Optimization-like Reasoning in LLMs over Expanding Search Spaces

Nicolas Astorga[*]
University of Cambridge

Nabeel Seedat
University of Cambridge

Mihaela van der Schaar
University of Cambridge

Abstract

Verifiable reward training has improved mathematical and coding reasoning, but these domains capture only part of step-by-step decision making. Many real-world tasks require finding a high-value feasible plan among many valid alternatives. We introduce OPT⋆, a scalable family of optimization-style tasks for training and evaluating LLM step-by-step optimization-like reasoning along a complexity axis: each task provides a feasibility checker and evaluator, while a complexity parameter expands the search space without requiring new human labels. This motivates studying these tasks in two regimes: (i) solver-guided online policy optimization, which uses a solver as a value oracle for partial states and uses rank-based advantages to reinforce better next steps, and (ii) search-based offline RL when such solvers are unavailable. Theoretically, we relate success in large search spaces to the information a reasoner extracts per unit of search budget. Empirically, we ablate the ingredients that make search efficient on OPT⋆ and show that training on OPT⋆ improves step-by-step optimization-like reasoning. Code: GitHub Repository.

1 Introduction

Large language models (LLMs) have shown remarkable success on diverse reasoning tasks [51, 64], but they remain brittle on constrained step-by-step decision-making tasks. In such problems, a locally plausible step may make the remaining problem infeasible or force a low-quality completion [24, 55]. This issue is especially visible in optimization-style tasks: there are often many feasible answers, but the goal is to construct a better answer under constraints, not just any feasible completion.

A common way to improve reasoning is *verifiable-reward training*: given a question and an automatically checkable answer, the model samples candidate traces and reinforces those that lead to correct answers [12, 20, 26, 45, 57–59]. This recipe is especially effective for math and code, where correctness can often be verified through exact answers and many established benchmarks exist. However, scaling the idea to develop other forms of reasoning faces two obstacles. First, it is difficult to generate many tasks that reliably require nontrivial reasoning. Second, even when such tasks are available, obtaining supervision that identifies which intermediate steps are useful is often expensive.

Optimization as a source of step-by-step supervision. We study constrained optimization tasks as a scalable source for developing optimization-like reasoning. Optimization problems naturally contain the ingredients that verifiable training needs: many instances can be generated automatically; partial actions can often be checked for feasibility; complete solutions can be scored by an objective; and difficulty can be increased by enlarging the decision space. They also differ from standard answer-correctness settings. In many optimization problems, there are many feasible solutions, but only some are high value. Thus, the model must learn not only to produce a valid solution, but also to choose intermediate steps that lead toward a better final outcome.

[*]Correspondence: nja46@cam.ac.uk

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

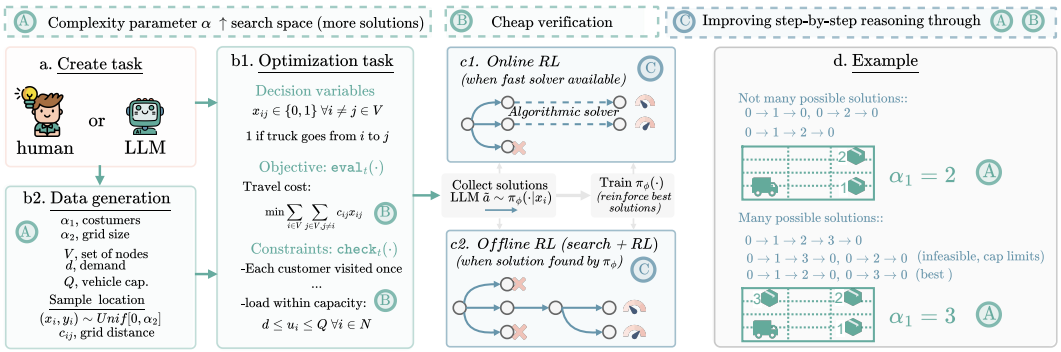


Figure 1: **OPT⋆ overview using a Traveling Salesman-style example. (a)** Tasks can be instantiated by humans, programs, or LLM-assisted generators. **(b1)** Classical optimization problems naturally expose objectives and constraints, making them auto-verifiable. **(b2)** Difficulty can be scaled by changing a complexity parameter α, such as the number of customers or the structure of the instance. Increasing α enlarges the search space while feasibility and objective evaluation remain cheap. **(c)** We study two training recipes: solver-guided next-step supervision when a solver is available, and search-based discovery when only feasibility checks and terminal objective values are available. **(d)** A model action can correspond to fixing one or more structured decision variables.

This perspective is useful both scientifically and practically. Scientifically, optimization tasks provide controlled environments for studying step-by-step decision making under constraints. Practically, many real workflows require optimization-like reasoning: assigning people to roles, scheduling jobs, routing deliveries, packing objects, selecting subsets under a budget, or coordinating multi-step plans. These tasks require models to balance hard constraints against soft preferences, local gains against future flexibility, and feasibility against objective value.

OPT⋆ tasks. We formalize this setting as OPT⋆: *OPTimization-based Scalable Tasks for Auto-verifiable Reasoning*. An OPT⋆ task is generated by a procedure $\text{Build}_\alpha(\cdot)$, where α controls task complexity. Each instance exposes two inexpensive verification routines: a feasibility checker $\texttt{chk}_t(\cdot)$ for partial actions and an outcome evaluator $\texttt{eval}_t(\cdot)$ for complete solutions. Increasing α enlarges the number of possible solutions or feasible trajectories, but does not require new human labels. This decouples reasoning difficulty from annotation effort.

Key properties of OPT⋆ tasks. (See Traveling Salesman Problem example in Figure 1)

1. Scalable generation. Task instances are sampled from a generator controlled by a complexity parameter α. Increasing α increases the size or structure of the search space (e.g., # of customers).

2. Cheap verification. A feasibility checker $\texttt{chk}_t(\cdot)$ (e.g., enforcing load limits) validates partial actions, and an outcome evaluator $\texttt{eval}_t(\cdot)$ (e.g., Euclidean distance) scores complete solutions. These routines are much cheaper than searching for an optimal solution.

3. Step-level training signal. When a solver is available, it can score candidate next steps by their best possible completion. When no fast solver is available, search can still discover high-value trajectories using $\texttt{chk}_t(\cdot)$ and $\texttt{eval}_t(\cdot)$.

Research objective: We study optimization-like reasoning over expanding search spaces as α grows, focusing on how performance is affected by search-space size and how to improve across different regimes. ► **Empirically:** we use the OPT⋆ structure in two complementary settings. In the large-search-space regime, a fast solver is unavailable or too expensive to call at every state. We therefore use structure-aware search to identify high-value complete trajectories, then distill them into the model. Feasibility checks prune invalid actions, while duplicate-action merging avoids wasting search budget on distinct text outputs that correspond to the same structured move. In the small-search-space regime, a solver is available. We use it as a value oracle for partial states: candidate next steps are scored by the best completion reachable after taking that step, and the policy is updated with rank-based advantages. ► **Theoretically:** we analyze optimization-like reasoning capability and how its components are affected as the search space expands.

Why learn reasoning when solvers exist? Even when solvers can verify or solve some generated optimization problems, learning a policy is valuable: it distills reusable optimization heuristics into general-purpose models and can be applied when exact solvers are unavailable, too slow to call repeatedly, or not integrated into the deployment environment.

Contributions

① **(Conceptual)** In §2, we formalize OPT⋆ as a family of *auto-verifiable, scalable* optimization tasks, specifying components for task generation, constraint checking, and outcome evaluation. In §2.1, we discuss what fits in OPT⋆, highlighting task diversity and the reasoning challenge when α increases.

② **(Algorithmic)** In §2.3, we identify the challenges of step-by-step reasoning on OPT⋆. Based on this, we develop complementary offline RL (§3.1) and online RL (§3.2) procedures tailored to OPT⋆.

③ **(Empirical)** In Exp. ⓐ–ⓑ, we show that OPT⋆ improves step-by-step reasoning in optimization tasks. In offline RL, we examine how feasibility pruning via $\texttt{chk}_t(\cdot)$, grouping equivalent actions, and reward shaping based on $\texttt{eval}_t(\cdot)$ guide search toward promising reasoning traces. In online RL, Exp. ⓒ compares terminal and partial-state credit under autonomous evaluation; Exp. ⓓ–ⓕ examine transfer and curriculum effects as α increases.

2 OPTimization-based Scalable Tasks for Auto-verifiable Reasoning (OPT⋆)

Preliminaries. A task instance is a finite- or countable-horizon constrained decision process

$$t = (S, A, P, s_0, \tau, \texttt{chk}_t, \texttt{eval}_t), \tag{1}$$

with state space S, action space A, transition $P(\cdot \mid s, a)$, initial state s_0, and terminal predicate $\tau : S \to \{0, 1\}$. We write the terminal states as $\mathrm{Term}_t := \{s \in S : \tau(s) = 1\}$.

Definition 2.1 (Task feasibility and admissibility)**.** An action a is *admissible* at state s for a task t if $\texttt{chk}_t(s, a) = 1$. We define the set of admissible actions as $\mathcal{A}_t(s) := \{a \in A \mid \texttt{chk}_t(s, a) = 1\}$. A trajectory is *feasible* if it only uses admissible actions (from $\mathcal{A}_t(s)$).

Definition 2.2 (Outcome evaluator)**.** An *outcome evaluator* $\texttt{eval}_t$ is a function that assigns a real-valued score to any terminal state:

$$\texttt{eval}_t : \mathrm{Term}_t \longrightarrow \mathbb{R}$$

Without loss of generality, we assume this score is to be maximized.

Definition 2.3 (Auto-verifiable task)**.** A task t is *auto-verifiable* if there exist algorithms $\texttt{chk}_t$ and $\texttt{eval}_t$ such that, for all state-action pairs (s, a) and terminal states $s_T \in \mathrm{Term}_t$, $\texttt{chk}_t(s, a)$ and $\texttt{eval}_t(s_T)$ can be computed in time polynomial in the input size. Thus, checking feasibility and evaluating a proposed solution are fast, even when finding a high-scoring solution may be hard.

Definition 2.4 (OPTimization-based Scalable Tasks for Auto-verifiable Reasoning)**.** For each complexity level α, let Θ_α be an instance-parameter space and let $\mathcal{D}_\alpha$ be a polynomial-time sampleable distribution over Θ_α. Let $\mathrm{Build}_\alpha : \Theta_\alpha \to \mathcal{T}$ construct an auto-verifiable task:

$$t = \mathrm{Build}_\alpha(\theta) = (S, A, P, s_0, \tau, \texttt{chk}_t, \texttt{eval}_t), \qquad \theta \sim \mathcal{D}_\alpha.$$

We write $t \sim \mathcal{T}_\alpha$ for the induced task distribution. The family $\{\mathcal{T}_\alpha\}$ is *scalable* if there exist an unbounded nondecreasing function g and a small ϵ such that, with probability at least $1 - \epsilon$ over $t \sim \mathcal{T}_\alpha$, (i) a chosen search-complexity measure $M(t)$ satisfies $M(t) \geq g(\alpha)$, and (ii) at least one feasible terminal state is reachable from s_0. Typical choices of $M(t)$ include the logarithm of the number of feasible trajectories or the logarithm of the number of reachable terminal states.

This definition captures the main design goal: as α increases, the model faces a larger reasoning problem, but the task remains automatically checkable and comparable through $\texttt{chk}_t$ and $\texttt{eval}_t$.

2.1 Why OPT⋆?

Motivation. As LLMs are used in more complex workflows, they will need skills beyond standard mathematical derivations and code generation. Optimization under constraints includes resource allocation, scheduling, routing, packing, subset selection, and multi-step coordination. In these settings, there is rarely a single correct answer. Instead, many completions are feasible, and the value of reasoning comes from consistently choosing better trade-offs while satisfying hard constraints.

Synthetic tasks. Any task family satisfying the auto-verifiability and scalability conditions in §2 can instantiate OPT⋆. Such tasks can be designed to target specific reasoning skills—for example, grid-based navigation with obstacles for spatial planning, discrete resource-allocation puzzles for combinatorial trade-offs, or rule-satisfaction games where agents must construct sequences that obey logical constraints (see App. B). By exposing $\texttt{chk}_t$ and $\texttt{eval}_t$ and scaling a complexity parameter α

Task family	Scales in α	Search space $M(t)$	Objective & sample hard rule	Reasoning
Role assignment	roles n; extra cands; conflict	$n!$ perms	Max total fit; e.g. forbidden pairs or ≤ 1 per group.	comb. matching
Task scheduling	jobs n; precedence density	$n!$ job orders	Min total weighted tardiness; e.g. release dates	temporal planning
Max satisfiability	vars n; clauses m; clause mix	2^n assignments	Max satisfied clause weight; e.g. budget $\sum_i x_i \leq B$.	logical reasoning
TSP	cities n	$\approx (n-1)!/2$ tours	Shortest tour visiting all cities; e.g. time windows.	route planning
2D packing	n pieces; bin size; rotations?	super-exponential	Feasible non-overlap packing (or max filled area).	spatial reasoning
QAP	facilities n; flow sparsity	$n!$ assignments	Min flow$\times$distance; e.g. cluster/capacity limits.	comb. optimization

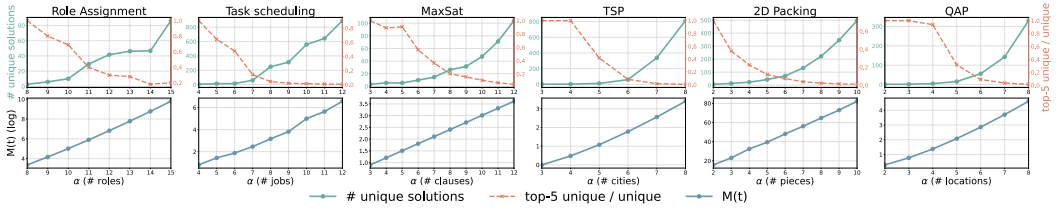


Figure 2: **Traditional optimization tasks. Top**: Task description detailing objectives, constraints, required reasoning, and scaling behavior with α. **Bottom**: Solver experimental results averaged over eight runs. Plots illustrate search space growth, the number of unique solutions, and the fraction representing the top-5 solutions as a function of complexity parameter α.

(e.g., grid size or horizon), these simulators yield large, auto-verifiable curricula. In this paper, we focus on synthetic tasks motivated mainly from traditional optimization problems.

Traditional optimization tasks. Mathematical optimization models are natural OPT$\star$ tasks: decision variables $\boldsymbol{x}$ are selected to maximize an objective $f(\boldsymbol{x})$ subject to equality constraints $h_j(\boldsymbol{x}) = 0$ and inequality constraints $g_i(\boldsymbol{x}) \geq 0$. Their explicit formulations enable direct feasibility checks and objective evaluations, satisfying OPT$\star$ conditions. The task-solving complexity $M(t)$ is decoupled from the complexity parameter α: in the Traveling Salesman Problem (TSP, Fig. 1), for example, increasing the number of cities/nodes (α) dramatically enlarges the set of possible routes ($M(t)$), while route validity and length remain cheap to verify and compute. Here, a single action may fix one or more decision variables, i.e., choose a component or subset of $\boldsymbol{x}$.

Complexity and diversity. Different optimization families elicit different reasoning skills. Routing problems require spatial planning and global consistency. Assignment and matching problems require trade-offs under capacity, preference, and conflict constraints. Scheduling problems require temporal reasoning and coordination. These tasks can also be enriched with additional structure, such as time windows, precedence relations, fairness requirements, or logical clauses, so that the same base family can elicit different forms of reasoning.

Examples. Figure 2 illustrates classical optimization problems and the reasoning capabilities required to solve them. As the complexity parameter α increases, both the search space $M(t)$ (blue line) and the number of unique solutions (teal line) grow, making the *best solution* (orange line) increasingly difficult to find and thereby necessitating more advanced reasoning strategies or heuristics if these tasks were solved by a reasoning model.

2.2 Reasoning on OPT$\star$ Tasks

From environment dynamics to language reasoning. OPT$\star$ tasks are defined as Eq. 1. However, reasoning is performed by an LLM in natural language. To enable this, we expose a language interface that converts states into prompts and parses model outputs back into actions.

Language interface. Let Σ be the token alphabet and let Σ^* denote the set of finite token strings. Let $\mathcal{X} \subseteq \Sigma^*$ denote the set of well-formed prompt states containing the task description, constraints, and relevant history. Let $\tilde{\mathcal{A}} \subseteq \Sigma^*$ denote model outputs containing a reasoning trace and a parsable action. The interface consists of a promptization map $\psi_\alpha : \mathcal{T}_\alpha \times S \to \mathcal{X}$, implemented using a template family Ψ_α, and a deterministic parser $\rho : \tilde{\mathcal{A}} \to A^{\leq K}$ that extracts up to K structured actions from a model output $\tilde{a}_i \sim \pi_\phi(\cdot \mid x_i)$. In short:

$$(t, s) \xrightarrow{\psi_\alpha} x \in \mathcal{X} \xrightarrow{\text{LLM}} \tilde{a} \in \tilde{\mathcal{A}} \xrightarrow{\rho} a \in A^{\leq K}.$$

The prompt $x_i = \psi_\alpha(t, s_i)$ includes: (i) a natural-language task description, (ii) the constraints that must be respected (e.g., admissible actions), and (iii) any auxiliary instructions. When $K = 1$ the model acts step by step; for $K > 1$ it may emit a plan in one shot. Importantly, stating constraints in the prompt does not guarantee compliance, which motivates the method in the next section.

State trajectories. State updates follow the environment dynamics $s_{i+1} \sim P(\cdot|s_i, a_i)$, with current state s_i and action $a_i = \rho(\tilde{a}_i)$. We then reuse ψ_α to construct the next instruction prompt, i.e., $x_{i+1} = \psi_\alpha(t, s_{i+1})$. The language-level evolution mirrors the environment-level trajectory:

$$\underbrace{x_0 \xrightarrow{\tilde{a}_0} x_1 \xrightarrow{\tilde{a}_1} \cdots \xrightarrow{\tilde{a}_{N-1}} x_N}_{\text{Information processed in natural language}} \quad \Longrightarrow \quad \underbrace{s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{N-1}} s_T}_{\text{Information processed by formal programs}} . \tag{2}$$

where $s_T \in \mathrm{Term}_t$ is terminal. For simplicity, in what follows we assume that P is deterministic. LLMs process information in natural language, while formal programs—such as Python environments or simulators—maintain and update environment states, check action feasibility, and evaluate completed solutions. When available, solvers operate on the same formal representations.

Theorem A.3: scaling difficulty without scaling annotation cost

Let $Z^\star = Z^\star(T)$ be the canonical optimum of $T \sim \mathcal{T}_\alpha$, and suppose $Z^\star$ is spread over N_α effective optima. For any reasoner R, let H_j be its history after budget j, and let Y_j be its output. Then

$$\Pr[Y_j = Z^\star] \le (I(Z^\star; H_j) + \log 2) \,/\, \log N_\alpha.$$

Thus, as α increases and the effective number of optima grows, a constant success rate requires either more budget or more information about $Z^\star$ per unit of budget.

We describe empirical search concentration through the effective branching factor. For an ϵ-good terminal set

$$\mathcal{G}_\epsilon(t) = \{s_T : \mathtt{eval}_t(s_T) \ge V_t^\star - \epsilon\},$$

let $p_{\mathrm{hit},R}(t)$ be the probability that a budget-B search procedure R discovers some terminal state in $\mathcal{G}_\epsilon(t)$. Define

$$p_{\epsilon,R}(t) = 1 - (1 - p_{\mathrm{hit},R}(t))^{1/B}, \qquad b_{\mathrm{eff},R}(t) = 1/p_{\epsilon,R}(t).$$

Lower b_{eff} means that the same search budget places more effective mass on high-value terminal states. We use this success-derived quantity as a theorem-motivated diagnostic; it does not independently estimate the $\Omega(\log N_\alpha)$ information requirement.

2.3 What constitutes "better" step-by-step reasoning?

Let $\mathrm{Term}_t(s) \subseteq \mathrm{Term}_t$ be the set of terminal states reachable from s by feasible trajectories with $\mathrm{chk}_t(\cdot, a) = 1$. For deterministic P, the optimal terminal value from any partial state s is given by $V_t^\star(s) := \max_{s_T \in \mathrm{Term}_t(s)} \mathrm{eval}_t(s_T)$. In general, we want to reinforce responses $\tilde{a}$ sampled from the model conditioned on prompt x_i (representing s_i) that lead to a higher downstream value:

$$\tilde{a}_i^\star = \arg\max_{\tilde{a}_i \in \tilde{\mathcal{A}}_t(s_i)} V_t^\star\Big(P(s_i, \rho(\tilde{a}_i))\Big) \tag{3}$$

where $\tilde{\mathcal{A}}_t(s_i) := \{\tilde{a}_{i,g}\}_{g=1}^G$ with $\tilde{a}_{i,g} \sim \pi(\cdot \mid x_i)$. Eq. 3 raises some practical challenges:

- **[P1]** We note that an action $\tilde{a}_i \sim \pi(\cdot \mid x_i)$ does not necessarily imply that $\mathtt{chk}_t(s_i, a_i = \rho(\tilde{a}_i)) = 1$, i.e., responses from the LLM can be invalid, failing to respect the dynamics of the problem.
- **[P2]** Finding the best solution from a partial state s_i, i.e., computing $V_t^\star(s_i)$, requires computing $\max_{s_T \in \mathrm{Term}_t(s)}$ over potentially large space. If these policies are LLM-based, the responses can be redundant, i.e., $\tilde{a}_{i,g_1} \neq \tilde{a}_{i,g_2}$ but $\rho(\tilde{a}_{i,g_1}) = \rho(\tilde{a}_{i,g_2})$.

3 Offline and Online Reinforcement Learning with OPT$\star$

In this section we instantiate the framework from §2 with practical algorithms that optimize Eq. 3. The idea is to turn the inexpensive terminal-only signal from the outcome evaluator $\mathrm{eval}_t(\cdot)$ into informative intermediate feedback that rewards partial decisions and steers subsequent actions. ▶ **Developed techniques:** We explore two ***complementary*** modalities to improve step-by-step reasoning capabilities: offline and online RL. We use offline RL in the general case where a fast solver is unavailable (or too slow), so we rely on the model policy itself to generate many candidate complete solutions that are compared post hoc using $\mathtt{eval}_t(\cdot)$, and we reinforce the best-achieved trajectories. We use online RL when we have access to a solver that can compute, or tightly approximate within a fixed budget, the best completion value from a partial state, enabling fast feedback about which next action is best from a given intermediate state. **Why offline and online RL?** Offline training is broadly applicable because it only requires ranking completed solutions, whereas online

methods can yield stronger improvements via on-policy updates when fast solver-based verification is available [25]. Our offline procedure is closest to search-guided rejection sampling and SFT on evaluator-selected trajectories. We use the term "offline RL" in a broad sense: trajectories are selected using programmatic rewards from $\texttt{eval}_t$, and training is performed on an offline-sampled set of trajectories.

3.1 Offline RL: Search and bootstrapping top solutions with OPT⋆

Training. For offline RL, we simplify Eq. 3 by estimating $V_t^\star(s_0) := \max_{s_T \in \mathrm{Term}_t(s_0)} \mathrm{eval}_t(s_T)$ only at the root state s_0, turning learning into a search problem over trajectories from the root. After search, we *reinforce the best-achieved trajectory* for each problem by supervised fine-tuning on each $(x_i, \tilde{a}_i)$ along the trajectory attaining the highest $\mathrm{eval}_t(\cdot)$, using a rejection-sampling style objective. Other training objectives (e.g., DPO) are also compatible. This procedure distills search-time discoveries into π_ϕ, improving search performance at inference [43, 44] and implicitly maximizing Eq. 3, while avoiding expensive data collection from all nodes in the search tree.

Search. To improve step-by-step reasoning, we approximate $V_t^\star(s) := \max_{s_T \in \mathrm{Term}_t(s)} \mathrm{eval}_t(s_T)$ using a finite set of LLM-generated solutions. In principle, any search method could be used; here we study MCTS and beam search, adapted to OPT⋆ as MCTS⋆ and BeamSearch⋆. We modify their procedures to better balance exploration and exploitation, focusing on child creation to address **[P1–2]**. **[C1]** After expanding a node, we first prune any child with $\texttt{chk}_t(s, a) = 0$ (addressing **[P1]**). **[C2]** We then address **[P2]** by *grouping* children with different textual responses $\tilde{a}_{i,g_1} \neq \tilde{a}_{i,g_2}$ that map to the same structured action $\rho(\tilde{a}_{i,g_1}) = \rho(\tilde{a}_{i,g_2})$, retaining a single response chosen uniformly at random. Note that **[C1–2]** are ***plug-and-play*** methods that can be applied to any search method. We implement two MCTS variants and one beam-search variant, with details in App. F.1; relative to standard implementations, the main difference is that rewards are recomputed as new terminal scores are discovered. In App. A.5, Theorem A.10 explains why these components improve search.

3.2 Online RL: OPT⋆ Solver-Guided Policy Optimization (PO⋆)

We propose **solver-based online RL**, which exploits optimization structure to convert hard-to-verify actions into verifiable subproblems. Our recipe, PO⋆, couples OPT⋆'s feasibility checks and **solver** with policy optimization. For each prompt state, it samples candidate actions and scores the best completion reachable from each next state. Algorithm 1 uses centered pairwise advantages with a clipped policy objective, replacing the usual GRPO advantage estimator [41]. Related objectives include GSPO [63] and PPO [40]; we denote our GRPO-style instance by GRPO⋆.

Algorithm 1: OPT⋆-PO⋆: solver-guided PO

Input: clipped policy objective, group size G
Collect dataset $\mathcal{D}$
for $x \in \mathcal{D}$ **do**
  Sample $\tilde{a}_{1:G} \sim \pi_{\phi_{\text{old}}}(\cdot \mid x)$
  Parse actions $a_{1:G}$ and states $x'_{1:G}$
  for $j = 1$ **to** G **do**
    if a_j parses and $\mathrm{chk}(x, a_j) = 1$ **then**
      $R_j \leftarrow V(x'_j)$
    else
      Mark candidate j invalid
    end if
  end for
  Compare pairs: valid beats invalid; valid scores decide
  Count wins W_j and losses L_j; ties contribute zero
  $A_j \leftarrow (W_j - L_j)/\max(G-1, 1)$
  Update ϕ using A_j on response tokens
end for

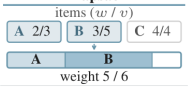
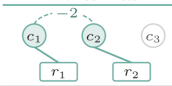
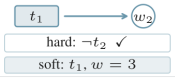
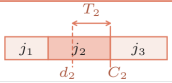
Collecting dataset $\mathcal{D}$. In the solver-prefix setting, each instance contributes one selected state from a solver trajectory. Exp. ⓒ instead generates autonomous trajectories. **Pairwise advantages.** For each fixed prompt, sample G responses. Valid actions receive the exact best-completion score $V(x')$; a valid candidate beats an invalid one, and equal scores or two invalid candidates tie. Candidate j receives $A_j = (W_j - L_j)/(G-1)$ from its pairwise wins and losses, with $A_j = 0$ for a singleton. These centered advantages train response tokens directly, without further mean/std normalization. Appendix F.3 distinguishes the legacy rank mapping; Exp. ⓒ extends the comparison to root-and-turn groups.

4 Related work

Prior work studies verifiable-reward optimization [41, 56], domains verified by compilers/solvers [2, 5, 10, 11, 18, 29, 36, 47, 50, 52, 65], automatic task generation and verification [1, 3, 19, 27, 39, 46], search, iterative refinement, and model formulation [4, 6–9, 14–16, 22, 31, 38, 53, 54, 61, 62], and action selection via information gain [21, 23].

SIRL [13] rewards executable models and solver-interface code, while NP-Engine [28], NPG-Muse [49], and G1 [17] reward directly generated optimization or graph solutions. RLVE [60] studies adaptive verifiable environments, including optimization tasks. These output rewards can

Table 1: Visual overview of OPT⋆ task families studied.

Family	0-1 Knapsack	Role assignment with conflicts	Constrained MaxSAT	Single-machine scheduling	Polyomino target cover
Example					 3 targets covered
Action	Add an unused item	Fill a role with unused candidate	Add a task + eligible worker	Append an unscheduled job	Rotate + place an unused piece
Check	Weight $\leq C$; each item used at most once	One per role; each candidate used at most once	No task/worker reuse; eligibility; budgets; hard clauses	One machine; no idle time; no preemption	In bounds; $\leq K$ placements; no overlap with pieces / obstacles
Score	MAX **total value** $\sum_i v_i x_i$	MAX **total fit** minus penalties	MAX **satisfied** soft-clause weight	MIN **weighted tardiness** $\sum_j w_j T_j$	MAX **newly** covered targets
Scale α	Items; capacity C	Roles; extras; conflict density	Tasks; workers; resources; clauses	Jobs; processing-time range	Grid; piece pool; move budget K

be graded; the distinction is credit timing and assignment. Math-Shepherd [48], AlphaMath [8], and ReST-MCTS* [61] derive process supervision from final-answer correctness in LLM-generated continuations. We contribute (i) conditions for automatic task generation and verification in OPT⋆ along a controlled complexity axis; (ii) experiments on incremental structured solutions and efficient search; and (iii) reusable feasibility pruning **[C1]** and action merging **[C2]** for search-based offline RL, alongside solver-guided online RL. Solver completion values provide objective-aware partial-state credit, with pairwise comparisons grouped by fixed prompt state or autonomous root and turn.

5 Experiments

We study how search-space growth, controlled by complexity α_i, affects the model's task-solving ability and how offline and online RL can mitigate this difficulty. For offline RL, we use solver-computable optima to analyze (1) search components **[C1-2]** that improve search and (2) whether solver-based RL is consistently better. For online RL, we first isolate terminal-only versus partial-state credit under autonomous evaluation, then study transfer and curricula. We organize the research as:

RL	Topic	Question
Offline	Search	*Can structure enable efficient optimization search?* ⓐ
Offline	Solver signal	*Can correct search match solver-guided step improvements?* ⓑ
Online	Credit assignment	*Does intermediate solver credit improve autonomous solutions?* ⓒ
Online	Task design	*Should optimization reasoning be trained on optimization tasks?* ⓓ
Online	Generalization	*Can decomposable spatial tasks capture diverse optimization skills?* ⓔ
Online	Curriculum	*How do different α_i sequences affect learning?* ⓕ

Data generation for training. We use two data-collection protocols.

Training from collected trajectory states. We use four complexity levels $\alpha_1, \ldots, \alpha_4$, generating 1,000 unique instances per task and level. For solver-prefix online training, the configured dataset selects one state per retained instance from its optimal solver trajectory. Candidate actions are policy-sampled, but subsequent prompts remain solver-derived. We concatenate prompts and process levels sequentially, without shuffling. Offline RL instead searches from each root, selects the best discovered trajectory using $\texttt{eval}_t(\cdot)$, and applies SFT to its state–action examples, completing α_i before α_{i+1}. Retaining trajectory edges yields 1,000–5,000 offline examples per level, depending on task and complexity (App. C).

Autonomous trajectory generation. Learning from solver-provided states leaves open whether the policy can construct complete solutions through its own decisions. Exp. ⓒ therefore uses separate fixed-horizon instances and root-started rollouts. The configured training set contains 2,048 unique instances per domain, with 512 at each of four difficulty levels. The model generates one reasoning/action turn at a time, retaining conversation history and receiving generic continuation prompts. The environment validates actions and tracks state internally; it does not return refreshed states or solver feedback to the policy. This produces model-induced trajectories for comparing terminal-only supervision with intermediate solver credit.

Instantiation details. We instantiate offline RL with MCTS⋆ and online RL with GRPO⋆. Solver-prefix training extends `verl` [42] with eight responses per prompt; Appendix F.3 distinguishes legacy and pairwise ranking. Our MCTS⋆ implementation uses 16 rollouts and 20 children per expansion. We evaluate Qwen2.5-3B, Qwen2.5-7B [37], and Llama-3.2-3B-Instruct [35]; the appendix provides prompts, generation settings, and training details.

Table 2: Search-only discovery on OPT⋆ ($\epsilon_{\text{rel}} = 0.05$). Entries are estimates ± paired-bootstrap standard errors from 20,000 resamples of 16 task-by-difficulty cells (App. H.5). Rates are percentages; pass@64 is calibrated restart success. Errors reflect benchmark-cell composition, not seed variability. Bold marks the best point estimate within each family. Table 17 gives all six metrics.

Configuration	MCTS⋆					BeamSearch⋆				
	$p_g \uparrow$	$b_{\text{eff}} \downarrow$	pass@64↑	Feas.↑	Exact↑	$p_g \uparrow$	$b_{\text{eff}} \downarrow$	pass@64↑	Feas.↑	Exact↑
Check + de-duplication	**12.2** ± 3.3	**15.04** ± 2.86	**88.9** ± 1.4	**72.8** ± 6.6	**10.9** ± 3.4	**7.2** ± 1.5	**13.04** ± 2.05	**93.8** ± 1.4	**53.3** ± 3.8	**7.8** ± 2.1
No check/prune	4.0 ± 1.5	25.70 ± 3.52	84.8 ± 1.4	16.4 ± 3.0	3.7 ± 1.5	1.4 ± 0.4	53.07 ± 9.68	65.4 ± 4.3	10.2 ± 1.7	2.0 ± 0.7
No check or de-duplication	5.1 ± 2.0	25.26 ± 3.49	84.6 ± 1.3	19.8 ± 3.6	4.8 ± 2.0	2.5 ± 0.7	82.88 ± 16.12	50.9 ± 4.2	12.8 ± 2.6	2.7 ± 1.0
Sequential	4.9 ± 2.0	25.30 ± 3.56	84.6 ± 1.4	19.8 ± 3.8	4.6 ± 2.0	4.9 ± 2.0	25.30 ± 3.56	84.6 ± 1.4	19.8 ± 3.8	4.6 ± 2.0
Solver reference	0.2 ± 0.1	520.1 ± 135.7	18.7 ± 5.6	2.3 ± 0.9	0.5 ± 0.3	0.2 ± 0.1	499.4 ± 135.6	19.4 ± 5.7	2.5 ± 1.1	0.6 ± 0.4

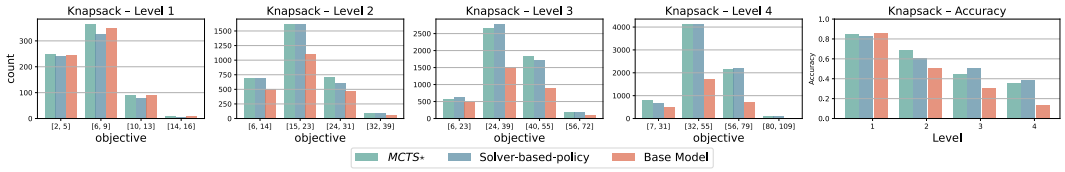


Figure 3: **Cols 1-4:** Distribution of solution quality (histograms). **Col 5:** Accuracy on Knapsack problems across four α_i. The MCTS⋆ method approaches solver-guided performance, particularly at lower complexities. This suggests that verification-guided self-improvement can recover much of the gain from solver-guided supervision.

5.1 Offline RL on OPT⋆

For these experiments we use Llama-3.2-3B-Instruct and evaluate on classical optimization problems. We report aggregate search diagnostics in Table 2, highlight Knapsack in Fig. 3, and provide additional results in App. H.4.

ⓐ Exp 1: Efficient search on OPT⋆. ► Motivation. (A) We first study the no-training regime, testing **[C1-2]**. This isolates the search ***plug-and-play*** components that generate offline trajectories before training. **(B)** Second, we examine consistency with the search-efficiency analysis in Theorem A.10. **►Setup.** We evaluate MCTS⋆ and BeamSearch⋆ (plus one method in App. F.1), with/without **[C1-2]**. A DFS solver serves as reference. Table 2 averages over Role Assig., MaxSat, Knapsack, and QAP. Metrics are the good-terminal mass p_g, effective branching factor b_{eff}, estimated samples for 90% success k_{90}, pass@64, terminal feasibility, and exact optimality.

Results. Table 2 reports search concentration under the component ablations. With both enabled, MCTS⋆ has $b_{\text{eff}} = 15.04$ and BeamSearch⋆ has $b_{\text{eff}} = 13.04$; disabling pruning, with or without duplicate merging, increases the effective branching factor and sharply reduces terminal feasibility and exact optimality. These changes describe greater concentration on high-value terminals under the same budget, consistent with the search-efficiency account in §2.2. Since b_{eff} is computed from good-terminal mass, this comparison is descriptive.

The solver reference in Table 17 is faster in wall-clock time, but assigns little mass to ϵ-good natural-language trajectories: pass@64 is below 20%, versus 88.9% for MCTS⋆ and 93.8% for BeamSearch⋆. QAP is clearest: MCTS⋆ achieves $b_{\text{eff}} = 24.27$ and pass@64 $= 85.1\%$, while the solver reference has $b_{\text{eff}} = 727.9$ and pass@64 $= 13.3\%$. Thus, offline RL is useful when solver labels are unavailable or costly at partial states, whereas online RL uses solver completion values to rank candidate actions. **Obs.** Solvers can be much more efficient than LLMs; LLM methods are most useful when solvers are unavailable and exhaustive DFS-style search becomes costly as complexity grows.

ⓑ Exp 2: Improving models. ►Motivation. We test whether **(A)** training on the best trajectory found by the model with **[C1-2]** improves over the base model and **(B)** solver-assisted training is consistently better. **►Setup.** For knapsack (task details in App. C.1), we report: (a) the cumulative number of distinct solution groups found over 16 MCTS rollouts on the test set for each model, and (b) the percentage of samples recovering the best solution. Since this is an optimization problem, multiple valid solutions may exist. **►Results.** Figure 3 shows difficulty increases with complexity. At higher complexity, MCTS⋆ and solver-based MCTS substantially outperform the base model, especially at α_4, where they find more *high-value solutions*. Solver-based policy performs best, but MCTS⋆ is close, suggesting that components **[C1-2]** largely recover the strongest model-improvement gains.

Table 3: Autonomous four-action optimization. Entries are estimates ± approximate 95% root-cluster confidence margins from 256 held-out roots per domain, with eight rollouts per root. These margins quantify evaluation-root uncertainty, not training-run variability. Rates are percentages; bold marks the best point estimate in each domain, not statistical significance.

Domain	Training signal	Terminal quality		Success and concentration		
		Feasible↑	Exact↑	p_g=pass@1↑	b_{eff} ↓	pass@8↑
Knapsack	Base	12.16 ± 1.86	0.83 ± 0.41	2.98 ± 0.90	14.440 ± 0.598	17.19 ± 4.63
	Outcome RS/SFT	21.48 ± 2.33	2.10 ± 0.65	6.98 ± 1.49	11.295 ± 0.770	33.98 ± 5.82
	Outcome DPO	19.14 ± 2.14	1.51 ± 0.67	5.52 ± 1.31	12.380 ± 0.725	27.34 ± 5.47
	Outcome GRPO	24.27 ± 2.47	1.95 ± 0.61	7.47 ± 1.53	11.024 ± 0.786	34.77 ± 5.84
	Solver-based GRPO	**26.90** ± 3.08	5.62 ± 1.23	14.50 ± 2.45	8.113 ± 0.870	49.61 ± 6.13
	+ ranking	23.14 ± 3.51	**10.45** ± 2.33	**19.19** ± 3.35	**7.394** ± 0.900	**50.00** ± 6.13
Role Assignment	Base	10.89 ± 1.51	1.22 ± 0.47	1.51 ± 0.55	15.867 ± 0.470	10.94 ± 3.82
	Outcome RS/SFT	19.53 ± 2.69	2.73 ± 0.84	3.27 ± 1.04	14.264 ± 0.608	17.97 ± 4.70
	Outcome DPO	30.37 ± 2.37	4.15 ± 0.94	4.79 ± 1.10	12.629 ± 0.706	28.12 ± 5.53
	Outcome GRPO	75.59 ± 2.59	9.03 ± 1.59	10.40 ± 1.84	**9.377** ± 0.831	**44.92** ± 6.10
	Solver-based GRPO	76.46 ± 2.35	7.62 ± 1.49	8.79 ± 1.65	9.977 ± 0.802	43.75 ± 6.10
	+ ranking	**81.40** ± 2.65	**9.33** ± 2.12	**10.50** ± 2.35	10.144 ± 0.823	36.72 ± 5.92

5.2 Online RL with OPT⋆

ⓒ Exp 3: Autonomous optimization and credit assignment. ►Motivation. Does intermediate solver feedback improve the quality of complete solutions when the policy constructs its own trajectory from the root? **►Setup.** New Knapsack and Role Assignment instances require four actions, creating three intermediate states for solver credit. Outcome RS/SFT and DPO use terminal-quality labels or preferences; Outcome GRPO receives reward at termination. Solver-based GRPO uses exact completion-value differences $r_t = V(s_{t+1}) - V(s_t)$, standardized within each root-and-turn group. Its ranking variant substitutes centered pairwise advantages; intermediate states within a group may differ. The three GRPO arms share Qwen2.5-3B-Instruct initialization, group size 8, training-step budget, and token limits, without equating realized token counts or solver compute. Appendix I gives the ordered four-level configuration and implementation details.

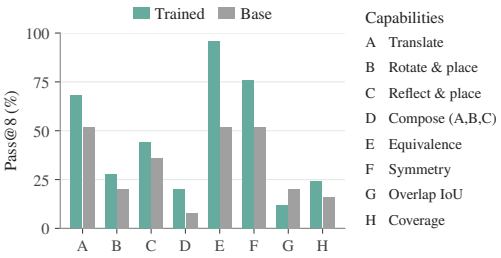
►Evaluation. We use 256 held-out roots per domain and eight autonomous rollouts per root, without solver guidance or search. Table 3 reports terminal feasibility, exact optimality, observed $p_g = \text{pass@1}$ and pass@8, and descriptive b_{eff}. Because b_{eff} and pass@8 use the same good-terminal indicators, their within-set association does not independently establish predictive validity. Pass@8 is measured from the saved rollouts. Confidence margins reflect evaluation-root sampling, not training-run variability. These new fixed-horizon instances differ from the subsequent solver-prefix evaluation, so the scores are not directly comparable. **►Results.** On Knapsack, Solver-based GRPO (ranking) achieves 10.45% exact optimality, 19.19% pass@1, and 50.00% pass@8, versus 1.95%, 7.47%, and 34.77% for Outcome GRPO. Role Assignment is mixed: ranking has the highest feasibility (81.40%), while Outcome GRPO has the highest pass@8 (44.92%, versus 36.72% with ranking). Intermediate credit therefore helps in a task-dependent manner; ranking does not uniformly improve solution quality or multi-sample coverage.

Solver-prefix evaluation. We evaluate model responses from states along solver trajectories using the retained validity-and-reference-value-matching scorer. Pass@k measures this solver-prefix score, separately from autonomous terminal success. Table 4 reports Qwen pass@1; Appendix H compares pass@1 and pass@8 for both 3B models.

Table 4: Pass@1 (%) ± standard error on optimization tasks using Qwen2.5-3B-Instruct. All models are evaluated on the same prompts, with eight responses per prompt. Bold marks the highest scores. See Appendix F.3 for evaluation details.

Method	Role Assignment (ID)				MaxSAT (OOD)				Scheduling (OOD)			
	α_1	α_2	α_3	α_4	α_1	α_2	α_3	α_4	α_1	α_2	α_3	α_4
Base	33.82 ± 0.78	25.57 ± 0.63	21.15 ± 0.57	16.87 ± 0.49	42.11 ± 1.65	29.44 ± 1.45	25.49 ± 1.22	22.36 ± 0.92	57.67 ± 0.87	44.34 ± 0.69	36.30 ± 0.50	37.30 ± 0.49
MATH-500	32.50 ± 0.84	24.89 ± 0.66	20.65 ± 0.56	16.58 ± 0.49	41.67 ± 1.69	27.72 ± 1.35	24.82 ± 1.20	21.09 ± 0.86	57.60 ± 0.82	44.47 ± 0.63	35.94 ± 0.49	37.36 ± 0.52
Code	33.35 ± 0.79	25.77 ± 0.66	20.28 ± 0.56	16.56 ± 0.48	41.94 ± 1.66	29.44 ± 1.39	25.18 ± 1.15	21.81 ± 0.91	58.12 ± 0.82	44.61 ± 0.67	36.07 ± 0.50	37.39 ± 0.52
Trained	**43.53** ± 0.91	**37.31** ± 0.65	**33.47** ± 0.58	**28.61** ± 0.56	**46.82** ± 1.73	**35.09** ± 1.68	**30.87** ± 1.40	**25.09** ± 1.05	**60.08** ± 0.96	**46.17** ± 0.76	**38.96** ± 0.60	**39.14** ± 0.59

ⓓ Exp 4: Domain training and transfer. ►Motivation. We ask whether optimization-specific training improves over verifiable math or code RL. **►Setup.** We compare pairwise training on Role Assignment with ordinary GRPO on MATH-500 [19, 30] (500 problems, two epochs) and MBPP [5] (four epochs). We evaluate Qwen2.5-3B-Instruct on Role Assignment (ID), MaxSAT, and Machine Scheduling (OOD) using the same frozen solver-prefix evaluation. **►Results.** In Table 4, math/code training remains close to the base model, while pairwise Role Assignment training improves all twelve



Evaluation task	Dist.	Training set: Base	Training set: α_1–α_4 1k each	Training set: α_1 only 4k	Training set: α_4 only 4k
Polyomino α_1	ID	4.4 ± 1.30	10.0 ± 1.90	**32.4** ± 2.97	4.0 ± 1.24
Polyomino α_2	ID	2.4 ± 0.97	22.8 ± 2.66	**38.4** ± 3.08	2.4 ± 0.97
Polyomino α_3	ID	5.3 ± 1.03	**22.5** ± 1.69	17.1 ± 1.59	3.6 ± 0.86
Polyomino α_4	ID	1.2 ± 0.39	**5.2** ± 0.77	2.5 ± 0.56	0.4 ± 0.23
Spatial QA (diff 1)	OOD	32.0 ± 3.13	**46.0** ± 2.95	28.0 ± 2.84	29.5 ± 3.00
Spatial QA (diff 2)	OOD	24.5 ± 2.84	**34.0** ± 2.93	31.0 ± 2.77	27.5 ± 3.05

Figure 4: **2D spatial reasoning. Left:** Spatial QA (difficulty 1), base versus mixed-curriculum training. **Right:** Curriculum comparison: pass@8 (%) ± evaluation SE (percentage points). Bold marks the best means. SEs reflect evaluation uncertainty for saved checkpoints, not training-run variability; Appendix G.1 details the clustering and stratification.

pass@1 point estimates. These domain comparisons use different training datasets and budgets; Appendix F.3 details the protocol and scorer limitations.

ⓔ **Exp 5: Generalization (spatial tasks).** ►**Motivation.** Optimization tasks can involve heterogeneous constraints, and solving them can require a broad set of skills. We demonstrate this using a 2D polyomino task (Table 1), where models must select, rotate, and translate pieces to cover target cells. ►**Setup.** Qwen2.5-7B was the only model with sufficient capacity to sample rewardable solutions and bootstrap RL in this task. We evaluate acquired capabilities on (1) the held-out polyomino test set and (2) auxiliary spatial QA tasks covering translation, rotation, reflection, symmetry, and coverage. These auxiliary tasks are QA pairs rather than optimization problems, with two difficulty levels based on grid size (details in App. C). ►**Results.** Fig. 4 shows that the curriculum improves both in-distribution polyomino performance and out-of-distribution spatial QA, suggesting transfer to the underlying spatial operations.

Generalization (Math). GRPO training on Role Assignment+MaxSAT yields higher mean pass@1 on AMC'23, AIME'24, and AIME'25 [32–34] for both Qwen2.5-3B and Llama-3.2-3B, averaged over three decoding seeds. App. G.2 reports the pass@1 and pass@8 results with standard errors.

ⓕ **Exp 6: How the α_i curriculum affects learned skills.** We study curriculum learning on the polyomino task by varying the complexity parameter α_i. Using OPT⋆ to generate data across complexity regimes, we compare three training strategies: easy-only (α_1), hard-only (α_4), and mixed curriculum training ($\alpha_1 \to \alpha_4$). Figure 4 (right) shows three trends. First, hard-only training underperforms, consistent with sparse rewards on hard tasks. Second, easy-only training is strong on α_1 and α_2 but degrades as difficulty increases. Third, curriculum training gives the best performance on the harder ID levels and the strongest OOD Spatial QA results. These gains indicate that curriculum training improves the underlying spatial skills more reliably than single-level training.

6 Conclusion

We propose OPT⋆ to elicit optimization-like reasoning. We study how enlarging the search space affects such reasoning (Theorem A.3) and how training helps find high-value solutions. This view motivates two ***complementary*** methods: (1) ***plug-and-play* search components [C1–2]**, using feasibility checking and action merging to improve search efficiency, supported by theoretical analysis and search ablations; and (2) **solver-based RL**, using available fast solvers to provide objective-aware rewards for partial actions. Solver-prefix experiments show gains over domain-transfer references. In autonomous four-action experiments, solver-based credit with ranking improves Knapsack exact-optimality and success rates over terminal-only GRPO; Role Assignment reveals a feasibility–coverage tradeoff. These benefits remain task-dependent. Curriculum learning and generalization through scalable task generation remain promising directions.

Limitations and future work. A promising direction is to ***scale optimization reasoning*** using LLM priors to automate the generation of more complex and diverse tasks while evaluation remains simple. Scaling task generation and complexity could broaden this testbed for optimization-based generalization. The autonomous comparison covers only two synthetic domains, and its uncertainty estimates concern evaluation instances rather than independent training runs. Solver availability and partial-state evaluation costs constrain solver-based RL.

Acknowledgments and Disclosure of Funding

We thank the anonymous NeurIPS reviewers and members of the van der Schaar lab for their insightful comments and suggestions. Nicolas Astorga thanks the W. D. Armstrong Trust for its support and is funded by the Cystic Fibrosis Trust. This work was supported by Microsoft's Accelerate Foundation Models Academic Research initiative. The authors declare no additional competing financial interests.

References

[1] Alon Albalak, Duy Phung, Nathan Lile, Rafael Rafailov, Kanishk Gandhi, Louis Castricato, Anikait Singh, Chase Blagden, Violet Xiang, Dakota Mahan, and Nick Haber. Big-math: A large-scale, high-quality math dataset for reinforcement learning in language models, 2025. URL https://arxiv.org/abs/2502.17387.

[2] AlphaProof and AlphaGeometry teams. Ai achieves silver-medal standard solving international mathematical olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, July 2024. Accessed 2025-09-25.

[3] Nicolás Astorga, Anita Kriz, and Mihaela van der Schaar. Scaling Causal Reasoning with Increasingly Complex Causal Simulators. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2026. URL https://neurips.cc/virtual/2026/poster/148175.

[4] Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of Mathematical Optimization Models Using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 1864–1886. PMLR, 2025. URL https://proceedings.mlr.press/v267/astorga25a.html.

[5] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[6] 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, 2024.

[7] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi: 10.1109/TCIAIG.2012.2186810.

[8] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process, 2024. URL https://arxiv.org/abs/2405.03553.

[9] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7889–7903, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.463. URL https://aclanthology.org/2024.findings-emnlp.463/.

[10] Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Qiying Yu, Xuefeng Li, Jiaze Chen, et al. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. *arXiv preprint arXiv:2505.19914*, 2025.

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[12] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[13] Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-Informed RL: Grounding Large Language Models for Authentic Optimization Modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2505.11792.

[14] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games: CG 2006*, volume 4630 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2007. doi: 10.1007/978-3-540-75538-8_7.

[15] Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like Tree-Search can Guide Large Language Model Decoding and Training. *arXiv preprint arXiv:2309.17179*, 2023. URL https://arxiv.org/abs/2309.17179.

[16] 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. *arXiv preprint arXiv:2501.04519*, 2025.

[17] Xiaojun Guo, Ang Li, Yifei Wang, Stefanie Jegelka, and Yisen Wang. G1: Teaching LLMs to Reason on Graphs with Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2505.18499.

[18] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. arXiv:2105.09938.

[19] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[20] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STaR: Training verifiers for self-taught reasoners. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=stmqBSW2dV.

[21] Katarzyna Kobalczyk, Nicolás Astorga, Tennison Liu, and Mihaela van der Schaar. Active Task Disambiguation with LLMs. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5e07476b6bd2497e1fbd11b8f0b2de3c-Abstract-Conference.html.

[22] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer, 2006. doi: 10.1007/11871842_29.

[23] Anita Kriz, Antonin Berthon, Nicolás Astorga, and Mihaela van der Schaar. Amortizing Sequential Experimental Design in Reasoning LLMs. *arXiv preprint arXiv:2610.20832*, 2026. URL https://arxiv.org/abs/2610.20832.

[24] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

[25] Jack Lanchantin, Angelica Chen, Janice Lan, Xian Li, Swarnadeep Saha, Tianlu Wang, Jing Xu, Ping Yu, Weizhe Yuan, Jason E Weston, et al. Bridging offline and online reinforcement learning for llms. *arXiv preprint arXiv:2506.21495*, 2025.

[26] Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2665–2679. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.150. URL https://aclanthology.org/2023.acl-long.150/.

[27] Peiji Li, Jiasheng Ye, Yongkang Chen, Yichuan Ma, Zijie Yu, Kedi Chen, Xiaozhe Li, Ganqu Cui, Haozhan Li, Jiacheng Chen, Chengqi Lyu, Wenwei Zhang, Linyang Li, Qipeng Guo, Dahua Lin, Bowen Zhou, and Kai Chen. InternBootcamp technical report: Boosting LLM reasoning with verifiable task scaling. *arXiv preprint arXiv:2508.08636*, 2025. URL https://arxiv.org/abs/2508.08636v2.

[28] Xiaozhe Li, Xinyu Fang, Shengyuan Ding, Linyang Li, Haodong Duan, Qingwen Liu, and Kai Chen. NP-Engine: Empowering Optimization Reasoning in Large Language Models with Verifiable Synthetic NP Problems. *arXiv preprint arXiv:2510.16476*, 2025. URL https://arxiv.org/abs/2510.16476.

[29] 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, 2022. doi: 10.1126/science.abq1158. arXiv:2203.07814.

[30] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

[31] Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

[32] Mathematical Association of America. American mathematics competitions (AMC) 2023. Mathematical Association of America, 2023. Benchmark dataset distributed by `math-ai`. URL https://huggingface.co/datasets/math-ai/amc23. Accessed 2026-11-07.

[33] Mathematical Association of America. American invitational mathematics examination (AIME) 2024. Mathematical Association of America, 2024. Benchmark dataset by Yifan Zhang and the Math-AI Team. URL https://huggingface.co/datasets/math-ai/aime24. Accessed 2026-11-07.

[34] Mathematical Association of America. American invitational mathematics examination (AIME) 2025. Mathematical Association of America, 2025. Benchmark dataset by Yifan Zhang and the Math-AI Team. URL https://huggingface.co/datasets/math-ai/aime25. Accessed 2026-11-07.

[35] Meta. Llama 3.2 3b instruct model card. Hugging Face model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct.

[36] Arkil Patel, Siva Reddy, and Dzmitry Bahdanau. How to get your llm to generate challenging problems for evaluation. *arXiv preprint arXiv:2502.14678*, 2025.

[37] Qwen Team. Qwen2.5 technical report, 2024. URL https://arxiv.org/abs/2412.15115.

[38] Paulius Rauba, Claudio Fanconi, and Mihaela van der Schaar. Tiny Autoregressive Recursive Models. In *ICLR 2026 Workshop on AI with Recursive Self-Improvement*, 2026. URL https://arxiv.org/abs/2603.08082.

[39] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1904.01557.

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

[41] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[42] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*. ACM, 2025. doi: 10.1145/3689031.3696075.

[43] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[44] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[45] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023. URL https://arxiv.org/abs/2312.06585.

[46] Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. REASONING GYM: Reasoning environments for reinforcement learning with verifiable rewards, 2025. URL https://arxiv.org/abs/2505.24760.

[47] Trieu Trinh, Yuhuai Wu, Quoc Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 2024. doi: 10.1038/s41586-023-06747-5.

[48] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510/.

[49] Yuyao Wang, Bowen Liu, Jianheng Tang, Nuo Chen, Yuhan Li, Qifan Zhang, Chenyi Zi, Chen Zhang, and Jia Li. NPG-Muse: Scaling Long Chain-of-Thought Reasoning with NP-Hard Graph Problems. *arXiv preprint arXiv:2508.20373*, 2025. URL https://arxiv.org/abs/2508.20373.

[50] Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. *arXiv preprint arXiv:2505.14615*, 2025.

[51] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[52] Zhen Hao Wong, Jingwen Deng, Runming He, Zirong Chen, Qijie You, Hejun Dong, Hao Liang, Chengyu Shen, Bin Cui, and Wentao Zhang. Logicpuzzlerl: Cultivating robust mathematical reasoning in llms via reinforcement learning. *arXiv preprint arXiv:2506.04821*, 2025.

[53] 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. *arXiv preprint arXiv:2405.00451*, 2024.

[54] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[55] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

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

[57] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023. URL https://arxiv.org/abs/2308.01825.

[58] Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language models can teach themselves to think before speaking. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=oRXPiSOGH9.

[59] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

[60] Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. RLVE: Scaling Up Reinforcement Learning for Language Models with Adaptive Verifiable Environments. *arXiv preprint arXiv:2511.07317*, 2025. URL https://arxiv.org/abs/2511.07317.

[61] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*, 2024.

[62] Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*, 2024.

[63] Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

[64] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WZH7099tgfM.

[65] Qin Zhu, Fei Huang, Runyu Peng, Keming Lu, Bowen Yu, Qinyuan Cheng, Xipeng Qiu, Xuanjing Huang, and Junyang Lin. Autologi: Automated generation of logic puzzles for evaluating reasoning abilities of large language models. *arXiv preprint arXiv:2502.16906*, 2025.

Appendix Index

Quick guide to the appendix

Ref.	Content
A	Information-theoretic formalization of reasoning on OPT⋆
A.1	Effective optima and scalable complexity
A.2	Reasoners, histories, and success
A.3	Core information-theoretic results
A.4	Effective branching and empirical estimation
A.5	Checking and redundancy removal
B	Task examples
B.1	Mathematical optimization tasks
B.2	Logic puzzle tasks
B.3	Simulator-based tasks
C	Additional details on the generation process
C.1	0–1 Knapsack
C.2	Quadratic Assignment / CF–EU Manhattan
C.3	Role Assignment with Conflicts
C.4	MaxSat
C.5	Single-machine scheduling
C.6	Polyomino Target Cover
C.7	Auxiliary Grid-Spatial tasks
D	Prompt examples
E	Polyomino examples
F	Additional details and extra results
F.1	Search ablations and additional details of MCTS⋆
F.2	Policy training implementation for RFT
F.3	Policy training GRPO
G	Online RL results
G.1	Spatial reasoning results
G.2	Math benchmark results
H	Optimization problem results
H.1	Role Assignment results
H.2	MaxSAT results
H.3	Scheduling results
H.4	Additional MCTS results
H.5	Search diagnostics and complete Table 2 point estimates
I	Autonomous four-action control: protocol details
J	Broader impacts

A Information-theoretic formalization of reasoning on OPT⋆

Scope and notation. All logarithms are natural, so information is measured in nats. We use the OPT⋆ task definition from the main paper and add only the probabilistic notation needed for the theory. For a task t, let $\mathrm{Term}_t(s)$ denote the set of terminal states reachable from state s by feasible trajectories. We write s_0 for the initial state and use ξ for a complete rollout, with terminal state $s_T(\xi)$.

The results below are stated for finite effective solution spaces. This covers the finite combinatorial tasks used in our experiments. Countable or very large spaces can be handled by truncating the search space or by grouping terminal states into finitely many effective equivalence classes.

A.1 Effective optima and scalable complexity

For a realized task t, let

$$\mathcal{Z}_t := \mathrm{Term}_t(s_0)$$

be its reachable terminal set, or a finite quotient of that set into effective solution classes. Fix a deterministic tie-breaking rule $\mathrm{tb}(\cdot)$ that selects one element from any nonempty finite set. The canonical optimal terminal state is

$$Z^\star(t) := \mathrm{tb}\left(\arg\max_{z\in\mathcal{Z}_t} \mathtt{eval}_t(z)\right), \qquad V_t^\star := \mathtt{eval}_t(Z^\star(t)).$$

When $T \sim \mathcal{T}_\alpha$, the canonical optimum

$$Z^\star := Z^\star(T)$$

is a random variable.

Assumption A.1 (Effective uniform optimum)**.** For each complexity level α considered in the theorem statements, there exists a finite set $\mathcal{Z}_\alpha^\star$ of size $N_\alpha \geq 2$ such that

$$Z^\star \sim \mathrm{Unif}(\mathcal{Z}_\alpha^\star).$$

Equivalently,

$$H(Z^\star) = \log N_\alpha.$$

The same assumption can be applied after replacing terminal states by effective solution classes, for example after deterministic tie-breaking, symmetry reduction, or ϵ-optimal grouping.

A.2 Reasoners, histories, and success

A reasoner R may be an LLM policy, a search procedure using an LLM, or a hybrid method such as beam search, MCTS, feasibility-pruned search, or solver-guided search. After budget j, the reasoner has produced a history

$$H_j.$$

This history may contain sampled text actions, parsed structured actions, checker outputs, duplicate-removal decisions, terminal candidates, terminal values, and internal search statistics.

Let

$$\mathcal{Z}_R(t,j) \subseteq \mathrm{Term}_t(s_0)$$

be the set of feasible terminal states discovered by R on task t by budget j. The reasoner's output is denoted by Y_j. When at least one terminal state has been found, we take

$$Y_j := \mathrm{tb}\left(\arg\max_{z\in\mathcal{Z}_R(t,j)} \mathtt{eval}_t(z)\right).$$

When no terminal state has been found, we set $Y_j = \perp$. In all cases, Y_j is measurable with respect to H_j.

For $\epsilon \geq 0$, define the anytime success curve

$$S_R(j;\alpha,\epsilon) := \Pr_{T\sim\mathcal{T}_\alpha,R}\left[\exists z \in \mathcal{Z}_R(T,j) : \mathtt{eval}_T(z) \geq V_T^\star - \epsilon\right].$$

For exact optimality, take $\epsilon = 0$.

The reasoning information accumulated by budget j is

$$\mathcal{I}_R(j;\alpha) := I(Z^\star; H_j).$$

If the history is revealed incrementally as $H_j = (O_1, \ldots, O_j)$, define the per-step information gain

$$\mathrm{IG}_i := I(Z^\star; O_i \mid H_{i-1}).$$

By the chain rule,

$$\mathcal{I}_R(j;\alpha) = \sum_{i=1}^{j} \mathrm{IG}_i. \tag{4}$$

A.3 Core information-theoretic results

Lemma A.2 (Fano-style bound for identifying the optimum)**.** *Assume Assumption A.1. For any estimator Y of $Z^\star$,*

$$\Pr[Y = Z^\star] \le \frac{I(Z^\star; Y) + \log 2}{\log N_\alpha}.$$

Equivalently, if

$$\Pr[Y = Z^\star] \ge 1 - \delta,$$

then

$$I(Z^\star; Y) \ge (1-\delta)\log N_\alpha - \log 2.$$

Proof. Let

$$P_e := \Pr[Y \ne Z^\star].$$

Since $Z^\star$ is uniform on N_α possibilities,

$$H(Z^\star) = \log N_\alpha.$$

Let $E = \mathbf{1}\{Y \ne Z^\star\}$. Then

$$H(Z^\star \mid Y) \le H(E \mid Y) + H(Z^\star \mid Y, E) \le \log 2 + P_e \log N_\alpha.$$

Therefore,

$$I(Z^\star; Y) = H(Z^\star) - H(Z^\star \mid Y) \ge \log N_\alpha - \log 2 - P_e \log N_\alpha.$$

Since $1 - P_e = \Pr[Y = Z^\star]$, rearranging gives

$$\Pr[Y = Z^\star] \le \frac{I(Z^\star; Y) + \log 2}{\log N_\alpha}.$$

The second statement follows by rearranging the same inequality. □

Theorem A.3 (Complexity–reasoning tradeoff for OPT⋆)**.** *Assume Assumption A.1. Consider any reasoner R that, after budget j, outputs Y_j measurable with respect to H_j. Then*

$$\Pr[Y_j = Z^\star] \le \frac{\mathcal{I}_R(j;\alpha) + \log 2}{\log N_\alpha} = \frac{\sum_{i=1}^{j} \mathrm{IG}_i + \log 2}{\log N_\alpha}.$$

Consequently, maintaining any fixed positive probability of identifying the canonical optimum as N_α grows requires

$$\mathcal{I}_R(j;\alpha) = \Omega(\log N_\alpha).$$

Thus, as the effective search space expands, a reasoner must either use more budget or extract more information per unit of budget.

Proof. Apply Lemma A.2 with $Y = Y_j$. Since Y_j is measurable with respect to H_j, the data-processing inequality gives

$$I(Z^\star; Y_j) \le I(Z^\star; H_j) = \mathcal{I}_R(j;\alpha).$$

Substituting this into Lemma A.2 yields

$$\Pr[Y_j = Z^\star] \le \frac{\mathcal{I}_R(j;\alpha) + \log 2}{\log N_\alpha}.$$

The equality with $\sum_{i=1}^{j} \mathrm{IG}_i$ follows from (4). The asymptotic statement follows by rearranging the bound for any fixed nonzero target success probability. □

Corollary A.4 (Budget lower bound)**.** *Assume Assumption A.1. Suppose there exists $\kappa_\alpha > 0$ such that, for all $i \le j$,*

$$\mathrm{IG}_i = I(Z^\star; O_i \mid H_{i-1}) \le \kappa_\alpha.$$

Then

$$\Pr[Y_j = Z^\star] \le \frac{j\kappa_\alpha + \log 2}{\log N_\alpha}.$$

Equivalently, achieving

$$\Pr[Y_j = Z^\star] \ge 1 - \delta$$

requires

$$j \ge \frac{(1-\delta)\log N_\alpha - \log 2}{\kappa_\alpha},$$

whenever the numerator is positive.

Proof. By the chain rule,

$$\mathcal{I}_R(j;\alpha) = \sum_{i=1}^{j} \mathrm{IG}_i \le j\kappa_\alpha.$$

Substituting this into Theorem A.3 gives the first claim. The budget lower bound follows by rearranging. □

Approximate optimality. The same argument applies to ϵ-optimality by replacing $Z^\star$ with a finite effective class variable that indexes the relevant ϵ-good solution region. In that case, N_α should be read as the number of effective high-value regions the reasoner must distinguish.

A.4 Effective branching and empirical estimation

Theorem A.3 states that a harder effective search space requires more informative histories. We now connect this to the empirical quantities used in the experiments.

Definition A.5 (ϵ-good terminal set)**.** For a task instance t and tolerance $\epsilon \ge 0$, define

$$\mathcal{G}_\epsilon(t) = \{s_T \in \mathrm{Term}_t(s_0) : \texttt{eval}_t(s_T) \ge V_t^\star - \epsilon\}.$$

When the exact optimum $V_t^\star$ is unavailable, it can be replaced by a reference value obtained from the best solution found under a large search budget. The resulting quantity should then be interpreted as reference-good mass rather than true ϵ-optimal mass.

Definition A.6 (Budgeted hit probability)**.** Fix a search budget B. For a possibly adaptive search procedure R, define

$$p_{\mathrm{hit},R}(t;B) = \Pr_R\left[\mathcal{Z}_R(t,B) \cap \mathcal{G}_\epsilon(t) \ne \emptyset \mid t\right].$$

Equivalently,

$$p_{\mathrm{hit},R}(t;B) = \Pr_R\left[\max_{z \in \mathcal{Z}_R(t,B)} \texttt{eval}_t(z) \ge V_t^\star - \epsilon \mid t\right],$$

with the convention that the maximum over an empty discovered set is $-\infty$.

Definition A.7 (Good-terminal mass and effective branching)**.** For fixed budget B, define the budget-normalized good-terminal mass

$$p_{\epsilon,R}(t;B) = 1 - (1 - p_{\mathrm{hit},R}(t;B))^{1/B}.$$

The empirical effective branching factor is

$$b_{\mathrm{eff},R}(t;B) = \frac{1}{p_{\epsilon,R}(t;B)},$$

with the convention that $b_{\mathrm{eff},R}(t;B) = +\infty$ when $p_{\epsilon,R}(t;B) = 0$. When B is fixed by the experiment, we omit B and write $p_{\mathrm{hit},R}(t)$, $p_{\epsilon,R}(t)$, and $b_{\mathrm{eff},R}(t)$.

Interpretation. The quantity $p_{\epsilon,R}(t;B)$ is a calibration, not an independence assumption. It asks: what independent per-budget-unit success probability would yield the same budget-B hit probability? Thus it can compare plain rollouts, beam search, MCTS, feasibility-pruned search, and duplicate-aware search under a common scale.

Proposition A.8 (Independent restarts as a special case)**.** *Suppose R consists of B independent rollouts, and a single rollout hits $\mathcal{G}_\epsilon(t)$ with probability p. Then*

$$p_{\mathrm{hit},R}(t;B) = 1-(1-p)^B,$$

and therefore

$$p_{\epsilon,R}(t;B) = p.$$

Proof. The probability that all B independent rollouts miss the ϵ-good set is $(1-p)^B$. Taking the complement gives $p_{\mathrm{hit},R}(t;B) = 1-(1-p)^B$. Substituting into Definition A.7 gives $p_{\epsilon,R}(t;B) = p$. □

Proposition A.9 (Hit curves equal anytime success)**.** *For any possibly adaptive reasoner R,*

$$S_R(B;\alpha,\epsilon) = \mathbb{E}_{T\sim\mathcal{T}_\alpha}\left[p_{\mathrm{hit},R}(T;B)\right].$$

Proof. Condition on $T=t$. By Definition A.6, $p_{\mathrm{hit},R}(t;B)$ is exactly the conditional probability that R discovers an ϵ-good terminal state by budget B. Taking expectation over $T\sim\mathcal{T}_\alpha$ gives the result. □

Empirical estimation. For each held-out task instance t, run R independently m times with budget B. Let

$$c_t = \sum_{\ell=1}^{m} \mathbf{1}\left[\mathcal{Z}_R^{(\ell)}(t,B)\cap\mathcal{G}_\epsilon(t)\neq\emptyset\right].$$

Then

$$\widehat{p}_{\mathrm{hit},R}(t;B) = \frac{c_t}{m}.$$

The corresponding estimates are

$$\widehat{p}_{\epsilon,R}(t;B) = 1-\left(1-\widehat{p}_{\mathrm{hit},R}(t;B)\right)^{1/B}, \qquad \widehat{b}_{\mathrm{eff},R}(t;B) = \frac{1}{\widehat{p}_{\epsilon,R}(t;B)}.$$

For numerical stability when $c_t=0$, one may use the smoothed estimate

$$\widetilde{p}_{\mathrm{hit},R}(t;B) = \frac{c_t+1/2}{m+1}.$$

Connection to the main theorem. Theorem A.3 applies to the full history H_B of any search procedure, including adaptive methods such as MCTS or beam search. The empirical hit curve measures whether the search procedure places enough probability mass on high-value terminal states within the available budget. Thus, at fixed B, better search or a stronger learned policy should increase $p_{\mathrm{hit},R}$, increase $p_{\epsilon,R}$, and decrease $b_{\mathrm{eff},R}$. These success-based diagnostics do not estimate $\mathcal{I}_R$ or N_α, and therefore do not independently test the information bound.

A.5 Checking and redundancy removal

We now isolate the local effect of the two search components used in the paper: feasibility checking and duplicate structured-action removal.

Theorem A.10 (Checker and redundancy removal increase trace efficiency)**.** *Consider a fixed state with feasible structured action classes*

$$\mathcal{F}, \qquad |\mathcal{F}| = K \geq 2.$$

Let W be the structured action class whose continuation is optimal under a fixed tie-breaking rule, and assume

$$W \sim \mathrm{Unif}(\mathcal{F}).$$

A raw text action $\tilde{A}_i$ is parsed into a structured class

$$C_i = \rho(\tilde{A}_i).$$

The test succeeds when $C_i = W$.

Compare two procedures with $j \leq K$:

- ***Baseline without P1/P2:*** *draw j raw text actions from a fixed distribution independent of W. The parsed classes may be infeasible or repeated.*
- ***P1+P2:*** *use the checker to reject infeasible classes and the parser to reject repeated feasible classes, until j distinct feasible classes have been accepted.*

Let H_j denote the resulting transcript. For the baseline, let

$$U_j = |\{C_1, \ldots, C_j\} \cap \mathcal{F}|$$

be the number of distinct feasible classes tested. Then

$$\Pr[\textit{baseline hits } W] = \mathbb{E}\left[\frac{U_j}{K}\right] \leq \frac{j}{K} = \Pr[\textit{P1+P2 hits } W],$$

and

$$I(W; H_j^{\mathrm{P1+P2}}) \geq I(W; H_j^{\mathrm{base}}).$$

The success inequality is strict whenever infeasible or repeated classes make $\mathbb{E}[U_j] < j$. The information inequality is strict whenever the baseline leaves at least two feasible classes untested with positive probability.

Proof. Let

$$S_j := \{C_1, \ldots, C_j\} \cap \mathcal{F}$$

be the set of distinct feasible classes tested by the baseline, and let $U_j = |S_j|$. Infeasible classes cannot equal W, and repeated classes retest the same event.

Success probability. Conditioned on S_j, the baseline succeeds if and only if $W \in S_j$. Since W is uniform over $\mathcal{F}$,

$$\Pr[\text{baseline hits } W \mid S_j] = \frac{U_j}{K}.$$

Taking expectations gives

$$\Pr[\text{baseline hits } W] = \mathbb{E}\left[\frac{U_j}{K}\right] \leq \frac{j}{K}.$$

Under P1+P2, every accepted test is a new feasible class, so $U_j = j$ deterministically. Therefore,

$$\Pr[\text{P1+P2 hits } W] = \frac{j}{K}.$$

Mutual information. For $0 \leq u \leq K$, define

$$\phi_K(u) = \begin{cases} \left(1 - \frac{u}{K}\right) \log(K - u), & 0 \leq u \leq K - 1, \\ 0, & u = K. \end{cases}$$

This function is nonincreasing on the integer set $\{0, \ldots, K\}$. Conditioned on the tested set S_j, if a hit occurs then W is identified exactly. If no hit occurs, then W is uniform over the $K - U_j$ feasible classes in $\mathcal{F} \setminus S_j$. Hence

$$H(W \mid H_j^{\mathrm{base}}) = \mathbb{E}[\phi_K(U_j)].$$

Since $U_j \leq j$ and ϕ_K is nonincreasing,

$$H(W \mid H_j^{\mathrm{base}}) \geq \phi_K(j).$$

Under P1+P2, $U_j = j$ deterministically, so

$$H(W \mid H_j^{\mathrm{P1+P2}}) = \phi_K(j).$$

Thus

$$H(W \mid H_j^{\mathrm{base}}) \geq H(W \mid H_j^{\mathrm{P1+P2}}).$$

Since $H(W) = \log K$, this is equivalent to

$$I(W; H_j^{\mathrm{P1+P2}}) \geq I(W; H_j^{\mathrm{base}}).$$

The strictness statements follow from the same inequalities. □

Corollary A.11 (Non-redundant search dominates restart search)**.** *Take $\mathcal{F} = \{1, \ldots, N\}$, so every structured class is feasible. A search procedure that tests j distinct previously untested classes has success probability j/N, which is at least the success probability of a restart sampler that may repeat classes. It also has at least as much mutual information about the optimal class.*

Proof. This is Theorem A.10 with no infeasible classes. Duplicate removal makes the accepted tests distinct; restart sampling may repeat classes, so its number of distinct tested classes is at most j. □

Remark A.12 (Fixed raw budgets). Theorem A.10 counts accepted feasible structured tests. In the full search system, the raw generation budget is fixed. The theorem therefore gives a local explanation: P1 and P2 improve search when they convert raw generations into more distinct feasible tests. The empirical quantities $p_{\mathrm{hit},R}$ and $b_{\mathrm{eff},R}$ measure whether this local advantage translates into better budgeted search.

B Task examples

What fits OPT⋆? Any multi-step task family with a fast step-checker and a fast final evaluator, whose difficulty scales with a parameter α; this spans optimization, logic/constraint puzzles, and simulator-backed domains.

B.1 Mathematical optimization tasks

Task	State / Actions	D1: Checker (step)	D2: Evaluator (final)	D3: Stop	D4: Scale (α)	Typical Solver
Knapsack	Subset of items; add/remove item	Total weight $\leq C$	Total value (max)	All items considered / no improv.	#items, C	DP, ILP, B&B
Bin Packing	Item→bin assignment; place/open bin	Bin load $\leq$ cap.	#bins (min)	All items placed	#items, capacity	First/Best-Fit, ILP
TSP	Partial tour; insert/swap city	No repeats; valid edges	Tour length (min)	All cities visited	#cities	2/3-opt, B&B, Held–Karp
VRP	Vehicle routes; insert customer	Cap./time-window checks	Total route cost (min)	All customers routed	#cust., vehicles	Clarke–Wright, B&C, meta-heur.
Job Shop	(job,machine,time) placements	No machine overlap; precedence	Makespan (min)	All ops scheduled	#jobs, machines	ILP, CP, local search
Flow Shop	Job permutation; shared machine order	Feasible start/finish times	Makespan / flow time (min)	All jobs sequenced	#jobs, machines	Johnson (2-mach), ILP, heur.
Open Shop	Assign (job, machine, start)	No overlaps on machines	Makespan (min)	All ops assigned	#jobs, machines	CP, ILP, meta-heur.
Cutting Stock	Patterns; assign demand to patterns	Pattern length $\leq$ stock; demand track	#stocks used (min)	Demands satisfied	Demands, piece types	Column gen., ILP
Facility Location	Open facilities; assign customers	Capacity per open site	Open+service cost (min)	All customers assigned	Sites, customers	ILP, B&B, heuristics
Max Coverage	Choose up to k sets	Feasible $\lvert S\rvert \leq k$	Covered elements (max)	k used / no improv.	#sets, k	Greedy, ILP, local search
Set Cover	Choose sets covering universe	Coverage tracking	#sets or weight (min)	Universe covered	#sets, elements	Greedy, ILP, B&B
Max Flow / Min Cut	Edge flows; augment path	Capacity & conservation	s–t flow (max) / cut (min)	No augmenting path	Nodes, edges	Edmonds–Karp, Dinic
Assignment	Agent↔task matching	One-to-one constraint	Total cost (min)	n pairs formed	n	Hungarian, ILP
Quadratic Assignment	Permutation; swap facilities/locations	Valid permutation	$\sum_{i,j}$ flow·dist (min)	All n placed	n, sparsity	B&B, tabu, meta-heur.

B.2 Logic puzzle tasks

Task	State / Actions	D1: Checker (step)	D2: Evaluator (final)	D3: Stop	D4: Scale (α)	Typical Solver
Sudoku	Grid; place/remove digit	Row/col/box constraints	Valid completion (boolean)	Grid full and valid	Grid size, givens	Backtracking + CP
Kakuro	Runs; fill digits 1–9	Run sum and no repeats	All runs match sums	All cells filled, all sums met	Grid size, runs	CP, DFS/backtracking
KenKen	Grid; satisfy cages	Row/col uniqueness; cage op	All cages and rows/cols valid	Full valid grid	Grid size, cage ops	CP, backtracking
Nonogram (Picross)	Grid; shade/clear cells	Row/col run conformity	All run patterns satisfied	All rows/cols consistent	Grid size, run complexity	Line-solver, CP, SAT
Futoshiki	Grid with </>	Row/col uniqueness; inequalities	All inequalities satisfied	Full valid grid	Grid size, inequality density	CP, backtracking
Slitherlink	Edges on lattice; toggle edge	Cell numbers match incident edges; degree ≤ 2	Single loop satisfies all clues	Single simple cycle formed	Grid size, clue density	Loop logic, SAT/ILP
Hashiwokakero (Bridges)	Bridges between islands	No crossings; degree $\leq$ label; ≤ 2 parallel	All island degrees match; connected	Degrees matched and single component	#islands, labels	Graph heur., CP
Nurikabe	Shade/clear cells	No 2×2 black; island size $\leq$ label	Island sizes exact; sea connected	All labels satisfied	Grid size, label layout	CP, BFS/DFS logic
Logic Grid (Zebra)	Attribute matrix; mark yes/no	No-clash and one-of per category; apply clues	All clues satisfied; bijective assignment	All entities fully assigned	#entities, attributes	CP, SAT, tableaux
Mastermind	Color code; propose guess	Score guess with pegs; consistency with history	Min guesses to exact match	Code guessed or guess limit	Code length, colors	Knuth strategy, search
Hitori	Cells; black/white decisions	No orthogonal black adjacency; track row/col duplicates	No duplicates; white cells connected	All constraints met	Grid size, digit range	CP, DFS/backtracking
Battleships	Place fleet; mark water	Row/col ship counts; no adjacency; ship shapes	Fleet placed; counts match	All ships placed and valid	Grid size, fleet mix	CP, ILP/backtracking
Hidato	Place 1..N consecutively	Each k adjacent to $k\pm1$	Full 1..N chain	All numbers placed	Grid size, holes	Pathfinding + CP
Tents & Trees	Place tents near trees	Tent next to a tree; no tent-tent adjacency; row/col counts	Each tree has 1 tent; counts satisfied	All constraints met	Grid size, tree density	CP, logical heuristics

Abbrev.: CP = constraint programming; SAT = satisfiability; ILP = integer linear programming; DFS = depth-first search.

B.3 Simulator-based tasks

Task	State / Actions	D1: Checker (step)	D2: Evaluator (final)	D3: Stop	D4: Scale (α)	Typical Solver
Comb. circuit debug/synth.	Netlist; add/remove gate; rewire; change cell	Width/type compat.; one driver/net; no floating pins; no comb. cycles	Verilog TB sim; score=#tests passed / pass–fail	All tests pass / no improv.	#gates/#nets; depth; #vectors	Verilator / Icarus + static checks
Sequential circuit (bounded)	Netlist with FFs; edit regs/wires/modules	As left + single-clock; reset well-formed; no multiply-driven state	Simulate T cycles; compare trace vs. spec	Spec satisfied / cycle budget	#FFs+#gates; T; #vectors	Verilator (+ STA)
FSM synthesis (DFA/Mealy/Moore)	Add states/transitions; label start/accept	Determinism per symbol; well-typed I/O; reachable start	Run labeled traces; score=acc./coverage	All traces satisfied / budget	#states; alphabet; #traces×len	Automata simulator
Sorting network (bounded)	Append comparators (i, j) on n wires	Indices in range; level constraints; well-formed net	Poly test suite; score=#inputs sorted	Suite fully sorted / budget	#wires; #comparators; suite size	SN simulator / PBT harness
CA target synthesis (Life)	Toggle seed cells; optional pieces	In-bounds; edit budget	Evolve T; score=$-\mathrm{dist}(\text{target})$ / exact	Exact match / step budget	Grid size; T	CA engine
Grid-world robot plan	Append primitives (move/pick/drop)	Parseable; preconds (in-bounds; no collision)	Replay sim; reward=goal − path cost	Goal or horizon	Map size/obstacles; horizon	2-D grid sim
Compiler pass ordering (IR)	Sequence optimization passes	Pass applicability; IR verifies/compiles	Run tests; obj.=runtime/size with correct outputs	Tests pass & no further gain / pass limit	IR size; #tests; pass budget	LLVM `opt` + interpreter
Network routing (bounded DES)	Add/assign routes; flow splits	No simple cycles; link caps.	Simulate T; throughput/delay/feas.	All flows delivered or T	#nodes/#edges/#flows; T	Lightweight ns-3–style

C Additional details on the generation process

This appendix describes the instance-generation and prompt-generation pipeline used by the experiments. Each generated example is stored as a JSON object with an `id`, a `category`, a natural-language `instruction`, a structured `state`, and an `answer`. For the optimization domains, the `state` is the Markovian state used by the rollout code; the `answer` is the scalar optimum or target value when it is available from the generator. During search, the model never directly edits the full solution object. Instead, it emits a JSON action under the key `answer`; the parser extracts this action, the domain checker validates it, and the domain transition function applies it to produce the next state.

Common rollout interface All task families are implemented through the same plan-based interface. A domain specification defines:

1. a prompt function that converts the current structured state into a natural-language prompt;
2. a JSON action schema, specified by the required action keys;
3. a step validator, which implements $\texttt{chk}_t(s, a)$;
4. a state-transition rule, which applies a valid action to the current state;
5. a terminal-state predicate;
6. an objective function, which implements $\texttt{eval}_t(\cdot)$ on terminal states.

The model is asked to provide a short reasoning trace and then exactly one action. In the default experiments, the action is parsed from

`{"answer": [{...}]}`.

The parser also records whether the response was valid JSON, whether it contained the required keys, and whether the parsed action was domain-feasible.

Curriculum and levels For each task family, the generator exposes four main complexity levels, denoted $\alpha_1, \ldots, \alpha_4$. Higher levels increase the number of variables, the number of choices per step, the amount of clutter or constraints, or the horizon length. The original experiments, excluding the four-action control in Exp. ©, generate $1{,}000$ training instances per level. For their solver-prefix online RL setting, the configured dataset selects one state from each optimal solver trajectory, and orders these prompts by level, from α_1 to α_4. For offline RL, search is run from the root of each instance, the best discovered trajectory is selected using the outcome evaluator, and SFT examples are collected from the states along that trajectory.

Search-time variants For the offline ablation, the same generated instances are evaluated under several search configurations. The full configuration uses both feasibility pruning and duplicate-action merging. The no-pruning configuration permits infeasible intermediate actions to remain in the tree until terminal checking. The no-deduplication configuration keeps duplicate textual proposals even when they parse to the same structured action. The sequential baseline samples rollouts without tree reuse. The solver-reference configuration is an enumerative structural baseline: it enumerates broad next-step candidates directly from the state, rather than using LLM proposals, and is used as a reference trajectory generator in the search diagnostics.

C.1 0–1 Knapsack

Problem description The knapsack task is an add-only 0–1 knapsack problem. The state contains a capacity, item weights, item values, and the currently selected item set. At each step the model chooses one previously unselected item to add. A step is feasible if the new total weight does not exceed the capacity. The rollout terminates when no remaining item can be legally added.

Action schema The required action key is

`item_index`.

The transition appends this item to `selected` and sorts the selected set. The canonical solution key is the sorted tuple of selected item indices.

Objective and oracle The objective is the total value of the selected items, which is maximized. The generator stores an exact optimum when available. For a partial state, the oracle can compute the best completion value using the knapsack solver, while the offline search setting only uses terminal objective values discovered by rollouts.

C.2 Quadratic Assignment / CF–EU Manhattan

Problem description The QAP instances use a Manhattan grid. Facilities are assigned one-to-one to grid cells. Facilities have cluster labels, and the pairwise flow between two facilities is high when they are in the same cluster and low otherwise. The objective is

$$\sum_{i<j} \mathrm{flow}(i,j)\,\|\mathrm{loc}(i)-\mathrm{loc}(j)\|_1,$$

which is minimized.

Action schema The required action keys are

`facility`, `location`.

At each step, the model places exactly one currently unassigned facility into one free grid cell. The transition appends the pair

```
{"facility": f, "location": [r,c]}
```

to the assignment and sorts the assignment by facility id. The canonical solution key is the ordered list of facility-location pairs.

Generation process For each instance, the generator samples the number of facilities, grid size, cluster map, and a small partial assignment. The remaining unassigned facilities and unoccupied grid cells define the search space. The oracle computes the exact optimal completion by enumerating bijective assignments of the remaining facilities to free cells.

Complexity levels The QAP levels used in the experiments increase the grid size and the number of facilities:

Level	facilities n	grid size	clusters	preassigned fraction
α_1	3	4×4	2	0.25
α_2	3	5×5	2	0.25
α_3	4	5×5	2	0.25
α_4	4	6×6	2	0.25

C.3 Role Assignment with Conflicts

Problem description The role-assignment task assigns R roles to $C = R + E$ candidates. Each candidate-role pair has an integer fit score. Some candidate pairs have conflict penalties. A complete assignment must use each role exactly once and each candidate at most once. Conflicts are allowed, but their penalties are subtracted from the final score.

Action schema The required action keys are

`role`, `candidate`.

At each step, the model assigns one unfilled role to one unused candidate. The transition appends the pair

```
{"role": r, "candidate": c}
```

to the assignment. The canonical solution key is the sorted tuple of role-candidate pairs.

Objective and tie-break The raw objective is

$$\sum_{(r,c)} \mathrm{fit}(c,r) - \sum_{\{c_i,c_j\}\subseteq S} \mathrm{penalty}(c_i,c_j),$$

where S is the set of selected candidates. Ties are resolved by preferring the complete assignment with the largest minimum individual fit among its selected role-candidate pairs.

Generation process For each level, the generator samples a fit matrix, injects high-fit standouts and near-ties, samples structured conflicts, and rejects or resamples instances whose global optimum is not positive. The oracle enumerates all feasible one-to-one role-candidate assignments and selects the best assignment under the objective and tie-break.

Complexity levels

Level	roles R	extras E	candidates C	conflict density	penalty range	structure
α_1	3	1	4	0.15	[1,5]	chain
α_2	4	1	5	0.20	[2,5]	random
α_3	5	1	6	0.25	[2,6]	chain overlaps
α_4	6	1	7	0.35	[3,6]	clique-biased

C.4 MaxSat

Problem description The MaxSat task combines constrained task selection with worker assignment. The model must choose a subset of tasks and assign each selected task to a distinct eligible worker. The state contains task costs, resource budgets, eligible workers, hard Boolean clauses, and weighted soft clauses.

Action schema The required action keys are

`task_index`, `worker_index`.

At each step, the model selects one currently unselected task and assigns it to one unused eligible worker. The transition adds the task to `selected` and appends

```
{"task": t, "worker": w}
```

to the assignment list.

Feasibility A step is feasible only if:

1. the task has not already been selected;
2. the worker has not already been used;
3. the worker is eligible for the task;
4. all resource budgets remain satisfied;
5. all hard Boolean clauses remain satisfied.

The rollout terminates when no additional task-worker pair can be added without violating these conditions.

Objective and tie-break The learning objective is to maximize the total weight of satisfied soft clauses. For oracle reporting, ties are broken lexicographically: first minimize resource usage in the fixed resource order, then prefer fewer selected tasks.

Generation process The generator samples task costs, budgets, worker eligibility, hard clauses, and soft clauses. Hard clauses include implications, anti-pairs, and small at-most-one groups. Soft clauses include units, implications, anti-pairs, and at-least-one clauses with small integer weights. The oracle enumerates task subsets, checks budget and hard-clause feasibility, verifies that a matching to workers exists, and then scores the solution by the soft-clause objective and tie-breaks.

Complexity levels

Level	tasks	workers	resources	budget factor	eligibility	hard density	soft density
α_1	4	2	1	0.50	random	0.30	1.20
α_2	5	3	1	0.55	random	0.60	1.60
α_3	6	3	2	0.55	random	0.60	1.70
α_4	7	4	2	0.55	role-patterned	0.70	2.00

C.5 Single-machine weighted tardiness scheduling

Problem description The scheduling task is a single-machine sequencing problem. Each job j has processing time p_j, due date d_j, and weight w_j. The model builds a schedule one job at a time. The machine starts at time 0, runs without idle time, and jobs are non-preemptive.

Action schema The required action key is

`job_index`.

At each step, the model appends exactly one unscheduled job to the current prefix. The transition appends this job index to the order. The rollout terminates when all jobs have been scheduled.

Objective For a complete order, the objective is total weighted tardiness:

$$\min \sum_j w_j T_j, \qquad T_j = \max\{0, C_j - d_j\}.$$

The domain uses a minimization objective, while the generic search code internally converts objectives into maximization form when needed.

Generation process For each instance, the generator samples the number of jobs, processing times, due dates, and weights. Due dates are sampled using a tardiness-factor and relative-due-date-range recipe. The oracle solves the optimal suffix exactly from a partial prefix and emits a step-by-step path of next jobs.

Complexity levels

Level	jobs n	processing range	TF	RDD	max weight
α_1	5	[1,7]	0.30	0.60	3
α_2	6	[1,7]	0.50	0.60	5
α_3	7	[2,8]	0.70	0.60	10
α_4	7	[1,15]	0.60	0.60	6

C.6 Polyomino Target Cover

Problem description The Polyomino Target Cover task is a spatial optimization problem. The board contains target cells `t`, optional obstacles `#`, and possibly pre-placed example pieces. The model has a pool of labeled pieces and a move budget K. At each step, it chooses an unused piece, rotates it by $0°$, $90°$, $180°$, or $270°$, and places it at a top-left anchor. The goal is to maximize the number of newly covered targets within the move budget.

Action schema The action contains:

`piece_id`, `anchor`, `rotation`, `grid_after`.

The `anchor` is the global row-column coordinate of the top-left cell of the transformed piece's tight bounding box. The `grid_after` field is included so that the parser and checker can compare the declared placement with the resulting board.

Feasibility A placement is feasible only if all occupied cells lie inside the board and do not overlap obstacles or existing letters. A piece may be used at most once. Reflections are not used in this task.

Generation process The generator first samples a board size, target clusters, singleton target cells, and obstacles. It then samples a piece pool from a level-dependent piece library. Some levels include pre-placed example pieces, which do not count against the move budget. The oracle uses an exact bitboard branch-and-bound search over legal placements and returns the optimal additional target coverage and a decorated path of placements.

Complexity levels

Level	grid	K	pool size	piece set	examples	obstacles
α_1	4×4	1	3	domino-only	2	0
α_2	5×5	1	3	domino/square	2	0–1
α_3	5×5	2	4	full	2	0–1
α_4	6×6	3	5	full	1	0–1

C.7 Auxiliary Grid-Spatial tasks

Purpose The Grid-Spatial tasks are auxiliary held-out spatial QA tasks used to test whether training on the Polyomino Target Cover optimization task transfers to more elementary spatial reasoning skills. Unlike Polyomino Target Cover, these tasks are *not* multi-step optimization problems. They are mostly one-step questions about moving, rotating, reflecting, comparing, or placing small shapes on a grid. We use them as diagnostic probes for the spatial abilities that are implicitly required by the polyomino optimization task.

General format Each instance contains a small rectangular canvas represented as a Python-style list of lists. Cells may be empty, blocked, or marked as targets:

$$\texttt{"."} = \text{empty}, \qquad \texttt{"\#"} = \text{obstacle}, \qquad \texttt{"t"} = \text{target}.$$

Some instances also contain pre-placed uppercase letters, which act as occupied cells. The shape is given as a small list-of-lists using a letter such as `"A"` for occupied cells and `"."` for empty cells inside the shape's bounding box. Coordinates are always row-column coordinates, with rows and columns indexed from 0. An anchor $[r, c]$ denotes the top-left position of the transformed shape's tight bounding box after rotation or reflection.

A typical input therefore looks like:

```
canvas = [
  [".", ".", ".", ".", "."],
  [".", "#", ".", ".", "."],
  [".", ".", ".", "t", "."],
  [".", ".", ".", ".", "."],
  [".", ".", ".", ".", "."]
]

shape = [
  ["A", "."],
  ["A", "A"]
]

anchor = [2, 1]
operation = rotate 90 degrees clockwise
```

The model must reason about the transformed coordinates of the occupied cells, check whether the placement is legal, and return the requested answer in JSON form.

Tasks selected for testing The held-out Spatial QA evaluation is organized into the following capability categories. These are the categories reported in the spatial capability tables.

Capability	What the test instance asks
Translate	Move a given shape from one anchor to another without changing its orientation. The model must compute the final occupied cells and usually return the resulting grid or final anchor.
Rotate & Place	Rotate a shape by a specified angle in $\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$ and place it at a specified anchor. The model must update the shape coordinates after rotation and check whether the placement fits on the canvas.
Reflect & Place	Reflect a shape horizontally, vertically, or diagonally, then place it at a specified anchor. This tests whether the model understands mirror transformations on a discrete grid.
Compose	Apply a short composition of transformations, such as rotation followed by reflection followed by translation. These examples test whether the model can keep track of multiple spatial operations in order.
Equivalence	Given two shapes, decide whether they are the same up to rotation and, when allowed, reflection. If they are equivalent, the model returns a transformation explaining how one shape maps to the other.
Symmetry	Given one shape, report which symmetries it has, such as 90° rotation symmetry, 180° rotation symmetry, horizontal mirror symmetry, vertical mirror symmetry, or diagonal mirror symmetry.
Overlap / IoU	Place two transformed shapes on the same empty canvas and compute their intersection cells, intersection count, and union count. This is a small discrete analogue of intersection-over-union reasoning.
Coverage	Given a target cell and a transformed shape, list the anchors for which the shape would cover that target cell. This tests inverse spatial reasoning: instead of asking where a shape lands after an anchor is chosen, the task asks which anchors could have produced coverage of a specific cell.

Examples of what the selected tasks look like Below are simplified examples illustrating the style of the held-out Spatial QA tasks.

Translate. The prompt gives a shape and an initial/final anchor. The model must shift all occupied cells by the anchor displacement.

```
shape = [
  ["A", "."],
  ["A", "A"]
]
old_anchor = [0, 1]
new_anchor = [2, 3]
```

The expected reasoning is that the occupied local cells of the shape are

$$(0,0),\ (1,0),\ (1,1).$$

At anchor $[2,3]$, these become global cells

$$(2,3),\ (3,3),\ (3,4).$$

Rotate & Place. The prompt gives a shape, a rotation angle, and an anchor.

```
shape = [
  ["A", "."],
  ["A", "A"]
]
rotation = 90
anchor = [1, 2]
```

The model must rotate the shape clockwise, normalize the rotated shape to its tight bounding box, then place that bounding box with top-left cell at $[1,2]$. It must also check that no occupied cell lands outside the board or on an obstacle.

Reflect & Place. The prompt gives a reflection mode such as horizontal or vertical.

```
shape = [
  ["A", "A", "."],
  [".", "A", "A"]
]
reflection = vertical
anchor = [0, 1]
```

The model must mirror the shape across the requested axis and then place the reflected shape at the anchor.

Compose. A compose instance asks the model to apply several transformations in sequence:

```
shape = [
  ["A", "."],
  ["A", "A"]
]
operations = rotate 90, then reflect horizontal, then place at [2, 1]
```

These examples are harder because an error in the first operation changes all later coordinates.

Equivalence. An equivalence instance gives two shapes:

```
shape_1 = [
  ["A", "."],
  ["A", "A"]
]

shape_2 = [
  ["B", "B"],
  ["B", "."]
]
```

The model must decide whether `shape_2` can be obtained from `shape_1` by rotation, and possibly reflection. The answer may specify a rotation angle and whether reflection is needed.

Symmetry. A symmetry instance gives one shape:

```
shape = [
  ["A", "A"],
  ["A", "A"]
]
```

The model must report the transformations that leave the shape unchanged. For the square above, several rotations and mirror symmetries are valid.

Overlap / IoU. An overlap instance gives two shapes with their own transformations and anchors:

```
shape_1 anchor = [1, 1], rotation = 0
shape_2 anchor = [1, 2], rotation = 90
```

The model must compute the cells occupied by each placed shape, then return the intersection count and union count. This probes whether the model can compare two transformed coordinate sets.

Coverage. A coverage instance gives a target cell and asks which anchors would make a shape cover that cell:

```
target_cell = [2, 3]
shape = [
  ["A", "."],
  ["A", "A"]
]
rotation = 0
```

The model must reason backward from the target cell to possible anchors. For every occupied local cell (u, v) in the shape, an anchor candidate is

$$[r, c] = [2 - u,\ 3 - v].$$

The legal anchors are those whose resulting placement stays inside the grid and avoids obstacles.

Difficulty levels used for testing The Spatial QA tests use two main held-out difficulty settings. Both use the same task types above, but the second setting increases the grid size, the shape size, and the amount of clutter.

Difficulty	Canvas	Shape sizes	Obstacles	Decoy letters	Transform variety
Diff. 1	about 5×5	mostly 2–3 cells	0–1	0	simpler rotations/reflections
Diff. 2	about 6×6	mostly 3–4 cells	1–2	0–1	full rotations and more reflections

Why these tasks are useful These tasks are designed to isolate the low-level spatial operations needed for the optimization task. Polyomino Target Cover requires the model to choose pieces, rotate them, place them legally, avoid collisions, and reason about target coverage. The auxiliary Grid-Spatial tests separate these abilities into simpler questions. Thus, improved performance on these tests suggests that optimization training is not only teaching the model to imitate a specific polyomino solver, but also improving reusable spatial operations such as coordinate tracking, rotation, reflection, overlap computation, and coverage reasoning.

D Prompts (examples)

This section gives representative prompts produced by the task promptization functions. The examples show the user-facing prompt, not the hidden checker. In every optimization prompt below, the model is expected to give brief reasoning and then return exactly one JSON action under `answer`. The action keys shown here match the keys validated by the domain specifications.

0–1 Knapsack (example)

You are solving a **0–1 knapsack** problem **incrementally**.
Objective
Build a high-value feasible set of items. You may only add items. Stop only when no remaining item can be legally added without exceeding capacity.
State

```
Capacity = 45

Weights = [
  {"0": 4}, {"1": 18}, {"2": 1}, {"3": 8},
  {"4": 12}, {"5": 22}, {"6": 6}, {"7": 22},
  {"8": 17}, {"9": 19}, {"10": 4}, {"11": 19},
  {"12": 19}, {"13": 16}, {"14": 18}, {"15": 3}
]

Values = [
  {"0": 1}, {"1": 15}, {"2": 1}, {"3": 10},
  {"4": 10}, {"5": 8}, {"6": 5}, {"7": 37},
  {"8": 25}, {"9": 27}, {"10": 5}, {"11": 17},
  {"12": 21}, {"13": 6}, {"14": 15}, {"15": 1}
]

Currently selected item indices = []
Current total weight = 0
Current total value  = 0
```

Rules

- Add exactly one item at this step.
- Do not add an item that is already selected.
- The new total weight must be at most the capacity.

Briefly explain the trade-off you are using, such as remaining capacity, value, and value/weight ratio. Then propose the next item to add.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose **1** action with the following format:

```
{"answer":
[
  {
    "item_index": <int>
  }
]
}
```

CF–EU Manhattan / QAP (example)

You are solving an **incremental facility placement** problem.
Grid

```
[
```

```
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", "1", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."]
]
```

Legend
"." means a free cell. A number means that the corresponding facility is already assigned to that cell.
Assigned facilities

```
[
  {"facility": 1, "location": [2, 1]}
]
```

Unassigned facilities

```
[0, 2, 3]
```

Cluster map

```
{"0": 0, "1": 1, "2": 1, "3": 1}
```

Flow rule
Facilities in the same cluster have flow $H = 10$; facilities in different clusters have flow $L = 1$.
Distance metric
Manhattan distance.
Objective
Complete the assignment while minimizing

$$\sum_{i<j} \text{flow}(i, j) \times \text{Manhattan}(\text{location}(i), \text{location}(j)).$$

Rules

- Add exactly one unassigned facility.
- Place it in a currently free grid cell.
- A grid cell may contain at most one facility.

Briefly explain the placement reasoning, especially cluster and flow effects. Then propose the next placement.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose **1** action with the following format:

```
{"answer":
[
  {
    "facility": <int>,
    "location": [<row>, <col>]
  }
]
}
```

Role Assignment with Conflicts (example)

You are solving a **role assignment with conflicts** problem **incrementally**.
Objective

1. Assign exactly one candidate to each role.
2. Each candidate can be used at most once.

3. Maximize

$$\Big(\sum \text{chosen fit scores}\Big) - \Big(\sum \text{conflict penalties among selected candidates}\Big).$$

4. Tie-breaker: among equal total scores, prefer the assignment with the highest minimum individual fit.

Current state

- Number of roles: `6`
- Number of candidates: `7`
- Already assigned role-candidate pairs: `[]`

Fit matrix
Rows are candidates and columns are roles.

```
candidate 0: role scores {0: 9, 1: 9, 2: 1, 3: 3, 4: 8, 5: 3}
candidate 1: role scores {0: 6, 1: 4, 2: 3, 3: 1, 4: 8, 5: 1}
candidate 2: role scores {0: 1, 1: 0, 2: 1, 3: 7, 4: 9, 5: 5}
candidate 3: role scores {0: 3, 1: 8, 2: 5, 3: 3, 4: 1, 5: 3}
candidate 4: role scores {0: 6, 1: 2, 2: 6, 3: 8, 4: 1, 5: 0}
candidate 5: role scores {0: 8, 1: 7, 2: 4, 3: 1, 4: 8, 5: 3}
candidate 6: role scores {0: 4, 1: 6, 2: 8, 3: 9, 4: 1, 5: 1}
```

Conflict penalties

```
(0,6): 3
(0,5): 4
(0,2): 6
(0,1): 5
(4,5): 4
(5,6): 3
(1,5): 5
```

Rules

- Add exactly one new role-candidate assignment.
- The role must be currently unfilled.
- The candidate must be currently unused.
- Conflicts are allowed, but their penalties are subtracted in the objective.

Briefly explain the fit-vs-conflict trade-off and the tie-breaker. Then propose the next assignment.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose **1** action with the following format:

```
{"answer":
[
  {
    "role": <int>,
    "candidate": <int>
  }
]
}
```

Wished Assignments / Constrained MaxSAT (example)

You are solving a **Wished Assignments** puzzle **incrementally**. This is a constrained task-selection and worker-assignment problem.
Objective

1. Maximize the total weight of satisfied soft clauses, subject to budgets, worker eligibility, and hard logic.
2. Tie-breaker A: among equal soft-clause scores, minimize resource usage lexicographically.
3. Tie-breaker B: among remaining ties, prefer fewer selected tasks.

Budgets

$$\text{Money} = 12, \qquad \text{Time} = 11.$$

Workers

```
worker_index 0: W1
worker_index 1: W2
worker_index 2: W3
worker_index 3: W4
```

Tasks

```
task_index 0, A: costs {Money: 2, Time: 1}, eligible workers [2]
task_index 1, B: costs {Money: 2, Time: 4}, eligible workers [1]
task_index 2, C: costs {Money: 4, Time: 3}, eligible workers [0]
task_index 3, D: costs {Money: 5, Time: 4}, eligible workers [1, 3]
task_index 4, E: costs {Money: 1, Time: 2}, eligible workers [0, 2]
task_index 5, F: costs {Money: 5, Time: 3}, eligible workers [1, 3]
task_index 6, G: costs {Money: 2, Time: 3}, eligible workers [0, 2]
```

Hard clauses

Here $\neg$ means negation.

- $\neg A \vee \neg C$
- $\neg A \vee G$
- $\neg C \vee \neg D$
- $\neg C \vee \neg F$
- $\neg D \vee \neg F$

Soft clauses

1. $D \vee \neg F$ $(w = 1)$
2. $C \vee G$ $(w = 1)$
3. $\neg B$ $(w = 1)$
4. $\neg D \vee \neg F$ $(w = 1)$
5. $\neg C \vee F$ $(w = 1)$
6. $\neg A \vee \neg E$ $(w = 2)$
7. $B \vee C$ $(w = 2)$
8. $\neg B \vee C$ $(w = 1)$
9. $\neg B \vee G$ $(w = 2)$
10. $\neg D \vee \neg F$ $(w = 1)$
11. $\neg D \vee E$ $(w = 1)$
12. $G \vee C$ $(w = 2)$
13. $\neg C \vee B$ $(w = 2)$
14. B $(w = 2)$

Current state

```
Selected tasks = []
Assigned task-worker pairs = []
```

Rules

- Add exactly one task and assign it to exactly one worker.
- The task must be currently unselected.
- The worker must be currently unused.

- The worker must be eligible for the task.
- The new selected set must satisfy all hard clauses.
- The new resource usage must not exceed the budgets.

Briefly explain budget feasibility, hard-clause feasibility, worker eligibility, and the soft-clause effect. Then propose the next task-worker assignment.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose **1** action with the following format:

```
{"answer":
[
  {
    "task_index": <int>,
    "worker_index": <int>
  }
]
}
```

Single-machine Weighted Tardiness Scheduling (example)

You are solving a **single-machine weighted tardiness scheduling** problem **incrementally**.
Objective

$$\min \sum_j w_j \cdot \max(0, C_j - d_j).$$

Rules

- The machine starts at time 0.
- There is no inserted idle time.
- Jobs are non-preemptive.
- At this step, append exactly one unscheduled job to the current prefix.

Jobs

Id	job_index	p	d	w
A	0	3	9	4
B	1	8	29	4
C	2	11	22	5
D	3	6	14	1
E	4	15	28	2
F	5	7	30	2
G	6	7	19	5

Current state

```
Current partial order = []
Current partial order indices = []
Time elapsed so far = 0
Remaining job indices = [0, 1, 2, 3, 4, 5, 6]
```

Briefly explain your scheduling rationale, such as due dates, processing times, weights, and whether a job risks becoming tardy. Then choose the next job to append.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose **1** action with the following format:

```
{"answer":
[
  {
    "job_index": <int>
  }
]
}
```

Polyomino Target Cover (example)

You are playing a **Polyomino Target Cover** puzzle **incrementally**.
Objective
Maximize the number of target cells covered by placed pieces within the move budget.
Constraints

- Choose an unused piece.
- Rotate it by $0°$, $90°$, $180°$, or $270°$ clockwise.
- Place the transformed shape using the top-left anchor of its tight bounding box.
- All occupied cells must lie inside the board.
- Occupied cells may not overlap `"#"` or existing piece letters.

Budget

$$K = 3 \quad \text{moves total; current round } 0.$$

Board before example placement

```
initial_grid = [
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."]
]
```

Target locations

```
[(0,0), (0,1), (1,0), (1,1), (0,2), (1,2), (2,2),
 (0,3), (1,3), (1,4), (2,3)]
```

Current board after example piece placement

```
current_grid = [
  [".", ".", ".", ".", "A", "A"],
  [".", ".", ".", ".", "A", "A"],
  [".", ".", ".", ".", "A", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."],
  [".", ".", ".", ".", ".", "."]
]
```

Piece pool

```
A (P5), already placed at anchor [0,4], rotation 0:
[
  ["A", "A"],
  ["A", "A"],
  ["A", "."]
]

B (L4), unused:
[
```

```
  ["B", "."],
  ["B", "."],
  ["B", "B"]
]

C (Z4), unused:
[
  ["C", "C", "."],
  [".", "C", "C"]
]

D (O), unused:
[
  ["D", "D"],
  ["D", "D"]
]

E (R6), unused:
[
  ["E", "E", "E"],
  ["E", "E", "E"]
]
```

Briefly explain the placement reasoning, including feasibility and newly covered targets. Then return exactly one placement.
Immediately after reasoning within <think> your reasoning here </think>, propose **1** action with the following format:

```
{"answer":
[
  {
    "piece_id": "<ID>",
    "anchor": [<row>, <col>],
    "rotation": <0|90|180|270>,
    "grid_after": [
      [".", ".", "..."],
      ["...", "...", "..."]
    ]
  }
]
}
```

Auxiliary Grid-Spatial QA (example)

You are solving a **single-step grid-spatial reasoning** task.
Task name
`rotate_place`
Canvas

```
[
  [".", ".", ".", ".", "."],
  [".", "#", ".", ".", "."],
  [".", ".", ".", "t", "."],
  [".", ".", ".", ".", "."],
  [".", ".", ".", ".", "."]
]
```

Shape

```
[
```

```
  ["A", "."],
  ["A", "A"]
]
```

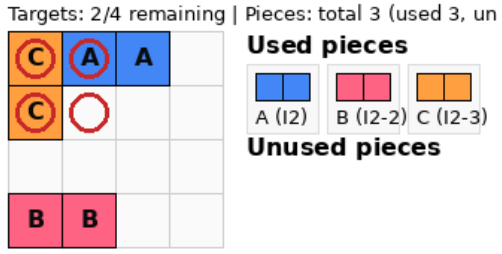
Operation
Rotate the shape by 90° clockwise and place it with global anchor `[2, 1]`.
Rules

- The anchor is the top-left coordinate of the transformed shape's tight bounding box.
- The placement must stay inside the canvas.
- The placement must avoid `"#"` and existing letters.

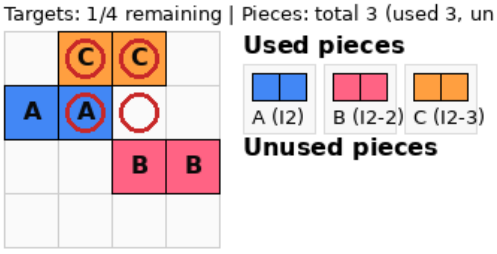
Return the transformed placement and the resulting grid.
Immediately after reasoning within `<think>` your reasoning here `</think>`, propose the answer with the following format:

```
{"answer":
[
  {
    "anchor": [<row>, <col>],
    "rotation": 90,
    "legal": <true|false>,
    "solution_grid": [
      [".", ".", "..."],
      ["...", "...", "..."]
    ]
  }
]
}
```

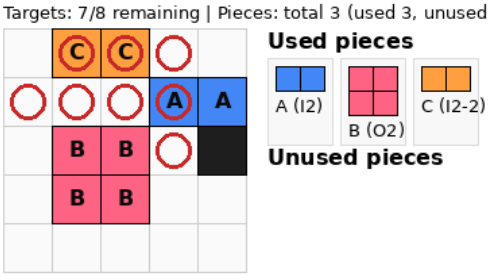
E Polyomino examples



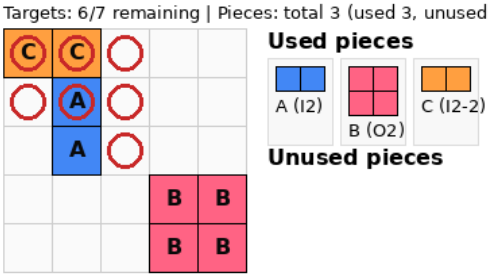
(a) Level 1



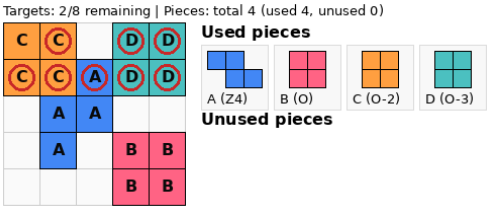
(b) Level 1



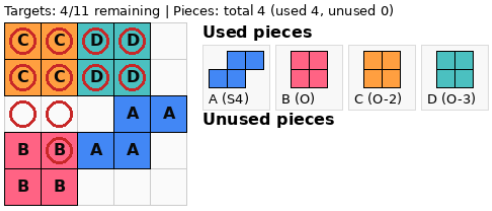
(c) Level 2



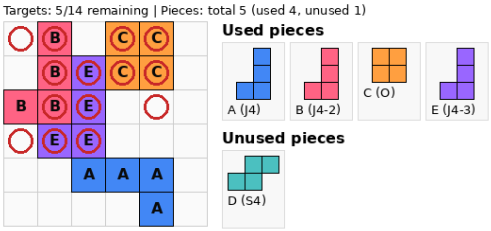
(d) Level 2



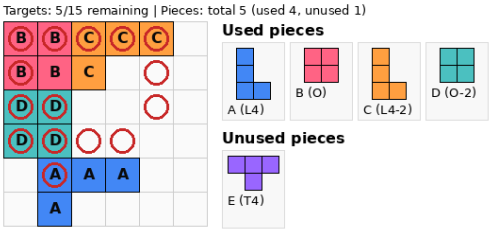
(e) Level 3



(f) Level 3



(g) Level 4



(h) Level 4

Figure 5: Polyomino target cover examples. This shows the optimal solution found by the solver. The first three levels have two randomly positioned pieces that serve as examples. The last level has only one example.

F Additional Details

F.1 Search Ablations

Task representation. We evaluate search methods for structured combinatorial optimization problems. The experiments use four task families: `qap`, `knapsack`, MaxSat, and `role_assignment`. For each family, we evaluate difficulty levels 1–4. Each instance is treated as a partial construction problem: a node in the search tree stores the current partial state, and a child node corresponds to one proposed next action. The language model proposes actions in a structured format, which is then parsed into a domain state. The domain checker verifies whether the output is valid JSON, whether it is structurally valid for the task, whether the partial path is feasible, and whether the node is a terminal answer. Terminal feasible answers are scored by their normalized objective value, with all objectives converted to a maximization convention.

Expansion and filtering. When a model-based search node is expanded, the language model proposes 16 candidate next actions. Each candidate receives an edge log-probability from the model. We use the average token log-probability both as an edge score and, after exponentiation, as the prior P_i. Invalid parses are discarded. Depending on the strategy, domain-infeasible partial actions may either be pruned immediately or allowed to remain in the tree until terminal checking. When redundancy removal is enabled, children with the same canonical solution/state key are merged so that duplicate proposals do not occupy multiple search slots. Terminal feasibility is always enforced in these ablations: infeasible final answers receive a negative reward rather than being counted as successful solutions.

Standard MCTS. The main method is Monte Carlo Tree Search (MCTS). Each MCTS rollout starts at the root, repeatedly selects a child, expands an unexpanded node, and backpropagates the terminal reward. Child selection uses a PUCT-style rule:

$$\mathrm{score}(i) = Q_i^{\mathrm{blend}} + c_{\mathrm{puct}} P_i \frac{\sqrt{N_{\mathrm{parent}}}}{1 + N_i + N_i^{\mathrm{bad}}},$$

where Q_i^{blend} is the value used for selection, P_i is the model prior, N_i is the child visit count, and N_i^{bad} counts visits through incorrect or infeasible partial paths. The exploration coefficient is $c_{\mathrm{puct}} = 5$. Feasible terminal solutions contribute objective-based rewards to the path. Incorrect or invalid final states receive reward -1. Depth-limited non-answer rollouts receive reward 0 and are down-weighted in the value estimate by the depth-discount factor 0.25.

Parent-initial-value MCTS. The parent-initial-value variant uses the same search procedure as the standard MCTS ablation: it still performs rollouts, uses the PUCT selection formula, maintains visit counts, and backpropagates rewards. The difference is only in the value term used by PUCT before enough rewards have been observed. After a node is expanded, each child receives an initial value I_i from the model score of that child. The default source is the average token log-probability. The parent also stores the average initial value of its children. The value term becomes

$$Q_i^{\mathrm{blend}} = \lambda Q_i + (1 - \lambda) I_i.$$

In the standard MCTS ablation, $\lambda = 1.0$, so the initial value is ignored and PUCT uses the observed MCTS value Q_i. In the parent-initial-value ablation, $\lambda = 0.5$, so the selection value is an equal blend of the observed MCTS value and the model-derived initial value. The scale parameter is $\mu = 1.0$, meaning the stored initial value is used directly. This variant is therefore best understood as MCTS with a model-score warm start, not as beam search.

Beam search ablation. Beam search is different from the parent-initial-value MCTS run. Instead of running PUCT rollouts and backpropagating rewards, beam search proceeds layer by layer. At each depth, it expands every node currently in the beam, collects all non-terminal children, ranks them by cumulative path log-probability, and keeps only the top 4 nodes for the next depth. The beam score is simply the path log-probability, i.e., the sum of edge average log-probabilities along the partial solution. The beam run still uses the same parsing, feasibility-pruning, redundancy-removal, and terminal-feasibility options as the corresponding strategy preset, but it does not use PUCT visit counts, PUCT backpropagation, or the parent-initial-value blend. Unlike MCTS, the beam computation is controlled by maximum depth and beam width rather than by a rollout budget.

Table 5: Difference between the standard MCTS run, the parent-initial-value MCTS run, and the beam-search ablation.

Ablation setting	Search mode	Main behavior
Standard ablation	MCTS, sequential baseline, solver reference	Uses the full strategy set. `S1`–`S3` are MCTS variants, `S4` is a sequential non-MCTS baseline, and `S5` is the solver-reference run. MCTS uses PUCT and reward backpropagation.
Parent-initial-value ablation	MCTS with parent initial values	Uses `S1`–`S3` and `S5`. The search is still MCTS. The only algorithmic change is that PUCT uses a $0.5/0.5$ blend of observed value and model-derived initial value.
Beam-search ablation	Beam search plus solver reference	Uses `S1`–`S3` and `S5`. Search is layer-based: expand the current beam, rank by cumulative log-probability, and keep the top 4 partial solutions. It does not use MCTS visit counts or reward backpropagation for selection.

Table 6: Strategy presets used in the ablations.

Strategy	Configuration
`S1`	Full search variant: expansion-time feasibility pruning, redundancy removal, and terminal feasibility enforcement.
`S2`	No expansion-time feasibility pruning; infeasible partial paths may continue until terminal checking; redundancy removal remains enabled.
`S3`	No expansion-time feasibility pruning and no redundancy removal; infeasible partial paths may continue until terminal checking.
`S4`	Sequential non-MCTS baseline used only in the standard ablation. It samples one path at a time without PUCT tree selection.
`S5`	Solver-reference search with a depth-first frontier. This run enumerates solver-generated actions rather than sampling language-model actions, and is used mainly to provide reference solutions for evaluation.

Strategy presets. The strategy names define the ablation axis. The same names are reused across MCTS and beam search so that the effect of pruning and duplicate removal can be compared under different search modes.

Default hyperparameters. The default experimental settings are summarized below. Unless otherwise stated, these values are shared across the ablations.

Table 7: Default hyperparameters used by the ablation experiments.

Hyperparameter	Default value
Base language model	`meta-llama/Llama-3.2-3B-Instruct`
Problem families	`qap`, `knapsack`, MaxSat, `role_assignment`
Difficulty levels	1–4
Maximum search depth	6 for all difficulty levels
MCTS rollout budget	16 rollouts per instance and strategy
Solver-reference budget	500 solver-reference rollouts per instance
Children per expansion	20
PUCT coefficient c_{puct}	5 for MCTS variants
Beam width	4 for the beam-search ablation
Generation settings	temperature 0.7, top-$p = 0.95$, maximum 768 new tokens
Repeats and seeds	one repeat with seed list $\{0\}$
Terminal handling	terminal feasibility enforced; incorrect or invalid final rewards set to -1
Depth-limited rollouts	non-answer depth-limit reward 0, with depth-based value discount 0.25
Parallelism	GPUs 0–3 with 4 workers per GPU; solver reference uses 4 CPU workers

Reporting and smartness metrics. All smartness quantities are computed as post-processing from the saved ablation metrics; this step does not rerun search. The same reporting procedure is applied to the standard MCTS ablation, the parent-initial-value MCTS ablation, and the beam-search ablation.

The report forms reference pools of unique terminal solutions. `solverref` contains solutions found by `S5`, while `unionref` contains all unique terminal solutions found by any strategy. The primary report setting uses `unionref`, relative-gap threshold $\epsilon = 0.05$, and $\delta = 0.10$, so the target success probability is $1 - \delta = 0.90$. A solution is counted as ϵ-good if it is feasible and within 5% relative gap of the best-known normalized objective for that instance.

For each problem and difficulty level, the report estimates an effective difficulty $D_{\alpha,\epsilon}$ from the fraction of reference-pool solutions that are ϵ-good. It then estimates τ, the rollout budget or wall-clock time required for a strategy to reach the target success probability. The main smartness score is

$$\text{smartness} = \frac{D_{\alpha,\epsilon}}{\tau},$$

so larger values mean that a strategy makes more progress per rollout or per second. The enhanced report also exports pass@$\{8, 16, 32, 64\}$, an effective-branching proxy, invalid proposal rates, duplicate proposal rates, feasible-terminal rates, and exact-optimal rates.

F.2 Policy Training Implementation for RFT

Training objective. Policy training is performed as supervised fine-tuning of the base causal language model. The goal is to teach the model to produce the next search action from a partial problem state. Each training example contains a system prompt, a user prompt describing the current partial state, and an assistant target containing the next action and its accompanying reasoning. The model is trained with the standard autoregressive next-token objective on the chat-formatted sequence. Padding tokens are masked from the loss.

Constructing supervision from search. Training data are generated from successful search trajectories. For each problem instance, search first builds a tree of partial solutions. Terminal nodes are filtered to keep valid terminal answers, and the best terminal node is selected according to the task objective. The path from the root to this terminal node is then decomposed into one supervised example per edge. If a trajectory contains d actions, it contributes d training examples. For an edge $s_t \to s_{t+1}$, the input prompt is built from the partial state s_t, and the target is the exact action text stored at s_{t+1}. This turns a complete discovered solution into step-level imitation data.

Only terminal answer trajectories are used for the default next-step policy data. If a search run does not find a valid terminal solution for an instance, that instance does not contribute a supervised trajectory. When multiple terminal solutions are available, the selected trajectory is the one with the best objective value, with ties broken randomly. The saved examples also contain metadata such as node depth, visit count, and value estimates, but the policy fine-tuning objective uses only the system prompt, user prompt, and next-step target text.

Training variants. We use the same fine-tuning procedure for the MCTS-generated and solver-style datasets. In the MCTS-generated setting, the demonstrations come from the best terminal paths found by the MCTS search policy. In the solver-style setting, the demonstrations come from a more direct solver-guided trajectory generator. A no-training baseline is also evaluated by running the base model directly under the same evaluation procedure. Thus, the training comparison isolates whether policy fine-tuning on search-generated trajectories improves subsequent search and solution quality.

Tokenization and formatting. Examples are formatted with the chat template of the base model tokenizer. The formatted conversation contains a system message, a user message, and an assistant message. The assistant message is filled with the selected next-step target during training. Sequences are padded or truncated to a maximum length of 2048 tokens. The tokenizer uses the end-of-sequence token as the padding token when no dedicated padding token is available.

Optimization. Fine-tuning is implemented with the Hugging Face causal language-model training stack. The default runs fine-tune the base model directly rather than using parameter-efficient adapters. Mixed precision is selected automatically: bfloat16 is used when supported by the GPU, otherwise

Table 8: Default supervised fine-tuning settings for the policy model.

Training setting	Default value
Base model	`meta-llama/Llama-3.2-3B-Instruct`
Training objective	causal language-model supervised fine-tuning
Training examples	one next-step target per edge on a selected successful trajectory
Trajectory selection	best valid terminal solution found by search
Learning rate	5×10^{-6}
Epochs	3
Per-device batch size	4
Gradient accumulation	16 steps
Maximum sequence length	2048 tokens
Optimizer	AdamW
Learning-rate schedule	linear schedule with 0.05 warmup ratio
Weight decay	0.01
Precision	bfloat16 when available; otherwise float16 on CUDA
Checkpointing	final checkpoint only
Evaluation during training	once per epoch when a validation set is provided

float16 is used on CUDA. The optimizer is AdamW with a linear learning-rate schedule, warmup ratio 0.05, and weight decay 0.01. Intermediate checkpoints are not saved; the final model and tokenizer are saved after training.

Evaluation protocol. When validation data are provided, the model is evaluated before fine-tuning, at the end of each epoch, and once more after training. After fine-tuning, the resulting policy is used as the proposal model in the same search procedure used for data generation and evaluation. This keeps the comparison focused on the effect of the learned proposal policy rather than changes in the downstream search algorithm.

Optional adapter path. The training code also supports an optional low-rank adapter mode. When enabled, LoRA adapters are attached to the query and value projection matrices with rank $r = 8$, scaling $\alpha = 32$, dropout 0.05, and no bias terms. This path is useful for memory-constrained runs, but the default training configuration uses direct fine-tuning of the base causal language model.

F.3 Policy training GRPO

Legacy ranking and the revised protocol. The legacy solver-prefix implementation maps distinct valid completion scores to equally spaced values in $[0, 1]$, with tied scores sharing a value and invalid actions receiving -1. These values are used directly as advantages, without group standardization; an all-valid, all-tied group receives $+1$. The revised protocol replaces this mapping with the centered pairwise advantages of Algorithm 1. Valid results outrank invalid ones, invalid results tie, and all-tied or singleton groups receive zero. The implementation compares exact score ties by default; nonzero absolute or relative tie tolerances must be recorded if enabled. Advantages are applied to assistant response tokens without a second normalization.

The configured online dataset selects one solver-trajectory state per instance, deterministically from the instance ID and data seed. Pairwise reruns retain this state selection, difficulty order, and solver-prefix evaluation. Table 4 now reports the completed pairwise transfer rerun. Spatial/curriculum tables retain the original results; Appendix G.2 reports the updated GRPO math-transfer comparison for both 3B models. Changing the rank estimator does not make these experiments autonomous. Table 9 records the rank-enabled legacy solver-prefix configuration, separately from the transfer rerun below, the math-transfer recipe in Appendix G.2, and the autonomous settings in Appendix I.

Pairwise transfer rerun (Table 4). The new Role Assignment pipeline requests 1,000 roots per level and selects one solver-prefix state per retained root, omitting empty solver trajectories without replacement. Training processes levels α_1–α_4 in order for one epoch, with 128 prompts per batch, eight responses per prompt, learning rate 10^{-6}, actor KL coefficient 0.001, and clipping bounds $0.2/0.28$. It uses centered pairwise advantages, zero tie tolerances, and no group filtering or refill. The final incomplete batch is dropped. The rebuilt MATH-500 and MBPP baselines use ordinary group-standardized GRPO for two and four epochs, respectively, with mathematical-answer verification and code-test rewards. These recipes do not equate training-data or update budgets. The rerun uses two H100 NVL GPUs (approximately 94 GB each). The pipeline records retained counts and runtime overrides in its run manifests.

Evaluation freezes the same twelve domain-by-level files for all four checkpoints. The generator requests 250 roots per stratum and retains all available solver-prefix states. Each prompt receives eight responses at temperature 1, top-$p = 0.7$, top-$k = 50$, and prompt/response limits of 1,256/2,048 tokens, using the same sampling seed. The retained scorer checks validity and compares the solver-completed scalar objective with the reference value. The default legacy path permits variable action counts and zero scores on solver exceptions; synthetic tests expose repeated-job Scheduling and zero-target failure matches. Thus, the reported score is not a strict single-action optimality certificate. Their occurrence in the saved evaluations has not been audited.

For prompt i, let q_i be the fraction of its eight responses marked valid and matching. The point estimate is $\hat{p} = N^{-1} \sum_i q_i$. Let G be the number of original puzzles, m_g their prefix counts, and $S_g = \sum_{i \in g} q_i$. The displayed standard error is

$$\mathrm{SE} = \frac{1}{N} \sqrt{\frac{G}{G-1} \sum_{g=1}^{G} (S_g - \hat{p}\, m_g)^2}.$$

Means and SEs are multiplied by 100. This preserves equal prompt weighting while clustering prefixes from the same puzzle. These are evaluation SEs conditional on each checkpoint, not 95% confidence margins or independent-training-run variability.

Table 9: Rank-enabled legacy solver-prefix training configuration. The Table 4 rerun uses the separate transfer pipeline described above.

Training setting	Value
Base model	`Qwen/Qwen2.5-3B-Instruct` by default; task wrappers also support `Qwen/Qwen2.5-7B-Instruct`
Training algorithm	GRPO-style clipped policy updates in VERL with the legacy rank mapping
Validation file	level-1 test Parquet during training; full level 1–4 evaluation after merging
Train batch size	128 prompts
Rollouts per prompt	8 responses
Maximum prompt length	1256 tokens for Role Assignment, MaxSat, and Polyomino Target Cover
Maximum response length	2048 tokens
Epochs	1 for the main task runs
Data order	`data.shuffle=false`
Actor learning rate	1×10^{-6}
Optimizer backend	VERL actor optimizer with FSDP training
PPO mini-batch size	128
PPO micro-batch size	4 per GPU
KL loss	enabled, coefficient 0.001, `low_var_kl`
KL controller coefficient	0.001
Gradient checkpointing	enabled
Remove padding optimization	enabled
Rollout engine	vLLM
Rollout tensor parallelism	2 during training
Rollout GPU memory utilization	0.4
Hardware configuration	1 node, 4 GPUs per node (4A100)
Checkpoint saving	effectively final checkpoint only; latest FSDP actor checkpoint is merged for inference

G Online RL results

Experiment scope. This appendix reports the spatial/curriculum results and the updated mathematical-transfer comparison. Spatial models were trained on Polyomino, whereas the math models were trained jointly on Role Assignment and MaxSAT. These use separate task-specific checkpoints from the Role Assignment-only comparison in Table 4. Appendix F.3 and the math subsection below describe their respective protocols. Spatial tables report proportions; the math comparison reports percentages.

G.1 Spatial reasoning

Evaluation uncertainty in Figure 4. Each prompt has eight saved responses; pass@8 indicates at least one successful response. Polyomino averages over 250, 250, 449, and 750 oracle-prefix prompts at levels α_1–α_4, respectively, from $G = 250$ puzzles per level. With m_g prompts and S_g successful prompts in puzzle g, $N = \sum_g m_g$ and $\hat{p} = \sum_g S_g/N$, the puzzle-clustered standard error is $\mathrm{SE} = \sqrt{\frac{G}{G-1}\sum_g (S_g - \hat{p}m_g)^2/N}$. This preserves prompt weighting despite unequal prefix counts. Spatial QA averages eight fixed capability groups of 25 questions each. With within-group sample variance s_h^2, its stratified error is $\mathrm{SE} = \sqrt{\sum_{h=1}^{8} s_h^2/25/8}$. Means and SEs are multiplied by 100. These evaluation SEs condition on each saved checkpoint; they are neither training-run variability nor 95% confidence intervals.

Table 10: Polyomino (Base vs Trained) — Progressive training L1–L4, reporting pass@k for $k \in \{1, 4, 8\}$.

	Level 1		Level 2		Level 3		Level 4	
pass@k	**Base**	**Trained**	**Base**	**Trained**	**Base**	**Trained**	**Base**	**Trained**
Pass@1	0.006	0.015	0.003	0.038	0.008	0.051	0.002	0.009
Pass@4	0.024	0.056	0.012	0.134	0.028	0.154	0.006	0.031
Pass@8	0.044	0.100	0.024	0.228	0.053	0.225	0.012	0.052

Table 11: Spatial 2D capabilities — Pass@8 at Level 1 and Level 2.

	Level 1				Level 2			
Capability	**Base**	**L1–L4**	**L1 Only**	**L4 Only**	**Base**	**L1–L4**	**L1 Only**	**L4 Only**
Translate	0.520	0.680	0.320	0.360	0.240	0.320	0.280	0.240
Rotate & Place	0.200	0.280	0.080	0.120	0.160	0.280	0.240	0.240
Reflect & Place	0.360	0.440	0.320	0.360	0.160	0.280	0.200	0.200
Compose (A,B,C)	0.080	0.200	0.040	0.040	0.040	0.120	0.040	0.160
Equivalence	0.520	0.960	0.760	0.520	0.440	0.840	0.880	0.440
Symmetry	0.520	0.760	0.400	0.600	0.600	0.600	0.520	0.560
Overlap IoU	0.200	0.120	0.120	0.120	0.120	0.040	0.040	0.080
Coverage	0.160	0.240	0.200	0.240	0.200	0.240	0.280	0.280

Table 12: Spatial 2D capabilities — Pass@1 at Level 1 and Level 2.

	Level 1				Level 2			
Capability	**Base**	**L1–L4**	**L1 Only**	**L4 Only**	**Base**	**L1–L4**	**L1 Only**	**L4 Only**
Translate	0.140	0.205	0.070	0.090	0.090	0.125	0.085	0.095
Rotate & Place	0.050	0.085	0.010	0.035	0.075	0.070	0.080	0.085
Reflect & Place	0.190	0.190	0.145	0.165	0.055	0.075	0.045	0.045
Compose (A,B,C)	0.020	0.035	0.005	0.005	0.015	0.035	0.010	0.030
Equivalence	0.295	0.605	0.630	0.325	0.325	0.505	0.680	0.315
Symmetry	0.100	0.180	0.125	0.100	0.190	0.170	0.175	0.130
Overlap IoU	0.025	0.020	0.015	0.015	0.015	0.005	0.010	0.015
Coverage	0.040	0.055	0.035	0.060	0.065	0.065	0.065	0.065

G.2 Math problems

Training and evaluation. We compare the Base and GRPO checkpoints of Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct. For each model, GRPO training uses the joint Role Assignment and MaxSAT curriculum with KL coefficient 0.01, one training seed, one epoch, 59 updates, 128 prompts per batch, eight responses per prompt, and learning rate 10^{-6}. Math benchmark questions are not included in this training. All four checkpoints are evaluated on the same AMC'23, AIME'24, and AIME'25 questions (40, 30, and 30) on A100 GPUs, with eight responses per question and decoding seed, temperature 1, top-$p = 0.7$, top-$k = 50$, and prompt/response limits of 1,024/2,048 tokens.

Table 13 reports pass@1 and pass@8 averaged over three decoding seeds. These seeds were carried forward from an earlier exploratory analysis and held fixed for this evaluation. Within each run, pass@1 averages the correctness of the eight responses to each question, while pass@8 records whether at least one is correct. Standard errors are the sample standard deviation of the three run-level estimates divided by $\sqrt{3}$. They describe decoding variation for fixed checkpoints and questions, rather than independent-training-run variability or uncertainty over a population of questions. GRPO has higher mean scores in all reported comparisons; the small benchmark sizes and single training run per model limit broader conclusions.

Table 13: Math transfer for Qwen2.5-3B and Llama-3.2-3B. Entries are mean pass@1 and pass@8 percentages $\pm$ decoding standard errors across three decoding seeds.

		Pass@1		Pass@8	
Model	Benchmark	Base	GRPO	Base	GRPO
Qwen2.5-3B	AMC'23	36.77 ± 1.33	38.75 ± 0.95	68.33 ± 3.00	72.50 ± 3.82
	AIME'24	5.42 ± 1.10	5.83 ± 0.96	16.67 ± 3.85	21.11 ± 2.22
	AIME'25	2.08 ± 0.24	2.78 ± 0.37	13.33 ± 1.92	15.56 ± 1.11
Llama-3.2-3B	AMC'23	18.65 ± 1.37	20.52 ± 0.55	51.67 ± 5.83	52.50 ± 1.44
	AIME'24	3.06 ± 0.14	4.03 ± 0.28	14.44 ± 1.11	15.56 ± 2.94
	AIME'25	0.28 ± 0.14	0.42 ± 0.24	2.22 ± 1.11	3.33 ± 1.92

H Optimization problems

Protocol and metrics. We compare Base and Role Assignment-trained checkpoints of Qwen2.5-3B-Instruct and Llama-3.2-3B-Instruct. Both use the Role Assignment-only pairwise recipe in Appendix F.3, with one training run per model; MaxSAT and Scheduling are out of distribution. Base and Trained within each model family use the same frozen prompts, scorer, decoding settings, and sampling seed, with eight responses per prompt. The Qwen pass@1 values reproduce the corresponding rows of Table 4.

All entries are percentages ± evaluation-root-cluster standard errors. Pass@1 averages the fraction of responses marked valid and matching the reference value; pass@8 records whether at least one of the eight responses meets both criteria. Prompts receive equal weight, and solver prefixes from the same original puzzle share a cluster. We apply the estimator in Appendix F.3 to each metric's prompt-level values. These errors describe evaluation uncertainty for fixed checkpoints, not independent-training-run variability.

Results. Qwen shows more consistent gains across the two sampling budgets: pass@1 improves in all twelve conditions, while pass@8 improves on Role Assignment and MaxSAT but decreases on Scheduling. Llama improves pass@1 at every Role Assignment and Scheduling level, with mixed MaxSAT results, while pass@8 decreases in all twelve conditions. Thus, higher single-response accuracy does not necessarily imply broader coverage with eight responses.

H.1 Role Assignment (ID)

Table 14: Role Assignment (ID): Base versus models trained on Role Assignment with pairwise advantages. Entries are pass@1 and pass@8 percentages ± evaluation-root-cluster standard errors.

		Pass@1		Pass@8	
Model	Level	Base	Trained	Base	Trained
Qwen2.5-3B	α_1	33.82 ± 0.78	43.53 ± 0.91	91.47 ± 1.03	94.67 ± 0.84
	α_2	25.57 ± 0.63	37.31 ± 0.65	86.00 ± 1.10	93.30 ± 0.86
	α_3	21.15 ± 0.57	33.47 ± 0.58	79.68 ± 1.22	89.76 ± 0.90
	α_4	16.87 ± 0.49	28.61 ± 0.56	70.00 ± 1.38	86.07 ± 0.87
Llama-3.2-3B	α_1	30.27 ± 0.84	46.67 ± 2.08	90.53 ± 1.16	55.60 ± 2.25
	α_2	24.52 ± 0.68	43.61 ± 1.97	83.60 ± 1.35	52.20 ± 2.10
	α_3	21.19 ± 0.56	39.10 ± 1.37	78.80 ± 1.20	53.12 ± 1.66
	α_4	18.76 ± 0.46	36.06 ± 1.41	74.60 ± 1.15	49.53 ± 1.70

H.2 MaxSAT (OOD)

Table 15: MaxSAT (OOD): Base versus models trained on Role Assignment with pairwise advantages. Entries are pass@1 and pass@8 percentages ± evaluation-root-cluster standard errors.

		Pass@1		Pass@8	
Model	Level	Base	Trained	Base	Trained
Qwen2.5-3B	α_1	42.11 ± 1.65	46.82 ± 1.73	91.40 ± 1.63	92.11 ± 1.68
	α_2	29.44 ± 1.45	35.09 ± 1.68	81.72 ± 2.29	84.14 ± 2.36
	α_3	25.49 ± 1.22	30.87 ± 1.40	74.82 ± 2.22	78.00 ± 2.13
	α_4	22.36 ± 0.92	25.09 ± 1.05	72.24 ± 1.89	73.16 ± 1.91
Llama-3.2-3B	α_1	45.25 ± 1.67	46.59 ± 2.43	90.68 ± 1.65	77.42 ± 2.55
	α_2	30.30 ± 1.34	27.37 ± 1.99	81.03 ± 2.18	58.28 ± 2.85
	α_3	25.89 ± 1.10	23.04 ± 1.62	76.04 ± 2.09	55.01 ± 2.56
	α_4	16.25 ± 0.77	12.78 ± 0.94	61.21 ± 2.15	42.46 ± 2.18

H.3 Scheduling (OOD)

Table 16: Scheduling (OOD): Base versus models trained on Role Assignment with pairwise advantages. Entries are pass@1 and pass@8 percentages ± evaluation-root-cluster standard errors.

		Pass@1		Pass@8	
Model	Level	Base	Trained	Base	Trained
Qwen2.5-3B	α_1	57.67 ± 0.87	60.08 ± 0.96	96.00 ± 0.54	93.84 ± 0.69
	α_2	44.34 ± 0.69	46.17 ± 0.76	89.40 ± 0.73	86.53 ± 0.85
	α_3	36.30 ± 0.50	38.96 ± 0.60	84.80 ± 0.76	80.23 ± 0.81
	α_4	37.30 ± 0.49	39.14 ± 0.59	86.51 ± 0.83	81.66 ± 0.86
Llama-3.2-3B	α_1	45.44 ± 0.84	55.97 ± 1.27	91.60 ± 0.81	69.20 ± 1.15
	α_2	34.30 ± 0.61	44.78 ± 1.05	85.67 ± 0.91	58.73 ± 1.21
	α_3	28.89 ± 0.45	38.71 ± 0.83	79.71 ± 0.83	57.71 ± 1.01
	α_4	29.09 ± 0.48	37.91 ± 0.81	79.77 ± 0.88	55.60 ± 1.00

Table 17: Complete search-only results for Exp. 1. Entries are estimates ± paired-bootstrap standard errors from 20,000 resamples of the 16 task-by-difficulty cells. These errors measure sensitivity to benchmark-cell composition, not within-cell or seed variability. Rates are percentages; b_{eff} and k_{90} are dimensionless. Pass@64 is calibrated restart success. The sequential baseline is shared across both blocks.

Configuration	$p_g \uparrow$	$b_{\text{eff}} \downarrow$	$k_{90} \downarrow$	pass@64↑	Feasible↑	Exact↑
MCTS⋆						
Check + de-duplication	12.2 ± 3.3	15.04 ± 2.86	56.5 ± 5.9	88.9 ± 1.4	72.8 ± 6.6	10.9 ± 3.4
No check/prune	4.0 ± 1.5	25.70 ± 3.52	73.8 ± 5.4	84.8 ± 1.4	16.4 ± 3.0	3.7 ± 1.5
No check or de-duplication	5.1 ± 2.0	25.26 ± 3.49	74.3 ± 5.2	84.6 ± 1.3	19.8 ± 3.6	4.8 ± 2.0
Sequential (outcome-based)	4.9 ± 2.0	25.30 ± 3.56	74.2 ± 5.3	84.6 ± 1.4	19.8 ± 3.8	4.6 ± 2.0
Solver reference	0.2 ± 0.1	520.1 ± 135.7	1776 ± 179	18.7 ± 5.6	2.3 ± 0.9	0.5 ± 0.3
BeamSearch⋆						
Check + de-duplication	7.2 ± 1.5	13.04 ± 2.05	43.0 ± 5.5	93.8 ± 1.4	53.3 ± 3.8	7.8 ± 2.1
No check/prune	1.4 ± 0.4	53.07 ± 9.68	180.7 ± 29.3	65.4 ± 4.3	10.2 ± 1.7	2.0 ± 0.7
No check or de-duplication	2.5 ± 0.7	82.88 ± 16.12	329.9 ± 44.9	50.9 ± 4.2	12.8 ± 2.6	2.7 ± 1.0
Sequential (outcome-based)	4.9 ± 2.0	25.30 ± 3.56	74.2 ± 5.3	84.6 ± 1.4	19.8 ± 3.8	4.6 ± 2.0
Solver reference	0.2 ± 0.1	499.4 ± 135.6	1743 ± 185	19.4 ± 5.7	2.5 ± 1.1	0.6 ± 0.4

H.4 Additional results MCTS

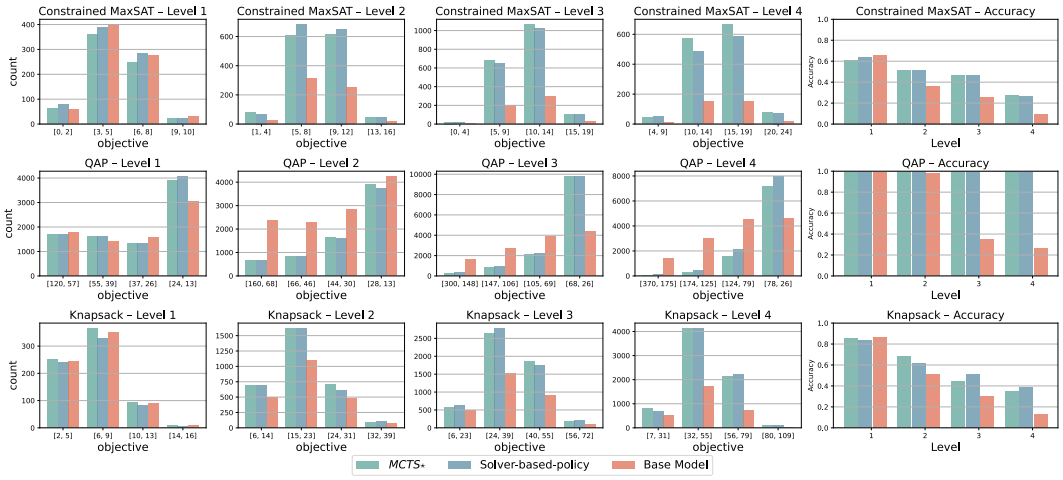


Figure 6: Accuracy and solution-quality histograms for baseline, pure MCTS, and solver-guided MCTS on QAP, 0/1 Knapsack, and Constrained MaxSAT tasks as complexity (Level 1 → 4) increases.

H.5 Additional search-only diagnostics for Exp. 1 on OPT⋆

Setup. This appendix provides additional search-only diagnostics for Exp. 1. All results use the same no-training setting as in the main text, with the union reference pool, relative tolerance $\epsilon_{\text{rel}} = 0.05$, and target failure probability $\delta = 0.10$ for the sample-complexity estimates. We compare the full search configuration (S1: feasibility checking/pruning with duplicate merging), the no-check/pruning ablation (S2), the cumulative ablation without checking/pruning or de-duplication (S3), the sequential baseline when available (S4), and the solver-reference trajectory generator (S5). We use the same component labels for MCTS⋆, the parent-initialized MCTS⋆ variant, and BeamSearch⋆ as in the main text.

Metric definitions. For a task instance t, let $V_t^\star$ denote the best value found in the reference pool. A terminal trajectory is counted as good if its task-normalized relative gap from $V_t^\star$ is at most ϵ_{rel}. This definition applies to both maximization tasks and minimization tasks such as QAP. We denote the rollout-level probability mass of good terminal trajectories by p_g.

Aggregation and uncertainty for Table 2. Let $\mathcal{R}$ be the set of evaluated roots and $R = |\mathcal{R}|$. For each root r, let G_r count $\epsilon_{\text{rel}} = 0.05$-good terminals among its M_r rollouts. The reported aggregates are

$$p_g = \frac{\sum_{r\in\mathcal{R}} G_r}{\sum_{r\in\mathcal{R}} M_r}, \qquad b_{\text{eff}} = \exp\left[-\frac{1}{R}\sum_{r\in\mathcal{R}} \log\left(\frac{G_r + 1/2}{M_r + 1}\right)\right]. \tag{5}$$

The pooled p_g is an unsmoothed, rollout-weighted frequency, displayed as a percentage. By contrast, b_{eff} gives each root equal weight in log space and uses the Jeffreys-smoothed rate $(G_r + 1/2)/(M_r + 1)$, which remains positive even when no good terminal is observed at a root. It is therefore the reciprocal geometric mean of these smoothed rootwise rates. Feasibility and exact optimality average rootwise rates, whereas p_g pools rollouts; these weightings differ when roots have different rollout counts, as in beam search. Weighting, smoothing, and nonlinear aggregation differ, so b_{eff} need not equal $1/p_g$; even with equal rollout counts, the arithmetic and geometric aggregates differ.

For full MCTS$\star$ with checking and de-duplication, 4,000 roots with 20 rollouts each give $p_g = 9{,}743/80{,}000 \approx 0.1218$, displayed as 12.2%. The rootwise log aggregate gives $b_{\text{eff}} = \exp(10{,}843.775004/4{,}000) \approx 15.04$, whereas the reciprocal pooled rate is approximately 8.21. These are different summaries of the same terminal observations. This arithmetic explains their numerical difference; it does not independently test Theorem A.3.

For all configurations in Tables 2 and 17, we use 20,000 paired bootstrap resamples of the 16 retained task-by-difficulty cells (four tasks and four levels; bootstrap seed 20260504), resampling the same cells jointly for every method and recomputing each aggregate. The standard errors are the sample standard deviations across resamples. They measure sensitivity to benchmark-cell composition, not within-cell or across-seed variability, and do not constitute paired-difference significance tests. The original searches used one repeat with seed 0; the sequential baseline is shared across both search families. In the calibration formulas below, $\widetilde{p}_g$ denotes a smoothed good-terminal probability at a fixed root; applying nonlinear transforms to pooled averages is generally different from aggregating rootwise diagnostics.

We also report the sample-complexity estimate

$$k_{90} = \left\lceil \frac{\log(0.10)}{\log(1 - \widetilde{p}_g)} \right\rceil, \tag{6}$$

which is the number of independent samples needed to obtain at least one good terminal with probability at least 0.90 under the calibrated restart model. For independent restarts at a fixed root, the calibrated success probability is

$$\text{pass@}k = 1 - (1 - \widetilde{p}_g)^k. \tag{7}$$

Tables 2 and 17 report this calibrated probability at $k = 64$, averaged over roots; k_{90} is likewise averaged over roots. These are restart-model estimates, rather than observed success rates from 64 independent searches. In addition, we report terminal feasibility, exact optimality, and the crossover ratio $\rho = b_{\text{eff}}/b_{\text{uniform}}$. Values of $\rho < 1$ indicate that the search policy concentrates more probability mass on good terminals than uniform sampling from the reference pool.

QAP. The Quadratic Assignment Problem (QAP) instances in OPT* are minimization tasks. Each instance assigns facilities bijectively to grid locations, with objective equal to the sum of flow-weighted Manhattan distances over facility pairs. QAP is a useful stress test because the number of possible terminal assignments grows factorially with the number of unassigned facilities. As a result, even small changes in good-terminal mass can lead to large changes in b_{eff} and pass@k.

Discussion. The task-separated branching curves support the main-text claim that feasibility checking/pruning and duplicate merging reduce the effective branching burden before training. On the branching readout, S1 achieves lower b_{eff} curves than the no-check and no-deduplication ablations across the tested difficulty levels. By contrast, the solver-reference trajectory generator has high b_{eff}, reflecting that it is not a calibrated stochastic search policy over all good terminals.

The aggregate and QAP-specific results show the same component effect through pass@k, terminal feasibility, and k_{90}. It is important to distinguish pass@k from exact optimality: pass@k uses the ϵ_{rel}-good threshold, so a method can often reach near-optimal terminals even when its exact-optimality

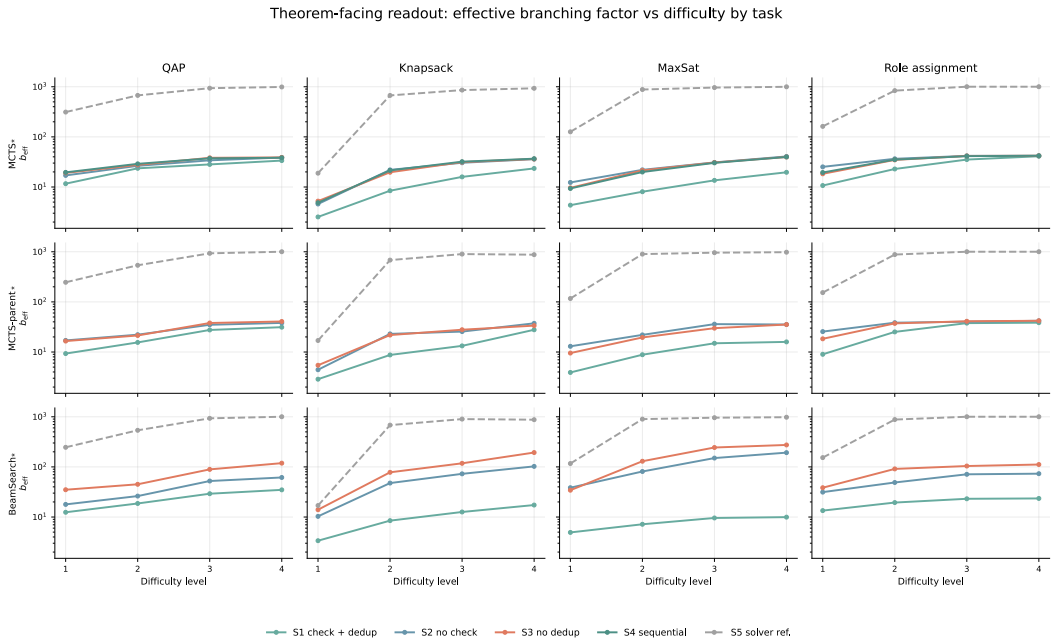


Figure 7: Effective branching factor versus OPT* difficulty level, separated by task and search family. Each column is a task, and each row is a search family. The y-axis is logarithmic, with lower values indicating better search concentration. Across tasks, the full S1 configuration achieves the lowest or near-lowest b_{eff} among model-based generators, while removing checking/pruning or duplicate merging increases the difficulty-dependent branching burden.

rate is lower. For this reason, the solver-reference rows should be interpreted as a trajectory-generation baseline, not as a statement about solver optimality itself.

A supplementary Knapsack audit uses exact feasible-subset counts of 45, 75, 215, and 560 across the four difficulty levels as structural proxies. Their logarithms have a negative association with observed without-replacement pass@8 (Spearman $\rho \in [-0.800, -0.600]$; exact permutation $p \in [0.333, 0.417]$). Only four levels are compared, with multiple task properties changing together. This is a non-significant descriptive consistency check: feasible-subset counts do not estimate N_α, and the audit does not test the theorem's information requirement.

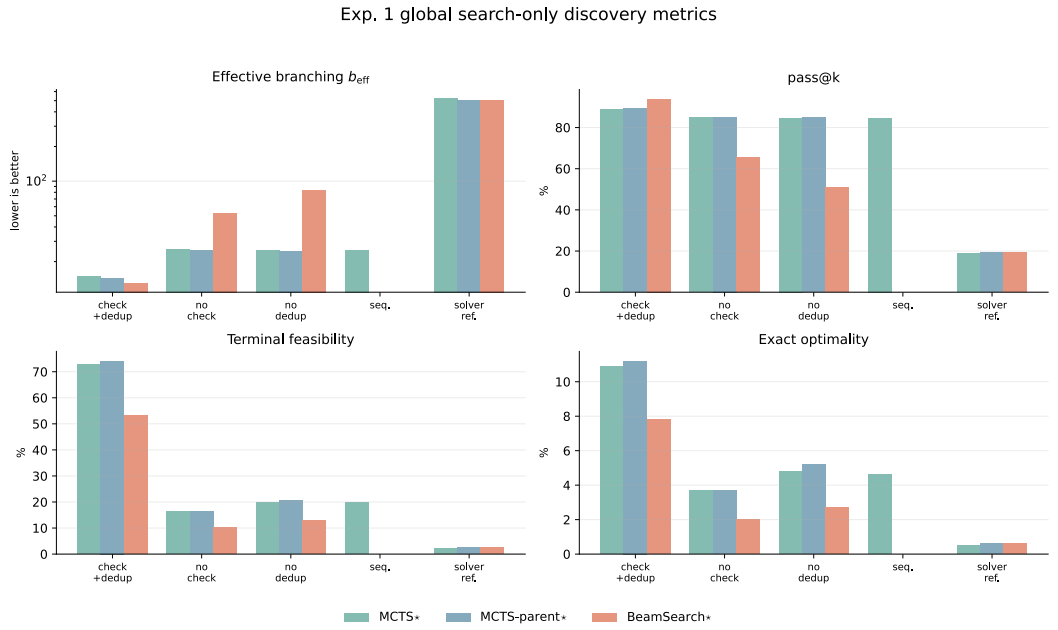


Figure 8: Global search-only metrics across the OPT* tasks, with $k = 64$ for pass@k. Among model-based generators, S1 consistently yields the smallest effective branching factor and the strongest pass@k. The ablations disable pruning (S2), or both pruning and duplicate merging (S3), and reduce terminal quality relative to S1.

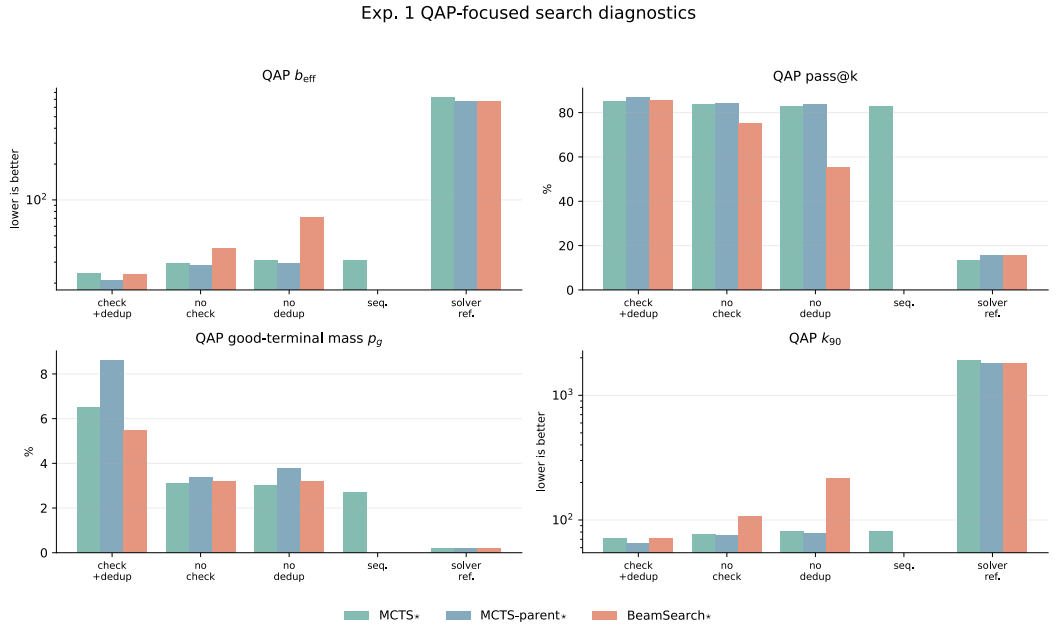


Figure 9: QAP-focused search metrics, with $k = 64$ for pass@k. QAP metrics average the four difficulty-level summaries and follow the same component pattern as the global aggregate, but the separation from the solver-reference generator is sharper under the trajectory-mass readout. Among the attached reports, the parent-initialized MCTS$\star$ variant achieves the lowest QAP b_{eff}.

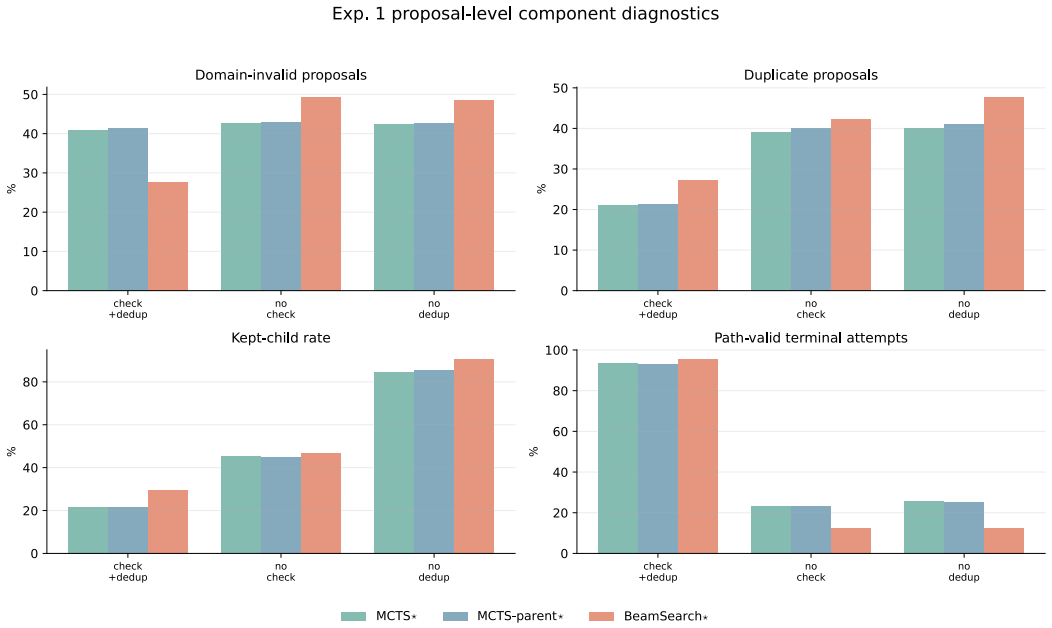


Figure 10: Proposal-level component diagnostics. The full S1 configuration retains fewer raw proposals, but the retained proposals are substantially more likely to remain path-valid and terminal-feasible. S2 exposes the search to invalid actions, while S3 retains many duplicate proposals; both effects reduce search efficiency.

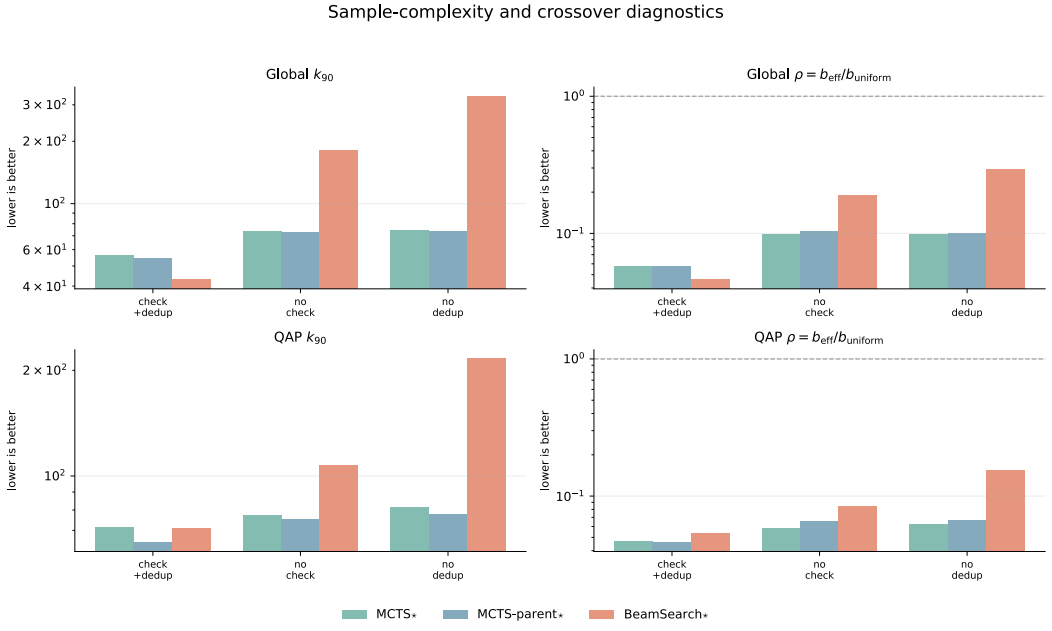


Figure 11: Sample-complexity and crossover readouts. Lower k_{90} means fewer samples are needed to observe an ϵ_{rel}-good terminal with high probability. Lower $\rho = b_{\text{eff}}/b_{\text{uniform}}$ means the search procedure is more efficient than the uniform reference-pool proxy.

I Autonomous four-action control: additional protocol details

Fixed-horizon construction. The comparison in Exp. © uses newly constructed Knapsack and Role Assignment instances requiring exactly four structured decisions. Some instances in the original experiments could finish in two decisions, providing limited separation between terminal and intermediate supervision. Four decisions create three genuine nonterminal states. The new control retains the same task families and difficulty principles, but its training and evaluation IDs differ from those used for the solver-prefix results in Table 4. Within this control, every method shares the same training/evaluation IDs and splits, prompts, action schema, parser, invalid-action handling, and terminal evaluator.

Interaction and supervision. The implementation details below describe the inspected runner; historical run settings remain to be confirmed. The initial prompt presents the instance and requests one structured action. The environment parses and validates each action and tracks the state internally. Subsequent turns retain the conversation history and receive a generic continuation prompt, without refreshed state descriptions, solver scores, or oracle actions. A complete legal solution requires four actions. Outcome GRPO assigns the signed terminal objective to a valid complete trajectory and an invalid sentinel otherwise. Outcome RS/SFT selects the highest-quality valid trajectory from each online group, skipping all-invalid groups. Outcome DPO uses a best-versus-worst terminal-quality pair per non-tied group, with the initial model as fixed reference and $\beta = 0.1$; skipped groups are not replaced.

Solver credit and pairwise advantages. Let $V(s)$ be the exact best achievable terminal objective, signed so that higher is better. The solver-based arms call the domain's exact completion solver and use

$$r_{i,t} = V(s_{i,t+1}) - V(s_{i,t}), \qquad A_{i,t}^{\text{rank}} = \frac{W_{i,t} - L_{i,t}}{K_t - 1}.$$

Solver-based GRPO centers and standardizes the numerical rewards within each root-and-turn group. The ranking arm instead uses the pairwise advantage above, with $W_{i,t}$ and $L_{i,t}$ counting wins and losses against the other $K_t - 1$ trajectories in that group. Valid rewards outrank invalid ones; ties contribute zero, and singleton groups receive zero. Trajectories share their initial instance and action index but may occupy different intermediate states. Each advantage applies only to assistant-generated tokens at that turn, with no additional normalization in the ranking arm. For fixed-prefix candidates, subtracting the common pre-action value preserves score ordering; for autonomous trajectories, the pre-action values can differ.

Invalid actions and incomplete trajectories. An invalid action receives the sentinel -99999, leaves the internal state unchanged, and invalidates the trajectory; generation continues for the fixed horizon. An incomplete or infeasible final trajectory without an earlier invalid action receives the sentinel on its final dense-reward event, preserving earlier turn credit. The pairwise estimator treats the sentinel as invalid rather than using its numerical magnitude.

Matched policy-optimization budget. The three GRPO variants share initialization, horizon, rollout group size, training-step budget, and token limits. The matched implementation also shares clipping, token-mean policy loss, and disabled KL regularization and group filtering. Consequently, the solver-based pair isolates the advantage estimator within this recipe; comparison with Outcome GRPO additionally changes credit timing. Equal token limits do not imply equal realized token counts, solver calls, or wall-clock cost. Outcome RS/SFT and DPO share the construction and evaluation protocol but use different training objectives. Table 18 records the current ordered-runner configuration, separately from the solver-prefix settings.

Table 18: Current autonomous implementation configuration. These are defaults of the inspected ordered-training runner; resolved historical run manifests were unavailable. Token limits specify allocations, not measured token use.

Setting	Configured value
Initial model	Qwen2.5-3B-Instruct
Action horizon	Four assistant reasoning/action turns
Training roots per domain	512 per level; 2,048 across four levels
Training schedule	One epoch, $\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4$, no shuffle; 32 roots/batch; 64 training steps
Training rollout group	Eight trajectories per root
Actor learning rate	10^{-6}
Held-out evaluation	64 roots/level, 256/domain; eight trajectories/root
Generated-token cap	2,048 tokens per assistant action
Prompt / response / context limits	1,792 / 12,288 / 18,432 tokens
GRPO-arm policy loss	Token mean; clipping 0.2 below and 0.28 above, dual-clip cap 10
GRPO-arm KL regularization	No reward KL or actor-loss KL
Group filtering / failed-group refill	Both disabled

Autonomous evaluation and uncertainty. No method receives solver actions, best-completion values, or search at evaluation. The policy constructs all four decisions from the initial instance, its conversation history, and continuation prompts. Each domain has 256 held-out roots with eight saved rollouts per root. Terminal feasibility, exact optimality, and $p_g = \mathrm{pass@1}$ use the reported evaluator readouts; pass@8 is observed directly from the eight rollouts rather than inferred from pass@1. The branching statistic b_{eff} is a descriptive diagnostic. It and pass@8 share the same good-terminal indicators, so their within-set association is not an independent predictive test. Establishing predictive validity would require a separate held-out study across tasks or training stages. The $\pm$ terms in Table 3 are approximate 95% root-cluster confidence margins; they quantify held-out-root sampling uncertainty and do not quantify variation across independent training runs.

J Broader impacts

OPT$\star$ provides controlled, automatically verifiable tasks for studying decision-making and may support more efficient planning and resource allocation. These synthetic results do not establish reliability for deployment. In applications such as scheduling or role assignment, optimizing an incomplete objective can disadvantage people or omit fairness and other domain constraints, even when every modeled constraint is satisfied. Feasibility checking and solver optimality certify only the encoded problem, not that its objectives or constraints are socially appropriate. Real-world use therefore requires domain-specific validation, appropriate fairness and safety constraints, and human oversight.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately summarize the OPT⋆ framework, the theoretical motivation, and the offline/online RL experiments; the claims are scoped to optimization-like reasoning over scalable auto-verifiable tasks.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion discusses solver availability and cost, the autonomous comparison's restriction to two synthetic domains, and evaluation uncertainty rather than independent training-run variability, alongside future work on task scaling and generalization.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical results state their assumptions and are proved in Appendix A, with the main theorem and supporting lemmas/corollaries cross-referenced from the main text.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [No]

   Justification: Section 5 and Appendices C–F and I document task construction, prompts, algorithms, and available settings. However, the historical autonomous-run configurations and some success and uncertainty definitions remain to be verified.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The current manuscript provides detailed reproducibility information, but it does not yet include an open code/data release link with executable reproduction commands.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [No]

   Justification: Training and evaluation settings are provided in Section 5 and Appendices F.2–F.3 and I. The resolved settings underlying the autonomous results and their exact success and uncertainty definitions remain to be confirmed.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We report bootstrap and evaluation standard errors and approximate root-cluster confidence margins in Tables 2, 3, and 4 and Fig. 4. Some uncertainty definitions remain incomplete, including the autonomous interval construction; independent training-run variability is not established.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The appendix reports GPU counts and parallelism, including four A100 GPUs for solver-prefix training, but per-experiment memory, execution time, and individual and total compute consumption are not fully reported.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses synthetic auto-verifiable tasks and public model/dataset benchmarks, does not involve human subjects or private data, and is intended for responsible study of reasoning methods.

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix J discusses potential benefits for allocation and scheduling, risks from omitted fairness objectives and verification limits, and the need for human oversight.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk scraped datasets, image generators, or new general-purpose pretrained models; the proposed assets are synthetic task generators and evaluation/training procedures.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The manuscript cites the main existing models, benchmarks, and tools used, but it does not yet explicitly list licenses and terms of use for each asset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [No]

    Justification: Appendices B–D and F describe the new task families and interfaces, but this manuscript does not provide a documented asset release with accompanying licenses and reproduction instructions.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, or research with human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human-subject research, so IRB or equivalent approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM usage is central to the method and is described throughout the paper, including the language interface, offline/online RL procedures, model backbones, prompts, and evaluation setup.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.