circle Packing (Rectangle) 32

G.3 Heilbronn (Convex) 32

H Potential Limitations and Broader Impacts **33**

A. Pseudo Code

We provide the full pseudocode for BES below. Algorithm 1 describes the main loop, which alternates between forward search steps (Algorithm 2) and backward scoring (Algorithm 3). Every K_{dec} steps, the backward goal tree is refined by decomposing an unsolved leaf into finer sub-goals (Algorithm 4), after which all pool scores are recomputed under the updated tree.

Algorithm 1 BES: Bidirectional Evolutionary Search

Require: problem x, policy π_θ, verifier V, budget B, decomposition interval K_{dec}
Initialize $\mathcal{G}$ with root goal g_0; prompt π_θ to decompose g_0 into initial sub-goals
$\mathcal{P} \leftarrow \{n_0\}$ with $n_0 = ()$ ▷ Empty root node
for $t = 0, 1, \ldots$ **while** $\text{calls}(t) < B$ **do**
  $n' \leftarrow \textsc{ForwardStep}(\mathcal{P}, \mathcal{C}_t, \mathcal{G}, \tau_t)$ ▷ Algorithm 2
  $s(n') \leftarrow \textsc{BackwardScore}(n', g_0, \mathcal{G})$ ▷ Algorithm 3
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{n'\}$
  if $(t+1) \bmod K_{\text{dec}} = 0$ **then**
    $\mathcal{G} \leftarrow \textsc{BackwardDecompose}(x, \mathcal{G}, \mathcal{P})$ ▷ Algorithm 4
    for $n \in \mathcal{P}$ **do** ▷ Recompute all scores under refined tree
      $s(n) \leftarrow \textsc{BackwardScore}(n, g_0, \mathcal{G})$
    end for
  end if
  if n' is terminal $\wedge\, V(x, n') = 1$ **then**
    return n'
  end if
end for
return $\arg\max_{n \in \mathcal{P}:\, \text{terminal}(n)} V(x, n)$

Algorithm 2 details a single forward search step. An operator is sampled from the fixed distribution over the five operators. Single-parent operators (expansion and deletion) select a parent via Boltzmann selection based on backward scores; two-parent operators (combination, translocation, crossover) select a pair that maximizes joint sub-goal coverage.

Algorithm 3 computes the backward score for a candidate node by recursively traversing the sub-goal tree. If a sub-goal is fully satisfied, the score is 1; otherwise, the score blends the local verifier output with the average score over its children.

Algorithm 4 refines the goal tree by selecting a random unsolved leaf sub-goal and prompting the policy to decompose it into finer children, each equipped with its own local verifier.

B. Formal Definitions of Evolution Operators

For two parents n_a, n_b, let $s = \max\{\ell : y_{a,1:\ell} = y_{b,1:\ell}\}$ denote their common-prefix length, and let $\sigma_a = (y_{a,s+1}, \ldots, y_{a,t_a})$ and $\sigma_b = (y_{b,s+1}, \ldots, y_{b,t_b})$ be their respective suffixes.

(i) Combination. $n' = y_{a,1:s} \oplus \sigma_a \oplus \sigma_b$.

(ii) Deletion. $n' = (y_1, \ldots, y_{\ell-1}, y_{\ell+1}, \ldots, y_t)$, where $\ell \sim \text{Uniform}\{2, \ldots, t-1\}$.

(iii) Translocation. $n' = y_{a,1:s} \oplus \sigma_a[1 : r-1] \oplus (\sigma_b)_q \oplus \sigma_a[r+1 : m_a]$, where $r \sim \text{Uniform}\{1, \ldots, m_a\}$ and $q \sim \text{Uniform}\{1, \ldots, m_b\}$.

(iv) Crossover. $n' = y_{a,1:s} \oplus \sigma_a[1 : i] \oplus \sigma_b[j+1 : m_b]$, where $i \in \{0, \ldots, m_a\}$ and $j \in \{0, \ldots, m_b - 1\}$ are sampled uniformly.

Algorithm 2 FORWARDSTEP: Forward Search Step

Require: pool $\mathcal{P}$, eligible set $\mathcal{C}_t$, goal tree $\mathcal{G}$, temperature τ_t
Sample operator $o \in \{\textit{expand}, \textit{combine}, \textit{delete}, \textit{translocate}, \textit{crossover}\}$ with fixed probabilities
if $o \in \{\textit{expand}, \textit{delete}\}$ **then** ▷ Single-parent operators
  Sample parent n from $\mathcal{C}_t$ via Boltzmann selection (Eq. 3)
  if $o = \textit{expand}$ **then**
    Sample $K \sim \mathrm{Uniform}\{1, \ldots, K_{\max}\}$
    $n' \leftarrow (y_1, \ldots, y_t, y_{t+1}, \ldots, y_{t+K})$ where $y_{t+k} \sim \pi_\theta(\cdot \mid x \oplus y_{1:t+k-1})$
  else
    Sample $\ell \sim \mathrm{Uniform}\{2, \ldots, t-1\}$; $n' \leftarrow (y_1, \ldots, y_{\ell-1}, y_{\ell+1}, \ldots, y_t)$
  end if
else ▷ Two-parent operators
  Sample pair (n_a, n_b) from $\mathcal{C}_t$ via pair Boltzmann selection (Eq. 4)
  Compute shared prefix length s and suffixes σ_a, σ_b
  if $o = \textit{combine}$ **then**
    $n' \leftarrow y_{a,1:s} \oplus \sigma_a \oplus \sigma_b$
  else if $o = \textit{translocate}$ **then**
    Sample r, q; $n' \leftarrow y_{a,1:s} \oplus \sigma_a[1:r-1] \oplus (\sigma_b)_q \oplus \sigma_a[r+1:m_a]$
  else if $o = \textit{crossover}$ **then**
    Sample i, j; $n' \leftarrow y_{a,1:s} \oplus \sigma_a[1:i] \oplus \sigma_b[j+1:m_b]$
  end if
end if
return n'

Algorithm 3 BACKWARDSCORE: Backward Search Scoring

Require: candidate node n, sub-goal g, goal tree $\mathcal{G}$
if $V_g(x, n) = 1$ **then**
  return 1
else if $\mathrm{ch}(g) \neq \emptyset$ **then**
  return $\alpha \cdot V_g(x, n) + (1-\alpha) \cdot \frac{1}{|\mathrm{ch}(g)|} \sum_{g' \in \mathrm{ch}(g)} \textsc{BackwardScore}(n, g', \mathcal{G})$
else
  return $V_g(x, n)$
end if

C. Theoretical Motivations

C.1. Theoretical Motivation for Evolution Operators

In this section, we give a theoretical justification for the evolution operators. Expansion-only search builds candidates one lineage at a time. Evolution, by contrast, recombines blocks from different trajectories, so the number of reachable candidates grows as a Cartesian product of the per-block libraries.

Fix a task x and a step horizon T. A trajectory is $Y = (y_1, \ldots, y_T)$, where each y_t is a step:

$$P(Y) := \pi_\theta(Y \mid x) = \prod_{t=1}^{T} P(y_t \mid y_{<t}). \tag{10}$$

To quantify the uncertainty at each step, we introduce the *surprisal* (pointwise information content) of the t-th step and its expected value, the *conditional entropy*:

$$Z_t := -\log P(Y_t \mid Y_{<t}), \qquad h_t(y_{<t}) := H_P(Y_t \mid Y_{<t} = y_{<t}). \tag{11}$$

Intuitively, Z_t measures how surprising the generated step is, while h_t measures how uncertain the policy is about the next step on average, given the history so far.

Algorithm 4 BACKWARDDECOMPOSE: Goal Tree Decomposition

Require: problem x, current goal tree $\mathcal{G}$, pool $\mathcal{P}$
1: $\mathcal{L} \leftarrow \{g \in \mathcal{G} : \mathrm{ch}(g) = \emptyset \wedge \max_{n \in \mathcal{P}} V_g(x, n) < 1\}$ ▷ Unsolved leaves
2: Sample g^* uniformly from $\mathcal{L}$
3: Prompt π_θ to decompose g^* into children $\{g'_1, \ldots, g'_c\}$ with local verifiers $\{V_{g'_1}, \ldots, V_{g'_c}\}$
4: $\mathrm{ch}(g^*) \leftarrow \{g'_1, \ldots, g'_c\}$; add to $\mathcal{G}$
5: **return** $\mathcal{G}$

Summing the per-step surprisals yields the total information content of the trajectory:

$$S_T := -\log P(Y) = \sum_{t=1}^{T} Z_t. \tag{12}$$

Taking its expectation recovers the trajectory-level entropy, which decomposes by the chain rule as

$$H_T := H_P(Y) = \mathbb{E}_P[S_T] = \sum_{t=1}^{T} \mathbb{E}_P\big[h_t(Y_{<t})\big]. \tag{13}$$

Finally, let $\mathcal{F}_t = \sigma(Y_1, \ldots, Y_t)$ denote the natural filtration, i.e. the information available after observing the first t steps. This filtration will serve as the basis for the martingale analysis that follows.

We assume that evolution operators and verification are computationally cheap relative to policy calls. This is typical in practice: evolution operators act by directly editing step sequences (Section 3.1), and verification is fast for most tasks of interest, e.g. test-case execution for coding. The computational bottleneck is therefore mainly the number of policy calls. Other assumptions have been stated in Section 4.1.

C.1.1. DISCUSSION OF ASSUMPTIONS

To begin with, we first briefly discuss why Assumptions 4.1–4.3 are mild and naturally satisfied in practice.

Assumption 4.1 (bounded per-step surprise). This requires that no single step carries unbounded information. It holds for any policy with a finite action space, which includes all LLMs with a fixed vocabulary and all agents with a discrete action set. For an LLM with vocabulary size $|\mathcal{V}|$, we have $L = \log |\mathcal{V}|$.

Assumption 4.2 (decaying step dependence). This requires that changing a single step has diminishing influence on the conditional entropy of distant future steps. In practice, this is satisfied when the policy's effective context dependence decays with distance, a property that holds for finite-context models and, empirically, for transformer-based LLMs whose attention weights concentrate on recent tokens for most layers. For a policy with finite memory (e.g., a Markov chain of order d), $\beta_\ell = 0$ for all $\ell > d$, trivially satisfying the assumption.

Assumption 4.3 (linear block total correlation). By the chain rule, the total correlation decomposes as $\mathrm{TC}_P = \sum_{j=1}^{k} I_P(U_j; U_{<j})$, where $I_P(U_j; U_{<j})$ is the mutual information between block j and all preceding blocks. This term is strictly positive whenever block j depends on its context, which is the generic case for coherent sequential reasoning: the content of later reasoning steps is informed by earlier ones. As long as each block's dependence on the past contributes at least a constant amount of mutual information, the total correlation grows linearly in the number of blocks, and hence linearly in T when blocks have bounded size. Empirically, this is a weak requirement: even simple autoregressive models on natural language exhibit strong inter-block dependence, and reasoning traces exhibit even stronger dependence since later steps logically build on earlier conclusions.

C.1.2. SHELL CONFINEMENT OF EXPANSION

The deviation $S_T - H_T$ between a trajectory's actual information content and the expected entropy governs how tightly ordinary rollouts cluster around the typical set. We aim to show that this deviation is small with high probability. To this end, we decompose it into two terms and bound each separately.

Lemma C.1 (Martingale decomposition)**.** *Define*

$$D_t := Z_t - h_t(Y_{<t}), \qquad M_T := \sum_{t=1}^{T} D_t, \qquad R_T := \sum_{t=1}^{T} \big(h_t(Y_{<t}) - \mathbb{E}h_t(Y_{<t})\big). \tag{14}$$

Then $(D_t, \mathcal{F}_t)$ is a martingale difference sequence, $|D_t| \le L$, and

$$S_T - H_T = M_T + R_T. \tag{15}$$

Here M_T captures the per-step noise, while R_T captures how the realized trajectory shifts the conditional entropy away from its unconditional mean.

Proof. By definition, $\mathbb{E}[Z_t \mid \mathcal{F}_{t-1}] = H_P(Y_t \mid Y_{<t}) = h_t(Y_{<t})$, so $\mathbb{E}[D_t \mid \mathcal{F}_{t-1}] = 0$. Assumption 4.1 gives $0 \le Z_t \le L$. Since $h_t(Y_{<t})$ is the conditional expectation of Z_t, we also have $0 \le h_t(Y_{<t}) \le L$, hence $|D_t| \le L$. Finally, $S_T = \sum_t D_t + \sum_t h_t(Y_{<t})$ and $H_T = \sum_t \mathbb{E}h_t(Y_{<t})$, which proves Eq. (15). □

Lemma C.2 (Concentration of R_T)**.** *Under Assumption 4.2, for every $u > 0$,*

$$\Pr(|R_T| \ge u) \le 2\exp\left(-\frac{u^2}{2TC_\beta^2}\right). \tag{16}$$

Proof. Let $G(Y) = \sum_{t=1}^{T} h_t(Y_{<t})$, so $R_T = G(Y) - \mathbb{E}G(Y)$. We construct a Doob martingale by progressively revealing the trajectory one step at a time. Define

$$A_s = \mathbb{E}[G(Y) \mid Y_1, \dots, Y_s] = \sum_{t=1}^{s} h_t(Y_{<t}) + \sum_{t=s+1}^{T} \mathbb{E}[h_t(Y_{<t}) \mid Y_1, \dots, Y_s], \tag{17}$$

where the first sum is already determined by the observed steps, and the second sum averages over the unobserved future. Note that $A_0 = \mathbb{E}[G(Y)]$ and $A_T = G(Y)$, so $R_T = A_T - A_0 = \sum_{s=1}^{T} \delta_s$ where $\delta_s = A_s - A_{s-1}$.

Expanding the increment gives

$$\delta_s = \sum_{t=s+1}^{T} \Big(\mathbb{E}[h_t(Y_{<t}) \mid Y_1, \dots, Y_s] - \mathbb{E}[h_t(Y_{<t}) \mid Y_1, \dots, Y_{s-1}]\Big). \tag{18}$$

Each term measures how much the prediction of h_t changes upon revealing Y_s. By Assumption 4.2, this change is bounded by β_{t-s} for each $t > s$. Therefore

$$|\delta_s| \le \sum_{t=s+1}^{T} \beta_{t-s} \le \sum_{\ell=1}^{\infty} \beta_\ell = C_\beta. \tag{19}$$

Since (A_s) is a martingale, $\mathbb{E}[\delta_s \mid Y_1, \dots, Y_{s-1}] = 0$. Applying the Azuma-Hoeffding inequality to $R_T = \sum_{s=1}^{T} \delta_s$ with bounded increments $|\delta_s| \le C_\beta$ gives Eq. (16). □

Lemma C.3 (Concentration of M_T)**.** *The first term M_T is itself a martingale with bounded increments $|D_t| \le L$. Let*

$$V_T := \sum_{t=1}^{T} \mathbb{E}[D_t^2 \mid \mathcal{F}_{t-1}]. \tag{20}$$

If $V_T \le v_T$ almost surely for a deterministic v_T, then for every $u > 0$,

$$\Pr(|M_T| \ge u) \le 2\exp\left(-\frac{u^2}{2(v_T + Lu/3)}\right). \tag{21}$$

Proof. This is Freedman's inequality for martingales with increments bounded by L, applied to M_T and $-M_T$. If no sharper variance proxy is available, the worst-case bound $v_T = TL^2$ is valid. □

Combining the two concentration results yields a finite-sample analogue of the asymptotic equipartition property (AEP): with high probability, the information content of a trajectory is close to the expected entropy.

Theorem C.4 (Finite-sample AEP for policy rollouts)**.** *Under Assumptions 4.1–4.2 and the variance condition in Lemma C.3,*

$$\Pr(|S_T - H_T| \geq u) \leq 2\exp\left(-\frac{u^2}{8(v_T + Lu/6)}\right) + 2\exp\left(-\frac{u^2}{8TC_\beta^2}\right). \tag{22}$$

In particular, if $v_T = O(T)$, then for every fixed $\epsilon > 0$,

$$\Pr\left(\left|\frac{S_T}{T} - \frac{H_T}{T}\right| \geq \epsilon\right) \leq \exp(-\Omega(T)). \tag{23}$$

Proof. By Lemma C.1, $S_T - H_T = M_T + R_T$. A union bound gives $\Pr(|S_T - H_T| \geq u) \leq \Pr(|M_T| \geq u/2) + \Pr(|R_T| \geq u/2)$. Applying Lemma C.3 and Lemma C.2 with threshold $u/2$ proves Eq. (22). Taking $u = \epsilon T$ and $v_T = O(T)$ gives the exponential concentration rate. □

Corollary C.5 (Shell confinement)**.** *For $\epsilon > 0$, define the typical set*

$$A_\epsilon^{(T)} := \{y : |-\log P(y) - H_T| \leq \epsilon T\}. \tag{24}$$

Then $P(A_\epsilon^{(T)}) \geq 1 - \exp(-\Omega(T))$ under Theorem C.4. Moreover,

$$|A_\epsilon^{(T)}| \leq \exp(H_T + \epsilon T). \tag{25}$$

In other words, almost all rollouts land in a set of size $\approx \exp(H_T)$. Expansion-only search, regardless of how it selects prefixes, can only explore within this shell.

Proof. The probability statement follows from Theorem C.4. Every $y \in A_\epsilon^{(T)}$ satisfies $P(y) \geq \exp(-(H_T + \epsilon T))$, so summing probabilities gives $1 \geq |A_\epsilon^{(T)}| \exp(-(H_T + \epsilon T))$, which proves Eq. (25). □

C.1.3. Shell Escape via Evolution

We now show that evolution operators construct candidates whose expected log-probability falls strictly outside the entropy shell. We prove the results using crossover; the analysis for other evolution operators (combination, translocation) follows analogously.

Lemma C.6 (Splice-point KL identity)**.** *Split a trajectory at position s and write $V = Y_{1:s}$ and $U = Y_{s+1:T}$. Let (V, U) and (V', U') be two independent policy rollouts, and form the crossover trajectory $\widetilde{Y} = (V, U')$. The expected increase in native surprise is*

$$\mathbb{E}[-\log P(V, U')] - H_T = \mathbb{E}_{V,V'}\left[D_{\mathrm{KL}}(P_{U|V'} \| P_{U|V})\right]. \tag{26}$$

Equivalently,

$$\mathbb{E}[-\log P(V, U')] - H_T = I_P(V; U) + D_{\mathrm{KL}}(P_V \otimes P_U \| P_{V,U}) \geq I_P(V; U). \tag{27}$$

The gap is strictly positive whenever the suffix distribution $P_{U|V}$ depends on the prefix.

Proof. Using $P(V, U) = P_V(V) P_{U|V}(U)$,

$$\mathbb{E}[-\log P(V, U')] = H(V) + \mathbb{E}_{V,V'}\mathbb{E}_{U \sim P_{U|V'}}[-\log P_{U|V}(U)], \tag{28}$$

$$H_T = H(V) + \mathbb{E}_{V'} H(P_{U|V'}). \tag{29}$$

Subtracting gives Eq. (26). The marginal law of (V, U') is $P_V \otimes P_U$, so the same difference equals the cross-entropy gap $H(P_V \otimes P_U, P_{V,U}) - H(P_{V,U})$, which is $H(P_V \otimes P_U) + D_{\mathrm{KL}}(P_V \otimes P_U \| P_{V,U}) - H(P_{V,U})$. Since $H(P_V \otimes P_U) - H(P_{V,U}) = I_P(V; U)$, Eq. (27) follows. □

Lemma C.6 analyzes crossover at a single splice point. We now generalize to k-way block evolution, where a trajectory is partitioned into multiple blocks and each block is drawn from a different seed trajectory.

Fix a partition $0 = s_0 < s_1 < \cdots < s_k = T$ and define blocks $U_j = Y_{s_{j-1}+1:s_j}$. Let P denote the joint law of $(U_1, \ldots, U_k)$ under the policy, and let P_j be the marginal law of U_j. A k-way blockwise evolution draws k independent seed trajectories and takes block j from seed j. The resulting distribution is

$$Q = \bigotimes_{j=1}^{k} P_j. \tag{30}$$

Lemma C.7 (Block evolution cross-entropy gap)**.** *For $\widetilde{Y} \sim Q$,*

$$\mathbb{E}_Q[-\log P(\widetilde{Y})] - H(P) = \mathrm{TC}_P(U_1, \ldots, U_k) + D_{\mathrm{KL}}(Q\|P) \geq \mathrm{TC}_P(U_1, \ldots, U_k), \tag{31}$$

where $\mathrm{TC}_P(U_1, \ldots, U_k) := \sum_{j=1}^{k} H_P(U_j) - H_P(U_1, \ldots, U_k)$ is the block total correlation.

Proof. The expected surprise of an evolution sample under the policy is the cross entropy $H(Q, P) = H(Q) + D_{\mathrm{KL}}(Q\|P)$. Since Q is the product of the policy's block marginals, $H(Q) = \sum_j H_P(U_j)$. Subtracting $H(P) = H_P(U_1, \ldots, U_k)$ gives Eq. (31). □

We can now state the main theorem.

Theorem 4.4 (Shell confinement and escape)**.** *Restated from Section 4.1.*

Under Assumptions 4.1–4.3, define the typical set $A_\epsilon^{(T)} := \{y : |-\log P(y) - H_T| \leq \epsilon T\}$.

(a) Shell confinement. Every trajectory $Y \sim P$ produced by expansion satisfies $\Pr[Y \notin A_\epsilon^{(T)}] \leq \exp(-\Omega(T))$. That is, expansion-only search is confined to a typical set of size at most $\exp(H_T + \epsilon T)$.

(b) Shell escape. Let $Q = \bigotimes_{j=1}^{k} P_j$ be the k-way evolution distribution. For any $\epsilon < \gamma$,

$$\mathbb{E}_Q[-\log P(\widetilde{Y})] \;\geq\; H_T + \gamma T \;>\; H_T + \epsilon T,$$

so evolution candidates have expected log-probability strictly beyond the shell boundary. Moreover,

$$\Pr_Q\Big[\widetilde{Y} \in A_\epsilon^{(T)}\Big] \;\leq\; 1 - \frac{(\gamma - \epsilon)T}{LT - H_T - \epsilon T} \;<\; 1,$$

confirming that a positive fraction of evolution candidates escape the shell.

Proof. Part (a) is Corollary C.5.

For part (b), Lemma C.7 gives $\mathbb{E}_Q[-\log P(\widetilde{Y})] = H_T + \mathrm{TC}_P + D_{\mathrm{KL}}(Q\|P) \geq H_T + \gamma T$ by Assumption 4.3, proving Eq. (8).

For Eq. (9), let $p = \Pr_Q[\widetilde{Y} \in A_\epsilon^{(T)}]$. By Assumption 4.1, $-\log P(\widetilde{Y}) \in [0, LT]$ always. If $\widetilde{Y} \in A_\epsilon^{(T)}$ then $-\log P(\widetilde{Y}) \leq H_T + \epsilon T$. Therefore

$$H_T + \gamma T \;\leq\; \mathbb{E}_Q[-\log P(\widetilde{Y})] \;\leq\; (H_T + \epsilon T)\, p + LT\,(1 - p). \tag{32}$$

Rearranging: $p \leq \frac{LT - H_T - \gamma T}{LT - H_T - \epsilon T} = 1 - \frac{(\gamma - \epsilon)T}{LT - H_T - \epsilon T}$, which is strictly less than 1 whenever $\gamma > \epsilon$. □

C.2. Theoretical Motivation for Bidirectional Search

Theorem 4.5 (Exponential advantage from backward sub-goal signals)**.** *Restated from Section 4.2.*

Let N candidates be sampled independently. Terminal-only search requires $N_{\mathrm{term}} = \Omega(1/\prod_{i=1}^{m} p_i)$ candidates to obtain constant success probability. By contrast, backward-guided bidirectional search requires only $N_{\mathrm{bidir}} = O\big(p_{\min}^{-1} \log(m/\delta)\big)$, where $p_{\min} = \min_i p_i$, to collect evidence for all sub-goals with probability at least $1 - \delta$. In the symmetric case $p_i = p$, the ratio is $N_{\mathrm{term}}/N_{\mathrm{bidir}} = \Omega\big(p^{-(m-1)}/\log(m/\delta)\big)$, which is exponential in the number of sub-goals m.

Proof. For a single candidate, terminal success requires all sub-goals:

$$\Pr[V(x,n)=1] \le \Pr\left[\bigcap_{i=1}^{m}\{C_i(n)=1\}\right] = \prod_{i=1}^{m} p_i.$$

Therefore, after N independent candidates, the probability that a terminal-only method observes a complete solution is at most

$$1-\left(1-\prod_{i=1}^{m} p_i\right)^N.$$

To make this probability constant, one needs

$$N=\Omega\left(\frac{1}{\prod_{i=1}^{m} p_i}\right).$$

Now consider backward verification. The pool contains evidence for all sub-goals if, for every i, at least one sampled candidate satisfies $C_i(n)=1$. This event has probability

$$\prod_{i=1}^{m}\left(1-(1-p_i)^N\right).$$

Using $(1-p_i)^N \le e^{-Np_i}$, we obtain

$$\Pr[\text{some sub-goal is missing}] \le \sum_{i=1}^{m} e^{-Np_i} \le m e^{-Np_{\min}}.$$

Thus

$$N \ge \frac{1}{p_{\min}} \log \frac{m}{\delta}$$

ensures that the pool contains evidence for every sub-goal with probability at least $1-\delta$. Once such evidence exists, backward search identifies it and evolution can recombine the corresponding partial trajectories. Hence bidirectional search replaces the one-shot probability $\prod_i p_i$ by the much larger local probabilities p_i, turning a multiplicative hitting problem into a sub-goal collection problem. In the symmetric case $p_i = p$, terminal-only search needs $\Omega(p^{-m})$ candidates, whereas bidirectional search needs $O(p^{-1}\log(m/\delta))$, giving the stated exponential advantage. □

D. Detailed Experimental Setup

D.1. Logical Reasoning

Resources. The trainer uses 2 H200 GPUs. A separate auxiliary GPU hosts a vLLM server for the backward decomposer (a copy of Gemma-3-1B-it serving the DECOMPOSE calls of the goal tree); the trainer reaches it over HTTP.

Data. We generate the K&K corpus with the official sampling pipeline. The SFT cold-start corpus contains 1,000 problems with $n_{\text{people}} \in \{2,3,4\}$. The post-training corpus contains 5,000 problems with $n_{\text{people}} \in \{4,5,6\}$. Validation contains 143 problems per difficulty level $n_{\text{people}} \in \{2,\ldots,10\}$ for a total of 1,287 problems.

Training schedule. Stage 1 is a 3-epoch supervised fine-tuning on the SFT corpus to teach the output format. Stage 2 is post-training. For BES, each training step runs the forward-backward search on every problem in the training batch, returning either eight unique terminal trajectories (when the search hits the budget or finds enough successes) or fewer, in which case the remaining slots are padded with single-rollout samples to keep the GRPO group size fixed at 8; the trainer then conducts post-training on these samples. We compare against two baselines (GRPO and MaxRL), both of which sample 8 i.i.d. trajectories per problem.

Forward search. The forward search maintains a pool of partial reasoning trajectories partitioned at paragraph granularity (`\n\n`-separated reasoning steps). At every search step we sample one of five actions: *combine* with probability 0.10; *deletion* with probability 0.05; *translocation* with probability 0.075; *crossover* with probability 0.075; and *expansion* with probability 0.70. The Boltzmann temperature τ is annealed linearly from $\tau_0 = 2.0$ at step 0 to $\tau_{\text{end}} = 1.0$ at step $B-2$. A trajectory becomes *terminal* when it contains a "`### Final Answer`" marker followed by a parseable JSON map `{name: 0/1}`; the rule-based KK scorer then assigns reward $r \in \{0, 1\}$ from exact match against the ground-truth assignment. The search runs until $B = 200$ policy calls have been used or eight unique terminal trajectories have been found.

Backward search. At the start of every `BES` search on a puzzle we instantiate a goal tree whose root goal is to identify every person's role correctly, verified by the rule-based K&K scorer applied to the trajectory's final answer. The root expands into one sub-goal per person (determining whether that person is a knight or a knave) and each per-person sub-goal further expands into elementary verification strategies that human solvers commonly apply to K&K puzzles, e.g. assume the opposite role and seek a contradiction, assume the correct role and confirm consistency, or jointly fix two people's roles and check pair-wise consistency. Leaf sub-goals are verified via lightweight syntactic checks on the trajectory against the reasoning markers each strategy is expected to produce.

Because Gemma-3-1B-it is too small to reliably perform open-ended goal decomposition, we restrict the backward LLM's role to scheduling traversal of this template tree rather than constructing it. At the start of the search the decomposer is asked to choose the order in which the per-person sub-goals should be verified; every $D = 10$ search steps thereafter it is asked to pick which verification strategies should be activated under the next-in-line per-person sub-goal. The recursive node score $s(n)$ is recomputed on every search step using $\alpha = 0.3$.

Hyperparameters. Hyperparameters are listed in Table 5.

Table 5. Hyperparameters for the logical reasoning experiment (Knights-and-Knaves).

Category	**Hyperparameter = Value**
Model	backbone = Gemma-3-1B-it; cold-start = 3 epochs SFT on 1 K problems; post-training = 4 epochs on 5 K problems.

Optimization	optimizer = AdamW; learning_rate = 1×10^{-6};
	train_batch_size = 32; ppo_mini_batch_size = 32;
	ppo_micro_batch_size_per_gpu = 16; ppo_epochs = 1;
	clip_ratio = 0.2; grad_clip = 0.3; kl_coef = 0.0.

Generation	max_prompt_length = 1024; max_response_length = 4096;
	max_model_len = 6144; train_temperature = 1.0;
	val_temperature = 0.6; val_top_p = 0.95.

BES search	adv_estimator = `maxrl`; group_size = 8 trajectories/problem;
	search_budget = 200 policy calls/problem;
	decompose_interval = 10 search steps;

D.2. Multi-Hop Reasoning

Resources. The trainer uses 2 H200 GPUs. Two auxiliary servers run independently and communicate with the trainer over HTTP: (i) a retriever (E5 encoder + FAISS over the 2018 Wikipedia dump) on 1 H200 GPU, and (ii) a backward decomposition server hosting Llama-3.1-8B-Instruct on 1 H200 GPU.

Data. We use the answerable subset of MuSiQue. The training set is the 3-to-4-hop solvable split of the MuSiQue training data and is held fixed across methods; the validation set is the full official MuSiQue validation set.

Training schedule. Two epochs of post-training. At each rollout the `BES` search returns 8 trajectories per problem; the same per-problem budget is used by the GRPO and Tree-GRPO baselines for fair comparison. Two epochs are sufficient; extra epochs lead to overfitting and training collapse.

Action format and reward. The agent emits actions as `<think>...</think>` followed by either `<search>q</search>` or `<answer>...</answer>`. Each `<search>` call goes to the offline retriever which returns the top 3 passages back to the agent inside `<information>...</information>` tokens.

Forward search. The forward search maintains a candidate pool of trajectories per question, where every trajectory is a sequence of $(\langle\texttt{think}\rangle, \langle\texttt{search}\rangle, \langle\texttt{information}\rangle)$ triples optionally followed by a terminal $(\langle\texttt{think}\rangle, \langle\texttt{answer}\rangle)$ pair. Strict format validation is enforced on every expansion.

At every search step we sample one of five actions with the same mixture used in the logical-reasoning experiment: *combination* 0.10, *deletion* 0.05, *translocation* 0.075, *crossover* 0.075, *expansion* 0.70. The evolution operators here operate at the triple boundary. The Boltzmann temperature τ is annealed linearly from $\tau_0 = 1.5$ at step 0 to $\tau_{\text{end}} = 0.3$ at the budget. We pick $\alpha = 0.7$.

Backward search. Backward search decomposes each question into an ordered chain of atomic sub-questions. The original question together with these sub-questions defines the per-question goal tree. The local verifier V_{g_i} is queried by the forward search for every candidate trajectory at every step, so a per-call LLM verifier would be impractical. We therefore instantiate V_{g_i} as a fast embedding model: every $\langle\texttt{search}\rangle$ query that a trajectory n has emitted is embedded into the `all-MiniLM-L6-v2` sentence-embedding space, and the same is done for each sub-question. A sub-question g_i is declared *covered* by trajectory n when

$$\max_{q \in \text{searches}(n)} \cos\big(\text{emb}(q), \text{emb}(g_i)\big) \;\geq\; \sigma_{\text{cov}} = 0.6, \tag{33}$$

i.e. when at least one of the trajectory's search queries semantically aligns with the sub-question.

Since a later sub-question's answer typically depends on those of the earlier ones, we evaluate sub-goals sequentially: a sub-goal is only checked if all preceding sub-goals have been satisfied.

Hyperparameters. Hyperparameters are listed in Table 6.

Table 6. Hyperparameters for the multi-hop reasoning experiment (MuSiQue).

Category	**Hyperparameter = Value**
Model	backbones = {Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct}; post-training = 2 epochs on the 3–4-hop solvable MuSiQue split.

Optimization	optimizer = AdamW; learning_rate = 1×10^{-6};
	lr_warmup_ratio = 0.285;
	train_batch_size = 128; val_batch_size = 32;
	ppo_mini_batch_size = 16; ppo_micro_batch_size = 8;
	kl_loss_coef = 1×10^{-3}; kl_loss_type = low_var_kl.

Generation	max_prompt_length = 4096; max_response_length = 2048;
	max_obs_length = 500; temperature = 1.0;
	max_turns = 3.

BES search	adv_estimator = `grpo`; group_size = 8 trajectories/problem;
	search_budget = 50 policy calls/problem; K-parallel = 4;
	embedder = all-MiniLM-L6-v2; sim_threshold_σ_{cov} = 0.6;

Retriever	encoder = intfloat/e5-base-v2; index = wiki-18 FAISS; topk = 3.

D.3. Open Problem Solving

Resources. All compute is on a single CPU node; LLM access is through the OpenAI API. We use `gpt-5` with `reasoning_effort = high` for forward proposals, meta-reasoning, and backward decomposition. Each run is capped at $50 of API spend; baseline frameworks (OpenEvolve, GEPA, ShinkaEvolve) are run under the same setting and we directly use the results from Skydiscover (Liu et al., 2026).

Benchmarks. Three open optimization tasks: (i) *Circle Packing (Square)*: pack $n = 26$ non-overlapping circles in the unit square to maximize the sum of radii; (ii) *Circle Packing (Rect)*: the same in a fixed-aspect rectangle; (iii) *Heilbronn (Convex, $n = 13$)*: place 13 points in the unit square to maximize the minimum area of any convex polygon formed by a subset of the points. We report mean and best objective value across 3 runs per benchmark.

Algorithm. We adopt ShinkaEvolve as the base program-evolution framework. A population of executable Python programs is maintained in an islanded SQLite archive; at each generation, parents are sampled from the archive, mutated by an LLM-driven proposer, and the resulting offspring are evaluated against the benchmark scorer. BES adds two components on top: (a) the four evolution operators (combination, deletion, translocation, crossover), realized as LLM-driven *joint rewrites* of two parent programs (since direct concatenation is not meaningful for executable programs), and (b) a backward goal tree that supplies dense intermediate scores.

Forward search. The four evolution operators of the main paper — combination, translocation, crossover, deletion — are realized at the program level. Direct token-level concatenation is not meaningful for executable programs, so we pass both parents to the proposer with an operator-specific instruction (e.g. "combine the structural decisions of program A with the parameter choices of program B") and ask it to return a single new program implementing the requested edit. Detailed prompts are listed in Appendix F.2.

Backward search. In the backward goal tree, each leaf carries a Python verifier expression that returns a continuous partial-progress score in $[0, 1]$, evaluated against a benchmark-supplied namespace describing the program's outputs. Detailed prompts are listed in Appendix F.1.

Tree growth is performed adaptively during the run. A decomposition pass is triggered whenever the population has failed to improve on the raw objective for $S = 5$ consecutive generations (improvement margin $\Delta = 10^{-2}$): we pick a leaf that no program in the archive has yet fully satisfied, and ask the decomposer LLM to propose 2–4 child sub-goals that together cover the leaf, each with its own natural-language description and Python verifier. This adaptive schedule lets the tree grow into the kinds of bottlenecks the population actually encounters.

When calculating the score, in order not to hurt ground truth signals, we rank programs by a *bucket-interpolation* effective score: programs are first bucketed by the raw objective at precision 10^{-2} and ranked by bucket; within a bucket, the recursive backward score acts as an intra-bucket sub-rank, scaled so that it can never push a program past the next bucket boundary. This guarantees that any improvement in the raw objective dominates any change in the backward signal, so the backward score mainly acts as a tie-breaker among programs of similar raw scores.

Hyperparameters. Hyperparameters are listed in Table 7.

Table 7. Hyperparameters for the open problem solving experiment.

Category	**Hyperparameter = Value**
LLM	backbone = gpt-5; reasoning_effort = high.

Evolution	num_generations = 100; max_evaluation_jobs = 2;
	max_proposal_jobs = 2; max_db_workers = 2;
	max_patch_resamples = 3; max_patch_attempts = 2.

Operators	patch_types = {diff, full, cross} with probs $\{0.6, 0.3, 0.1\}$; BES operators = {combine, deletion, translocation, crossover}.

Database	num_islands = 1; archive_size = 40;
	parent_selection_strategy = weighted; parent_selection_lambda = 10;
	effective_score = bucket-interpolation (precision 10^{-2}).

BES backward	trigger = stagnation; $S = 5$ generations; margin $\Delta = 10^{-2}$; children_per_decompose = 2–4; max_tree_depth = 2; recursive_blend_α = 0.3; monotonic_expansion = True.

E. Case Study

We illustrate the BES search process on a multi-hop question answering example from the MuSiQue dataset. The question is: *"What is the record label of the artist who originally recorded Back to Bedlam?"* The correct answer is Custard Records. Figure 5 shows the full search trace.

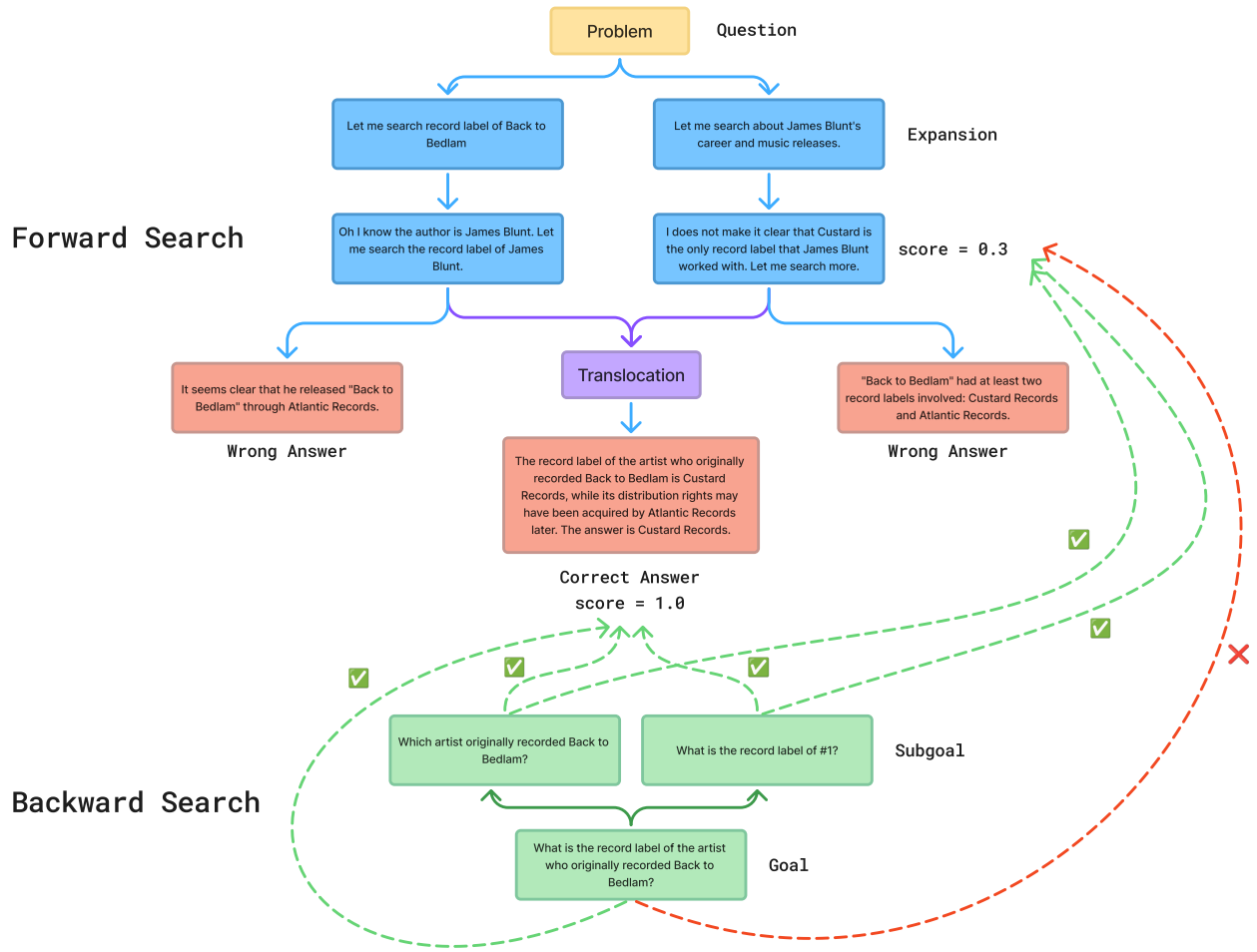

Figure 5. Case study of BES on a multi-hop reasoning problem. The forward search (top) explores two branches via expansion, both of which lead to wrong answers. A translocation operator then combines a reasoning step from the right branch into the left branch, producing a correct answer. The backward search (bottom) decomposes the original question into two sub-goals and provides dense verification feedback (green/red arrows) to guide parent selection.

Backward search. The backward search decomposes the original question into two sub-goals: (1) *"Which artist originally recorded Back to Bedlam?"* and (2) *"What is the record label of #1?"*. Each sub-goal is equipped with a local verifier based on embedding similarity (Section D.2). This decomposition allows the search to track partial progress: a candidate that correctly identifies the artist (James Blunt) but retrieves the wrong label still receives a non-zero score, providing a useful signal for parent selection.

Forward search: expansion. The search begins with two expansion branches from the root. The left branch searches for the record label of "Back to Bedlam" directly and concludes that the album was released through Atlantic Records, which is incorrect. The right branch takes a broader approach, searching for James Blunt's career and music releases, and discovers that both Custard Records and Atlantic Records were involved. However, it also fails to produce the correct final answer, arriving at a wrong conclusion.

Forward search: translocation. At this point, the backward search scores both branches. The right branch receives a score of 0.3, indicating partial progress: it has identified relevant information about Custard Records but has not resolved the question correctly. The key moment occurs when the translocation operator replaces the reasoning step in the left branch (which concluded Atlantic Records) with the more nuanced reasoning step from the right branch (which noted both Custard Records and Atlantic Records). The resulting candidate inherits the left branch's identification of James Blunt as the artist and the right branch's awareness of the Custard Records connection. This recombined trajectory correctly concludes that the original record label is Custard Records, achieving a score of 1.0.

This example highlights two core advantages of BES. First, the backward search provides dense intermediate feedback: even though both initial branches produce wrong final answers, the sub-goal scores distinguish the branch that has made more relevant progress (score 0.3), enabling informed parent selection. Second, the translocation operator constructs a correct trajectory by transplanting a single useful reasoning step from one branch into another. Neither branch alone would have reached the correct answer through further expansion, but their combination via translocation succeeds. This illustrates how evolution operators can discover solutions beyond what any single policy rollout can produce.

F. Prompts for Open Problem Solving Tasks

This appendix lists the prompts used by BES on the open-problem benchmarks. The backward-search decomposition prompt (one per benchmark) elicits a JSON array of verifiable sub-goals from a goal-tree leaf, while the four evolution-operation prompts (DIFF, DIFF_ABLATE, FULL, CROSS) drive forward-search mutations. All prompts use Python `str.format`-style placeholders such as `{code_content}`, `{performance_metrics}`, `{previous_attempts}`.

F.1. Backward Search: Goal Tree Decomposition

The decomposition prompt is benchmark-specific. We list here the prompts used for Circle Packing (Square); the prompts for Circle Packing (Rectangle) and Heilbronn (Convex) follow the same structure.

Goal Tree Decomposition Prompt

```
Decompose a circle-packing goal into smaller verifiable subgoals.

## Problem
Pack n=26 non-overlapping circles in [0,1]^2 to achieve sum_of_radii > 2.636 (
    STRICTLY EXCEED best-known).
Every candidate already satisfies validity (in-square, non-overlap, r>=0). Do NOT
    use those as subgoals.

## Elite reference layouts (top {n_elites} from current archive --- these define the
     search frontier)
Look ACROSS all of them to identify:
(a) structural properties EVERY elite has -> strong reference-kind subgoals
(b) properties where elites still VARY -> indicates room for improvement, target
    with aspirational subgoals
(c) properties NO elite has yet but a hypothetical sum_r > best solution would have
    -> aspirational

{elite_blocks}

## Parent goal to decompose
{goal_desc}

## Two kinds of subgoals (BOTH required)
Produce 3-4 subgoals, MIXING the two kinds below:

A. kind="reference" (1-2 subgoals): structural properties that EVERY elite above
    already
   has and that naive layouts demonstrably LACK (naive grid ~ 2.4 or concentric ring
     ~ 2.2).
```

```
   MUST be True on every elite AND False on a uniform-radii ring/grid. Generic
    properties
   that any symmetric layout has (e.g. "geometric center near (0.5,0.5)", "mirror
    symmetry")
   are BAD --- naive rings satisfy them too. Good shapes: "max radius >= some_value",
     "at
   least K circles with r >= some_value", "at least one pair of tangent circles with
   combined radius > some_value". Pick the threshold so EVERY elite passes.

B. kind="aspirational" (2-3 subgoals): concrete structural ideas for HOW to push
    sum_r
   further. These typically evaluate FALSE on most or all elites --- that is the
    point: they
   describe what an exceed-target solution would have but the current elites still
    lack.
   Each must ALSO be False on naive layouts (uniform grid ~ 2.4 or ring of 26 equal
    circles
   ~ 2.2); otherwise a naive layout trivially passes despite a poor sum_r. Good
    shapes:
   - the largest circle is strictly larger than the max max_radius across elites
     (naive baselines: grid r~0.083, ring r~0.12);
   - more circles above some "large" radius cutoff than any elite achieves;
   - tighter local packing: many pairs with center-distance < r_i + r_j + epsilon (
    tangent clusters);
   - a structural motif (hexagonal core, nested ring, dense corner clusters)
    requiring large radii.
   AVOID pure symmetry/centroid predicates --- naive baselines satisfy them
    trivially.
   Set thresholds slightly beyond what the best elite exhibits --- a layout that
    matches the
   elites still FAILS, but a layout that exceeds them can pass.

## Rules (both kinds)
- A strict sub-property of the parent --- never a rephrasing of the parent itself.
- Pick subgoals from DIFFERENT categories: radius distribution, boundary usage,
    spatial
  coverage, symmetry, local geometry. Don't write variants of one idea.

## verify_code: return a DENSE score in [0,1], not just bool
verify_code should evaluate to a float in [0,1] representing partial credit --- NOT
    a bool.
Use the form min(1.0, <actual> / <target>) so that progress toward the goal earns
    credit.
A bool is accepted (True->1.0, False->0.0) but wastes gradient; prefer dense.

CRITICAL FORMAT REQUIREMENT --- single Python expression only:
  verify_code is evaluated via Python's eval(). It MUST be a single expression
  (no semicolons, no multi-line x=...; y=...; result chains, no def/for/if
  statements, no newline-separated assignments). If you need intermediate values,
  inline them or use a one-shot generator/comprehension.

## Output
Each subgoal:
- kind: "reference" or "aspirational".
- description: short sentence; for aspirational, briefly say WHY it pushes.
- verify_code: Python expression returning float in [0,1] (or bool). Uses centers
    ((26,2)),
  radii ((26,)), n=26, sum_r=float(radii.sum()), np. Must run without error.
- expected_result: typical value across the elite reference layouts (1.0 if every
    elite
  meets it; a fraction < 1.0 if elites only partially meet it).

Output ONLY a JSON array:
```

```
```json
[
  {"kind": "reference",    "description": "...", "verify_code": "...", "
    expected_result": "..."},
  {"kind": "aspirational", "description": "...", "verify_code": "...", "
    expected_result": "..."}
]
```
```

F.2. Forward Evolution Operations

F.2.1. COMBINATION

Combination Iteration Prompt

```
# Current program

Here is the current program we are trying to improve:

```{language}
{code_content}
```

Here are the performance metrics of the program:

{performance_metrics}{text_feedback_section}

Task: trick combination

Below you will see SEVERAL inspiration programs. For EACH inspiration:
1. Identify the single most distinctive trick / mechanism it uses (a particular
   initialization, a refinement step, a numerical formulation, a heuristic, ...).
2. Decide whether that trick is compatible with the current program and is likely
   additive (i.e. attacks a different failure mode than what the current program
   already handles).

Then produce a NEW full program that is the current program PLUS the compatible
tricks from the inspirations stitched in. Be explicit in the <DESCRIPTION> about
which trick came from which inspiration and why you expect them to compose
without redundancy. Drop tricks that conflict.

IMPORTANT: This is a combination, not a free rewrite. The skeleton should follow
the current program; the inspirations only contribute identifiable plug-in tricks.

Key directions to explore:
1. The optimal arrangement may involve heterogeneous or variable-sized elements
2. Strong solutions often use hybrid global-local patterns
3. The optimization routine is critical - use models with carefully tuned parameters
4. Use scipy optimize, LP, or SLSQP to optimize variables given candidate structures
```

F.2.2. DELETION

Deletion Iteration Prompt

```
# Current program

Here is the current program. The evolution loop has been stuck on iterations of
    approaches similar to this one --- incremental tweaks have not been moving the
    score:

```{language}
{code_content}
```

Performance metrics of the current program:

{performance_metrics}{text_feedback_section}

{previous_attempts}

Task

The current implementation has plateaued. Iterating on it further is unlikely to
    help. Instead:

1. Identify components of the current code that look unreasonable or that may be
    holding the search inside a local optimum (heuristics that don't pay off, design
     choices the search keeps committing to, dead branches, parameter sweeps that
    add little).
2. DELETE those components.
3. Rewrite the program from a fundamentally new perspective: pick an algorithm class,
     data structure, or strategy that the current program does NOT use, and commit
    fully to it.

Do not iterate on the current implementation. Do not stitch new code onto the old
    skeleton. Commit fully to a different approach.

A fundamental change replaces the solution representation (e.g., closed-form <->
    free coordinates <-> discrete) or the search paradigm (e.g., gradient <->
    sampling <-> enumeration). Swapping the optimizer, picking a sibling parametric
    family, or adding numerical guards are NOT fundamental changes --- they leave
    the search trapped.

For example, the following are structurally orthogonal algorithm classes --- two
    attempts in the same class are minor variants of each other no matter how the
    surface code differs:
- Closed-form analytical construction (orbit of a finite symmetry group, vertices of
     a known polytope, regular polygon, root-system points)
- Low-discrepancy / quasi-random sampling on a fixed domain (Halton, Sobol,
    Hammersley, sunflower spiral, Fibonacci lattice)
- Lattice / grid enumeration (G x G square grid, hexagonal lattice, crystallographic
     packing --- search over subsets/labels)
- Continuous local optimization on free decision variables (gradient on a smoothed
    objective, SLSQP / Nelder-Mead / coordinate ascent on the raw objective)
- Population-based global search (CMA-ES, Differential Evolution, Genetic Algorithm
    --- many parallel candidates with selection)
- Discrete combinatorial search over a finite candidate set (simulated annealing on
    subset selection, branch-and-bound, ILP, beam search over partial states)
- Constructive online insertion (farthest-first / k-center, max-min greedy adding
    one element at a time, beam search building a configuration step by step)
- Physics / relaxation methods (Lloyd / centroidal Voronoi tessellation, repulsive
    force fields, gradient flow with hard-margin barriers, simulated cooling on
    continuous coordinates)
```

```
- Algebraic / number-theoretic structure (lattice orbits of a Coxeter group, points
    related by a Mobius / projective map, modular-arithmetic constructions)

The list is illustrative, not exhaustive --- feel free to commit to any class
    outside the previous attempts, including ones not above.

In the <DESCRIPTION>: name the OLD strategy in one sentence, the NEW strategy you
    committed to in one sentence, what you removed, and why a clean swap (not
    incremental tweaks) is the right move now --- what local optimum the old
    strategy is stuck in and how the new one structurally avoids it.

Key directions to explore:
1. The optimal arrangement may involve heterogeneous or variable-sized elements
2. Strong solutions often use hybrid global-local patterns
3. The optimization routine is critical - use models with carefully tuned parameters
4. Use scipy optimize, LP, or SLSQP to optimize variables given candidate structures
```

F.2.3. CROSSOVER

Crossover Iteration Prompt

```
# Current program

Here is the current program we are trying to improve (you will need to propose a new
     program with the same inputs and outputs as the original program, but with
    improved internal implementation):

```{language}
{code_content}
```

Here are the performance metrics of the program:

{performance_metrics}{text_feedback_section}

Task

Perform a cross-over between the code script above and the one below. Aim to combine
     the best parts of both code implementations that improves the score.
Provide the complete new program code.

IMPORTANT: Make sure your rewritten program maintains the same inputs and outputs as
     the original program, but with improved internal implementation.

Key directions to explore:
1. The optimal arrangement may involve heterogeneous or variable-sized elements
2. Strong solutions often use hybrid global-local patterns
3. The optimization routine is critical - use models with carefully tuned parameters
4. Use scipy optimize, LP, or SLSQP to optimize variables given candidate structures
```

F.2.4. TRANSLOCATION

Translocation Iteration Prompt

```
# Current program (the "near" parent --- keep its skeleton)

```{language}
```
```

```
{code_content}
```

Performance metrics: {performance_metrics}{text_feedback_section}

Task: trick translocation from a distant relative

Below you will see ONE inspiration program drawn from the archive (a "distant relative" --- likely structurally different from the current program). Your job:

1. Read it and pick the ONE trick that is most likely to help the current program --- a specific initialization, refinement step, constraint formulation, numerical detail, or heuristic. Be concrete; name it.
2. Transplant ONLY that trick into the current program. Keep the rest of the current program intact. Do NOT also fold in other ideas from the donor and do NOT broadly rewrite the recipient.
3. Adapt naming / signatures so the transplant compiles, but do not refactor surrounding code beyond what the transplant strictly requires.

Argue in the <DESCRIPTION>: which trick, why this one, and why grafting it onto the current skeleton is more promising than full crossover.

Key directions to explore:
1. The optimal arrangement may involve heterogeneous or variable-sized elements
2. Strong solutions often use hybrid global-local patterns
3. The optimization routine is critical - use models with carefully tuned parameters
4. Use scipy optimize, LP, or SLSQP to optimize variables given candidate structures

G. Identified Programs for Open Problem Solving Tasks

This appendix summarizes, for each open-problem benchmark, the structure of the best program discovered by BES.

G.1. Circle Packing (Square)

The best program for the n=26 unit-square instance is a hybrid global optimiser. It maintains both circle centres and radii as decision variables and alternates among four ingredients: (i) a K-nearest-neighbour radii projection that produces a feasible packing in closed form; (ii) an active-set LP cutting-plane routine that solves for the maximum feasible radii under the current contact graph, with stateful slack counters and an annealed tightness τ; (iii) a two-tier simulated-annealing search over centre perturbations with adaptive LP lock-ins and an annealed move mix; and (iv) periodic SLSQP micro-bursts on a focused subset of circles for local contact tightening. Search is launched from a portfolio of deterministic seeds, including hexagonal rows, edge rings, corner-weighted arrangements, and spokes-plus-concentric-pentagon interiors, each with controlled anisotropic jitter and short feasibility probing.

G.2. Circle Packing (Rectangle)

The best program for the n=21 rectangle-of-perimeter-4 instance is a deterministic multi-start constructor with a two-stage refinement. It first enumerates a dense grid of aspect ratios and a small set of jitter scales, generating candidate layouts from several heterogeneous hex-like row patterns. Each candidate is feasibility-clamped, and only the top-K seeds (ranked by sum of radii, tie-broken by the minimal pairwise slack) advance to optimisation. Each surviving seed is then refined by a two-stage SLSQP: stage 1 fixes the aspect ratio with tighter bounds, and stage 2 releases the aspect ratio so all variables (centres, radii, and aspect) are co-optimised under a shrink-only feasibility projection.

G.3. Heilbronn (Convex)

The best program for the n=13 convex-hull instance commits to a C_3-symmetric parameterisation: one centre point and four concentric 3-orbits (each an equilateral triangle of 3 points at 120° spacing), giving $1{+}4{\times}3 = 13$ points and only 8 free parameters (four radii via a softplus-ordered map, four phases reduced modulo $2\pi/3$). Optimisation is a Coordinate Pattern

Search with three custom ingredients: (i) K-guided weighted co-participation, where the top-K smallest triangles vote on which ring to perturb next; (ii) a symmetric finite-difference $\delta v/\delta r$ sensitivity per ring with adaptive ϵ and a per-sweep cache; and (iii) per-ring adaptive step sizes with mild growth on acceptance and targeted shrink after repeated failures. The objective is the exact minimum over all $\binom{13}{3} = 286$ triangles, with hull area normalised by Andrew's monotone chain plus the shoelace formula. Search is launched from a deterministic multi-seed pool with a short pre-refinement sweep before CPS.

H. Potential Limitations and Broader Impacts

Potential Limitations. We acknowledge the following limitations of the paper:

(1) BES requires an objective reward signal to guide the search. It has not been tested on subjective evaluation tasks where such signals are difficult to obtain, such as academic writing.

(2) The backward search relies on the policy's ability to decompose problems into meaningful sub-goals. For very weak models, this decomposition capability is limited.

(3) Due to resource constraints, our post-training experiments use relatively small models up to 8B parameters.

Broader Impacts. Our work proposes a general-purpose search framework for improving the quality of LLM and agent outputs. On the positive side, BES can help models achieve stronger reasoning performance, potentially reducing the need for larger models and their associated computational and environmental costs. The backward search component also improves interpretability by explicitly decomposing problems into verifiable sub-goals, making the search process more transparent. However, more effective search methods could also enable stronger performance on tasks that could be misused.