LINTREE: IMPROVING LLM REASONING WITH EXPLICITLY STRUCTURED SEARCH HISTORIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are increasingly used to perform or guide search in domains such as geometry theorem proving, formal proof search, and planning. A growing line of work trains LLMs to carry out the search process autoregressively, conditioning each decision on a trace of previous exploration, including failed attempts and backtracking. Compared with traditional heuristic-guided search, such a policy has a potential advantage: it conditions on the whole search trace rather than only on the current local state. We first examine this potential by comparing trace-conditioned policies against best-first search equipped with an LLM heuristic that only observes the current local state. Across three controlled domains, Blocks World, grid Navigation, and Sokoban, we find that raw access to search history alone is not enough to reliably outperform heuristic search. We then study one possible reason: the search tree structure is only latent in the trace, and when the model backtracks or switches branches, the trace does not explicitly identify which earlier search state is being revisited. We show that adding parent pointers that make this latent structure explicit (**LinTree**) improves both task performance and search efficiency over implicit policies, and matches the solve rate of LLM-heuristic-guided search while using fewer expansions. These results suggest that, for search problems, making the structure of the search history explicit helps LLMs learn more effective search behavior.

1 INTRODUCTION

Large language models (LLMs) are increasingly used to perform or guide search across a range of planning and search problems. AlphaGeometry (Trinh et al., 2024) trains a language model to propose auxiliary constructions inside a symbolic geometry solver, LLM-based provers search for formal proofs in Lean (Yang et al., 2023; Xin et al., 2024), and transformers have also been trained directly on serialized search traces, learning search procedures for planning and combinatorial reasoning tasks (Lehnert et al., 2024; Gandhi et al., 2024). Across these settings, learned models are becoming increasingly integrated into the search process itself. In some systems the model guides an external search controller, while in others it generates the search process autoregressively from a sequence of previous search steps. These search steps include intermediate exploration, failed branches, and backtracking, allowing the model to generate an entire search process as a sequence. Once search is represented in this way, the choice of representation becomes part of the system design: the model must read and produce a particular formalism chosen in advance. AlphaGeometry's model, for example, produces constructions in a formal geometry language for a symbolic engine to check, while models trained on search traces read and write a trace format specified by the designer.

A standard approach to such search problems is heuristic-guided search (Hart et al., 1968; Pearl, 1983), where a heuristic scores each frontier node from its local state and the search algorithm decides which node to expand next. A trace-conditioned LLM policy differs in one important way: instead of choosing the next expansion by scoring frontier nodes using local state information, it generates the next search step conditioned on the entire preceding search trace. With this richer information, the model could in principle use evidence gathered on one branch to guide choices on another, avoid revisiting neighborhoods that earlier expansions have already shown to be unproductive, and coordinate search using non-local context that a local heuristic never sees. Whether LLMs actually make effective use of this additional information, and whether it gives trace-conditioned search an advantage over local heuristic search, is not yet clear.

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

This framing suggests the first question we study: does conditioning on the whole trace translate into stronger search? We test this by comparing trace-conditioned policies that observe the whole search trace against best-first search equipped with a local-state LLM heuristic that sees only the current state and goal, with both models trained using the same supervised fine-tuning and reinforcement learning pipeline. Across three controlled domains, Blocks World, grid Navigation, and Sokoban, we find that trace access alone is not enough: the trace-conditioned policy does not reliably outperform the local-state heuristic baseline.

This result raises a follow-up question: why does trace access not help more? One explanation lies in how the trace presents its information. Each expansion descends from a state that already appears in the trace (Besta et al., 2024; Sel et al., 2023), but the trace does not name it, so the tree structure has to be inferred. This motivates our second question: does making the information that is latent in the trace explicit allow the model to use it? We test this by explicitly encoding the latent search-tree structure in the trace: each search step records the frontier state being expanded, exposing the parent-child relationships between states. We compare this representation against one in which these relationships remain implicit. The two formats are derived from the same underlying searches and are trained with the same supervised fine-tuning (SFT) followed by reinforcement learning (RL) pipeline. Differences between them therefore isolate the effect of making the latent tree structure explicit rather than differences in trace access, optimization, or task difficulty. Intuitively, this turns a loosely narrated search trace into a serialized search tree with clear topology (**LinTree**), making the available history easier for the model to use.

The RL stage of this pipeline requires a reward that values both a correct plan and a short search. Because correctness matters more than length, we require the reward to order traces lexicographically, comparing correctness first and search length only among correct traces, and we design a reward with this property.

We study these questions empirically in three controlled domains, Blocks World (Valmeekam et al., 2022), Grid Navigation (Mittal et al., 2025), and Sokoban (Guez et al., 2018). Our findings are summarized as follows:

- While trace-conditioned policies generate search steps conditioned on the whole search trace, they do not fully exploit this information when the tree structure is left latent: the policy does not reliably outperform a local-state heuristic baseline despite having access to more information.
- Making the latent tree structure explicit changes this: trained the same way, policies with explicit parent pointers outperform implicit policies on both solve rate and search efficiency, and match the solve rate of local heuristics with fewer expansions. They also extract plans from their own traces more reliably and explore the state space more widely.
- We design the RL reward to be lexicographic, so that correctness takes priority over search length. We show that our reward satisfies this property, and that a reward without it destabilizes training.

2 Related Work

Learning Search Policies from Search Traces. A growing line of work trains language models to carry out search themselves, by learning from serialized search traces in problems whose states and actions are precisely specified. Searchformer (Lehnert et al., 2024) trains transformers on A^* execution traces in maze navigation and Sokoban, and Stream of Search (Gandhi et al., 2024) trains on serialized depth-first and breadth-first searches for an arithmetic game, both showing that models can learn from serialized search processes and subsequently improve their search behavior. These works focus on learning and improving search policies from traces, while leaving the effect of the trace representation itself largely unexplored. Our work studies this design choice directly, comparing trace formats that differ only in whether the search tree is explicit, under the same training pipeline.

Chain-of-thought Reasoning and Self-reflection. Chain-of-thought (Wei et al., 2022; Nye et al., 2021) prompting improves LLM reasoning by eliciting a sequence of natural-language reasoning steps before the final answer. Subsequent work extends this basic paradigm through problem decomposition, reflection, and iterative refinement, allowing models to try multiple approaches, break problems into

simpler subproblems, or revise earlier attempts (Renze & Guven, 2024; Zhou et al., 2022; Madaan et al., 2023; Shinn et al., 2023). These approaches make reasoning more flexible, but the branching, revision, and selection process is typically left implicit in the generated text rather than represented as an explicit search structure.

LLMs within External Search Procedures. Another line of work uses LLMs as components of an externally managed search procedure. AlphaGeometry (Trinh et al., 2024) uses a language model to propose auxiliary constructions that are checked and incorporated by a symbolic geometry engine, while LLM-based theorem provers generate candidate proof steps within explicit proof-search procedures (Yang et al., 2023; Xin et al., 2024). For general reasoning, Tree of Thoughts (Yao et al., 2023), LATS (Zhou et al., 2023), and Reasoning-as-Planning (Hao et al., 2023) use tree-search algorithms to explore LLM-generated intermediate reasoning states, while Graph of Thoughts (Besta et al., 2024) extends this idea to graph-structured reasoning and beam-search methods maintain multiple partial reasoning chains during decoding (Xie et al., 2023). Learned heuristics for classical search follow the same broad pattern: a model scores candidate states while an external algorithm maintains the frontier and controls expansion (Mittal et al., 2025). Across these approaches, the global search structure is maintained outside the language model, and individual model calls condition on selected states, paths, or intermediate thoughts rather than on a serialization of the entire search history. Our work contrasts this setting with trace-conditioned search policies, asking whether access to the full search trace provides an advantage over local-state LLM guidance.

3 Reasoning Environments

We evaluate on three fully observable domains: Blocks World (Valmeekam et al., 2022), where the agent restacks blocks into a target configuration; Grid Navigation (Mittal et al., 2025), where the agent moves to a goal cell on a 2D grid with obstacles; and Sokoban (Guez et al., 2018), where the player pushes boxes onto goal squares. All three domains naturally require search, have exactly specified states and actions with a known transition function, and can benefit from strong heuristics to improve search efficiency. Sokoban is the hardest of the three, with longer search traces and more complex transition dynamics. For each domain, SFT traces are generated using best-first search with a rule-based heuristic. State and action representations, instance generation, dataset sizes, and the rule-based heuristics are detailed in Appendix A, and Figure 1 shows an example of each domain.

4 Trace-conditioned Search and Local Heuristics

We begin by comparing two ways to train LLMs for search: generating search traces and learning a local-state heuristic. In the trace-conditioned setting, the LLM observes the full search trace and directly predicts the next search step. In the local-state setting, we run an external best-first search algorithm guided by a heuristic, learned using the same LLM, that observes only the current candidate and the goal when expanding. Both policies are built on the same backbone LLM. We use two backbones, Qwen3-0.6B (Yang et al., 2025) and Qwen3.5-4B, and report results for both: Tables 2 and 3 for Qwen3-0.6B, and Table 6 in Appendix C for Qwen3.5-4B.

4.1 Trace-conditioned LLM reasoning policy

We train an LLM reasoning policy that conditions on the full search history and predicts the next search step. The search trace is linearized as a sequence of expansions, with each search step written using the template

```
EXPAND ACT {action} -> {resulting state}
```

This serialization mirrors the style of reasoning model traces, where the model tries different actions and backtracks when a branch fails but does not explicitly label the tree structure. Nevertheless, conditioning on the preceding trace allows the policy, in principle, to use non-local context to guide search decisions.

The training follows two stages: supervised fine-tuning (SFT) on best-first search traces, followed by reinforcement learning with GRPO. We describe the training details as follows.

Stage 1: SFT on best-first traces. For each domain, we generate search traces by running best-first search with a rule-based heuristic as described in Section 3. The resulting traces demonstrate valid search behavior (branch expansion, backtracking, and solution construction) and serve as the SFT data for the policy. Each SFT example consists of a prompt and a completion. The prompt contains the instance configuration: the initial and goal states for Blocks World, the grid with obstacles together with the start and goal cells for Navigation, and the grid with walls, boxes, and goals for Sokoban. The completion has two parts: (i) the thinking process, consisting of the search trace, written as a sequence of expansions, and (ii) the final plan, which is the sequence of actions that leads to the goal extracted from the search trace. The model is trained to generate both parts, conditioned on the prompt. Training details are given in Appendix B.

Stage 2: GRPO for search efficiency. After SFT, the LLM is initialized with reasonable search behavior, but since the rule-based heuristic that produced the SFT traces may not be optimal and the model may not fully exploit its access to the trace, we optimize the policy with GRPO (Shao et al., 2024), with a reward encouraging the model to improve its search for both correctness and efficiency.

In this class of problems we want two things from a trace: the plan should be correct, and its search should be short. Correctness is the more important of the two, so the reward should not allow a reduction in search length to compensate for failure. We make this requirement precise as a lexicographic ordering property. A trace τ is described by attributes $a_1(\tau), \ldots, a_k(\tau)$ listed in decreasing order of priority. Two traces are compared at the first level at which they differ, and the comparison used at level i may depend on the values of the higher-priority attributes $a_1, \ldots, a_{i-1}$. A reward R satisfies the lexicographic ordering property if $R(\tau) > R(\tau')$ whenever τ is preferred to τ' under this comparison. This is slightly more general than the standard lexicographic order, in which the comparison at each level is fixed in advance. In our setting there are two attributes: a_1 is correctness, with a valid and correct trace preferred to any other, and a_2 is the number of expansions, with fewer expansions preferred when the trace is correct and no preference otherwise. The dependence on the first level is what lets us compare length only among correct traces. The property rules out rewards in which objectives are traded off against one another. This property may also be relevant in safety and alignment contexts, where the safety objective is more important than the performance objective.

We use a reward with two components. First, the model receives a correctness reward that is either 1 or 0, depending on whether the generated search trace is valid and the resulting plan correctly solves the task. Second, when the trace and plan are correct, we apply an additional efficiency term that penalizes longer searches:

$$R(\tau) = \mathbf{1}[\mathrm{valid}(\tau) \wedge \mathrm{correct}(\tau)] \left(1 - \lambda \sum_{t=0}^{N_{\mathrm{exp}}(\tau)-1} \gamma^t\right), \tag{1}$$

where $N_{\mathrm{exp}}(\tau)$ is the number of search expansions in trace τ, $0 < \gamma < 1$ is the geometric decay factor, and λ sets the penalty scale. Each additional expansion subtracts one more discounted term from the correctness reward, so shorter correct traces receive higher return, while every incorrect trace receives zero.

Proposition 1. *Let $0 < \gamma < 1$ and $0 < \lambda < 1 - \gamma$. Then the reward in Eq. 1 satisfies the lexicographic ordering property for correctness and number of expansions: for any two traces τ and τ', (i) if τ is valid and correct and τ' is not, then $R(\tau) > R(\tau')$; (ii) if both are valid and correct and $N_{\mathrm{exp}}(\tau) < N_{\mathrm{exp}}(\tau')$, then $R(\tau) > R(\tau')$.*

The proof is provided in Appendix E. The bound on λ ensures that every correct trace receives a higher reward than any incorrect trace. When using a deterministic policy together with an approximator capable of representing the optimal policy, this is sufficient to incentivize the policy to find the correct policy with the minimum trace length for the observed problems. However, LLM policies may not be able to represent the optimal policy and can be non-deterministic. In these cases, we note that the expected reward may not satisfy the type of lexicographic ordering that we want for these types of tasks: the expected reward can be made higher by trading off probability of success for shorter successful searches within a range of parameters (see Appendix F). We would generally prefer the lexicographic ordering for the expected reward, so this would be an example of reward hacking. Despite this, we found the lexicographic reward property to be important in our experiments: with a

Algorithm 1 LLM heuristic-guided best-first tree search.

Require: initial state s_0, goal g, transition function T, LLM heuristic $h_\theta(s, a, g)$, search budget B
Ensure: plan from s_0 to a goal state, or FAIL

Initialize the search tree with root s_0
$\mathcal{V} \leftarrow \{s_0\}$ ▷ visited states, used for cycle avoidance
$\mathcal{F} \leftarrow \{(s_0, a) : a \in \text{Actions}(s_0)\}$ ▷ frontier of (leaf-state, candidate-action) pairs
for $t = 1$ to B **do**
  for each $(s, a) \in \mathcal{F}$ **do**
    Compute $h_\theta(s, a, g)$ ▷ LLM heuristic scores candidate state-action pairs
  end for
  $(s^\star, a^\star) \leftarrow$ candidate with the lowest h_θ value ▷ or sample from softmax during training
  $s' \leftarrow T(s^\star, a^\star)$ ▷ environment transition for the chosen action
  Add s' as a child of $s^\star$ in the tree; $\mathcal{V} \leftarrow \mathcal{V} \cup \{s'\}$
  if s' satisfies g **then**
    return the path from s_0 to s' in the tree
  end if
  $\mathcal{F} \leftarrow \big(\mathcal{F} \setminus \{(s^\star, a^\star)\}\big) \cup \{(s', a) : a \in \text{Actions}(s'),\ T(s', a) \notin \mathcal{V}\}$ ▷ skip already-visited successors
end for
return FAIL

constant per-step penalty, a sufficiently long correct trace can score below an incorrect one, causing GRPO training to collapse (Appendix G). By imposing the lexicographic property, we are able to address one of the failure modes of the algorithm. Understanding the possible constraints on the expected correctness and length rewards that can be achieved through design of the reward function, e.g. as done in Tan et al. (2026) to simultaneously learn for accuracy and confidence calibration, is left to future work. In our experiments, we set $\lambda = 0.005$ and $\gamma = 0.99$.

4.2 LOCAL-STATE LLM HEURISTICS

Our local-state LLM heuristic resembles a common pattern for applying LLMs to search problems: the LLM is invoked locally to score or propose continuations from a tree node, while an outer search procedure decides which node to expand next. Tree of Thoughts (ToT) (Yao et al., 2023) is one instance of this pattern: the LLM scores candidate "thoughts" branching from a node while the controller maintains the search tree. One feature of this paradigm is that the LLM is queried on a *local* view (the current path) rather than on the entire search trace. Note that ToT typically conditions on the full path from the root to the current node, whereas our heuristic conditions only on the leaf state and the goal (examples of per-domain inputs are listed in Table 1). This is fine in our setting since all three environments are Markovian, and thus the current state and goal are sufficient for the heuristic to evaluate candidate actions with respect to future progress toward the goal.

LLM heuristic-guided tree search. We run best-first search (BFS) guided by the LLM heuristic. At each expansion step, for each (state, action) candidate on the frontier of leaf nodes, we query the LLM heuristic with the current state, the candidate action, and the goal, and obtain a heuristic value via an additional MLP head. The best-first algorithm picks the candidate $(s^\star, a^\star)$ with the lowest heuristic value, applies the action to obtain the resulting state, and adds it as a child of $s^\star$ in the search tree. This continues until a solution is found or the search budget is exhausted. The full procedure is given in Algorithm 1. We set the search budget B to 500 expansions during RL training and 200 during evaluation. The original ToT is a training-free inference-time method, but for a fair comparison with the trace-conditioned reasoning policy of Section 4.1, we train the LLM heuristic with an analogous two-stage pipeline.

Supervised fine-tuning. The LLM backbone with its MLP head is trained on (state, action) pairs collected from the SFT traces, with the rule-based heuristic value as the target. This mirrors Stage 1 in Section 4.1 and initializes the heuristic for subsequent reinforcement learning.

Table 1: Per-domain training data for the local-state LLM heuristic. Each training pair consists of a (state, action) input collected from the SFT search traces and a target value computed by the rule-based heuristic.

Domain	# SFT Pairs	Target heuristic	Example input
Blocks World	1.74M	Negated number of "good" blocks	goal `S{0:D<C<B<A}`, state `S{0:A<B; 1:C; 2:D}`, action `B:A->TABLE`
Navigation	1.22M	Manhattan distance to goal	{grid}, goal `(7,8)`, state `P=(3,2)`, action `R`
Sokoban	1.28M	Min-cost box-to-goal matching	{grid}, goal `{(7,7),(1,1)}`, state `P=(2,3) B={(4,5),(6,2)}`, action `U`

Table 2: Policies with versus without access to the search trace, evaluated on the three reasoning domains with Qwen3-0.6B. Lower is better for expansions. Subscripts show standard errors.

	Blocks World		Navigation		Sokoban	
Method	Success ↑	Expansions ↓	Success ↑	Expansions ↓	Success ↑	Expansions ↓
BFS (pretrained)	$99.7_{\pm 0.17}$	$9.54_{\pm 0.31}$	$100.0_{\pm 0.00}$	$20.27_{\pm 0.42}$	$94.9_{\pm 0.70}$	$73.21_{\pm 2.11}$
BFS (RL)	$99.8_{\pm 0.14}$	$8.56_{\pm 0.18}$	$100.0_{\pm 0.00}$	$16.99_{\pm 0.37}$	$99.1_{\pm 0.30}$	$64.08_{\pm 1.86}$
SFT-implicit	$90.0_{\pm 0.95}$	$9.12_{\pm 0.16}$	$90.6_{\pm 0.92}$	$15.39_{\pm 0.18}$	$74.8_{\pm 1.37}$	$68.24_{\pm 7.70}$
GRPO-implicit	$97.3_{\pm 0.51}$	$8.25_{\pm 0.14}$	$94.9_{\pm 0.70}$	$14.80_{\pm 0.16}$	$85.9_{\pm 1.10}$	$63.54_{\pm 4.12}$

Reinforcement learning. We formulate the tree expansion process of selecting which candidate to expand as a decision-making problem (Mittal et al., 2025), guided by the LLM heuristic. We can then optimize the LLM heuristic with reinforcement learning, using the same reward (Eq. 1) used for the trace-conditioned reasoning policy. Since the search uses the oracle transition function and goal verifier, every trace is valid; therefore, the correctness term is zero only when the search fails to find a goal within the budget. We use the same advantage calculation as in GRPO. For each instance, we run search N times, compute reward r_i for each trace, and use the standardized advantage

$$A_i = \frac{r_i - \mu}{\sigma + \epsilon}, \tag{2}$$

where μ and σ are the mean and standard deviation of the rewards across the N traces. The resulting policy gradient is averaged over the N traces sampled from the tree expansion policy π_θ induced by the LLM heuristic:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} A_i \sum_{t=0}^{T_i - 1} \nabla_\theta \log \pi_\theta(c_{i,t} \mid \mathcal{C}_{i,t}), \tag{3}$$

where $\mathcal{C}_{i,t}$ is the set of candidate expansions on the frontier at step t of trace i, $c_{i,t} \in \mathcal{C}_{i,t}$ is the candidate sampled and expanded at that step, T_i is the expansion count in trace i, and the per-step distribution is the softmax over negated LLM heuristic scores, since lower heuristics are favored,

$$\pi_\theta(c \mid \mathcal{C}_{i,t}) = \frac{\exp(-h_\theta(c))}{\sum_{c' \in \mathcal{C}_{i,t}} \exp(-h_\theta(c'))}. \tag{4}$$

We train the LLM heuristic for 1 epoch of SFT and then 1 epoch of RL, using the same SFT and RL training instances as the reasoning policy.

4.3 Results

We measure search efficiency by the number of expansions per instance, which is proportional to the number of tokens in the search trace. Table 2 shows that the implicit reasoning model achieves lower success rates than BFS (RL), while offering only modest improvements in search expansions (Appendix C shows the same for Qwen3.5-4B). Although the implicit reasoning policy conditions on the whole trace, it is not reliably stronger than the local-state heuristic baseline. GRPO improves the implicit policy over its SFT initialization, but this alone still does not establish a clear advantage over local-state heuristic-guided search.

One explanation is that the available trace is simply a flat serialization of a search tree. The model sees the history, but the tree structure is only latent in the serialization. This leads to the next question: would making the latent tree structure explicit allow the model to better use the available history and therefore outperform local-state heuristics? We investigate this question in the next section by comparing implicit traces against a more structured trace representation that explicitly labels the parent-child relationships in the search tree.

5 Structured Search Traces

In this section, we investigate whether making the tree structure that is latent in the search trace explicit allows the policy to make better use of the trace. To do that, we train a second policy on the same underlying search traces but with explicit parent-pointer annotations that expose the tree structure. We then compare this explicit-trace policy against the implicit-trace policy of Section 4.1 and the local-state heuristic baseline of Section 4.

5.1 Trace representations

To make the search trace more useful, we introduce a modification to make it a more structured representation, where each expansion identifies the frontier state being expanded and the child state reached by the chosen action. We refer to this source-state identifier as a *parent pointer*, because it tells the model which earlier search state the new expansion descends from. In contrast, the implicit representation in Section 4.1 records only the action and resulting state, leaving the resumed search state to be inferred from context. The two representations describe the same search behavior and differ only in whether the parent of each expansion is written down: the implicit format leaves it to be inferred from the preceding states, while the explicit format lets the policy condition on it directly.

An example of the structured representation is shown below:

```
EXPAND sid=0 ACT {action} -> sid=1 {resulting state}
EXPAND sid=1 ACT {action} -> sid=2 {resulting state}
EXPAND sid=1 ACT {action} -> sid=3 {resulting state}
```

Full per-domain SFT examples for both the implicit and explicit formats are provided in Appendix D, and a visualization for each domain is shown in Appendix Figure 1.

The explicit policy is trained on the same instances using the SFT+GRPO pipeline as the implicit policy of Section 4.1, with the base model and training parameters held fixed. Comparing the two policies trained on the same underlying searches but with different trace representations isolates the effect of making the latent tree structure explicit.

5.2 Effect of structured trace annotations

We first compare the effects of the two different trace representations by comparing four policies in this pipeline: SFT-implicit, SFT-explicit, GRPO-implicit, and GRPO-explicit. At the SFT stage, explicit annotation raises average solve rate by 5.0 points across domains, driven mainly by gains in Navigation (95.2 vs. 90.6) and Sokoban (85.6 vs. 74.8), while Blocks World is essentially unchanged (89.6 vs. 90.0). The effect on search efficiency before RL is weaker: the explicit model uses nearly the same number of expansions in Blocks World (9.06 vs. 9.12), but slightly more in Navigation and Sokoban. This suggests that explicit traces primarily improve the model's ability to recover valid search behavior during imitation.

We then initialize GRPO from the SFT checkpoints. After GRPO training, the explicit model consistently outperforms the implicit model. GRPO-explicit reaches 100.0% solve rate in both Blocks World and Navigation and 89.6% in Sokoban, compared with 97.3%, 94.9%, and 85.9% for GRPO-implicit. It is also more search-efficient in every domain, reducing expansions from 8.25 to 7.31 in Blocks World, from 14.80 to 14.28 in Navigation, and from 63.54 to 52.82 in Sokoban. In Sokoban, the explicit model's solve rate remains below that of the local-state heuristic. This is because the transition function for Sokoban is harder to learn as it involves both player and box movements. The local heuristic search, however, has access to the transition function. To focus on the effect of search history access, we also evaluate the explicit model with the environment supplying the

Table 3: Implicit versus explicit trace annotations on controlled reasoning domains with Qwen3-0.6B. Lower is better for expansions. Subscripts show standard errors computed over evaluation instances.

Blocks World			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$90.0_{\pm 0.95}$	$9.12_{\pm 0.16}$
With parent pointer	SFT	$89.6_{\pm 0.97}$	$9.06_{\pm 0.45}$
No parent pointer	GRPO	$97.3_{\pm 0.51}$	$8.25_{\pm 0.14}$
With parent pointer	GRPO	$100.0_{\pm 0.00}$	$7.31_{\pm 0.08}$
–	BFS (pretrained)	$99.7_{\pm 0.17}$	$9.54_{\pm 0.31}$
–	BFS (RL)	$99.8_{\pm 0.14}$	$8.56_{\pm 0.18}$
Sokoban			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$74.8_{\pm 1.37}$	$68.24_{\pm 7.70}$
With parent pointer	SFT	$85.6_{\pm 1.11}$	$73.26_{\pm 5.88}$
No parent pointer	GRPO	$85.9_{\pm 1.10}$	$63.54_{\pm 4.12}$
With parent pointer	GRPO	$89.6_{\pm 0.97}$	$52.82_{\pm 3.35}$
With parent pointer	GRPO + generation constraint	$98.9_{\pm 0.33}$	$54.70_{\pm 4.62}$
–	BFS (pretrained)	$94.9_{\pm 0.70}$	$73.21_{\pm 2.11}$
–	BFS (RL)	$99.1_{\pm 0.30}$	$64.08_{\pm 1.86}$
Navigation			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$90.6_{\pm 0.92}$	$15.39_{\pm 0.18}$
With parent pointer	SFT	$95.2_{\pm 0.68}$	$15.63_{\pm 0.18}$
No parent pointer	GRPO	$94.9_{\pm 0.70}$	$14.80_{\pm 0.16}$
With parent pointer	GRPO	$100.0_{\pm 0.00}$	$14.28_{\pm 0.12}$
–	BFS (pretrained)	$100.0_{\pm 0.00}$	$20.27_{\pm 0.42}$
–	BFS (RL)	$100.0_{\pm 0.00}$	$16.99_{\pm 0.37}$

resulting state of each expansion, implemented as a generation constraint. With the same access as the baseline, the explicit model reaches 98.9% in Sokoban, matching the heuristic's 99.1% while using fewer expansions (54.70 vs. 64.08). This constrained evaluation is possible only because the explicit format names the state each action is applied to; the implicit format leaves that state unspecified.

Overall, explicit annotation improves solve rate at the SFT stage and improves both solve rate and search efficiency over the implicit representation after GRPO; against the local-state heuristic, it uses fewer expansions in every domain and matches the heuristic's solve rate when given the same access to the environment. This pattern suggests that making the search tree structure explicit not only makes demonstrated search behavior easier to learn during SFT, but also provides a stronger starting point for subsequent policy optimization. A likely reason is that explicit parent pointers help the model track the search tree and the history of visited states more reliably (Table 4), which in turn enables more effective backtracking and search decisions during RL. Taken together with the comparison in Section 4, these results suggest that the benefits of search trace access are most fully realized when the tree structure latent in the trace is made explicit. The gain comes from how the trace is represented, which suggests that making latent information explicit is a simple way to improve trace-conditioned search policies. The same pattern holds for Qwen3.5-4B (Appendix C).

5.3 Analysis of Different Policies

Explicit structure reduces both search and extraction errors. Table 4 breaks failures into search-side and extraction-side errors. Replaying every trace with the domain transition function, we count (i) traces that announce `GOAL_REACHED` on a node with no chain of legal transitions back to the root, (ii) traces containing at least one expansion that violates the tree, a child whose parent was never expanded, or an illegal transition from the named parent, and (iii) traces that found a goal but emitted the wrong plan. For the implicit policy, which names no parent, we recover the parent from previously expanded states for which the recorded action produces the recorded resulting state under

Table 4: Failure analysis after GRPO (Qwen3-0.6B, values in percent).

		Search		Extraction
Domain	Policy	Disc. goal	Tree viol.	fail
Blocks World	GRPO-implicit	0.2	1.7	0.3
	GRPO-explicit	**0.0**	**0.0**	**0.0**
Navigation	GRPO-implicit	0.5	2.8	4.6
	GRPO-explicit	**0.0**	**0.0**	**0.0**
Sokoban	GRPO-implicit	3.4	11.2	5.9
	GRPO-explicit	**1.0**	**5.7**	**1.1**

Table 5: Average pairwise distance $\bar{d}$ of visited states on Navigation (Qwen3-0.6B). Higher values indicate more diverse exploration.

Method	$\bar{d}\uparrow$
BFS (RL)	3.99
GRPO-implicit	4.11
GRPO-explicit	4.19

the transition function, choosing the shallowest when several match. A trace can fall into more than one category, and traces that never announce a goal are not shown. The explicit policy makes fewer errors of every kind in all three domains, suggesting that explicit tree annotation may make the plan easier to read off and help keep the model's expansions attached to the tree it has actually built.

Explicit trace-conditioned LLM explores state space more effectively. We study the search behavior of different policies by looking at the distribution of visited states during search in the navigation domain, where we define the distance between two states as the shortest-path distance between them on the grid. Given a search tree, we measure the diversity of the set of visited states S by the average pairwise distance (Rao, 1982),

$$\bar{d}(S) = \frac{1}{|S|^2} \sum_{s,s' \in S} d(s, s'), \tag{5}$$

where $d(s, s')$ is the shortest-path distance between s and s'. Higher values indicate that the visited states are spread further apart on the grid.

Table 5 reports the average $\bar{d}$ across navigation instances. The explicit policy attains the highest value (4.19), followed by the implicit policy (4.11) and the local-state heuristic (3.99). A higher $\bar{d}$ means the visited states are spread further apart on the grid, while a lower value indicates that the policy concentrates its expansions in a small neighborhood. The trace-conditioned policies, especially the explicit policy, thus cover more of the state space in their search. A plausible explanation is that conditioning on the search trace lets the policy see which neighborhoods earlier expansions already covered and steer subsequent expansions elsewhere, whereas the local-state heuristic has no memory of past expansions to guide such avoidance; explicit parent pointers further amplify this effect by making the explored region of the tree easier to track.

6 Conclusion

We studied LLMs that carry out search by generating a search trace, asking whether access to the full trace makes such a policy stronger than a local-state LLM heuristic. Across Blocks World, Navigation, and Sokoban, trace access alone does not reliably yield better search: when the tree structure is left latent in the trace, the model often fails to exploit the additional history. Adding parent pointers that identify which search state each expansion descends from improves both solve rate and search efficiency over the implicit representation, and matches the solve rate of the local-state heuristic with fewer expansions. Explicit structure also helps the model recover final plans from its own traces and explore the state space more widely. Training these policies with RL relies on a reward that orders traces lexicographically, with correctness taking priority over search length; rewards without this property were unstable to train.

The benefit of trace-conditioned search thus depends not only on whether the model sees the search history, but on how that history is represented. The parent pointer is one instance of a more general choice: an intermediate quantity of the reasoning process can be left for the model to infer or written into the trace, and writing it down can make the learning problem easier. In settings where the model reasons in a representation designed for the problem, as in the systems discussed in Section 1, such choices are available to the designer and deserve the same attention as training objectives and model scale.

AI USE STATEMENT

In this work, we used generative AI tools to help with grammar and style in writing the paper. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We did not collect any new human-subject data or process personally identifiable information. No potential harms, such as bias amplification, toxic outputs, or dual-use risks, were observed in this work.

REPRODUCIBILITY STATEMENT

We are committed to reproducibility. All details of the data used and generated are documented in the paper, along with comprehensive descriptions of our experimental setup and hyperparameters.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, Timothy Lillicrap, and Victor Valdes. An investigation of model-free planning: boxoban levels. https://github.com/deepmind/boxoban-levels/, 2018.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, 2023.

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

Andreas Junghanns and Jonathan Schaeffer. Sokoban: A challenging single-agent search problem. In *IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*, pp. 27–36. Morgan Kaufmann Publishers San Francisco, CA, 1997.

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *arXiv preprint arXiv:2402.14083*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Dixant Mittal, Liwei Kang, and Wee Sun Lee. Learning to search from demonstration sequences. In *The Thirteenth International Conference on Learning Representations*, 2025.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. 2021.

Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1983.

C Radhakrishna Rao. Diversity and dissimilarity coefficients: a unified approach. *Theoretical population biology*, 21(1):24–43, 1982.

Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

Bilgehan Sel, Ahmad Al-Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. *arXiv preprint arXiv:2308.10379*, 2023.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Chee Heng Tan, Zhuoyi Lin, Mehul Motani, and Wee Sun Lee. On the effectiveness of reward functions in reinforcement learning for confidence calibration of large language models. *arXiv preprint arXiv:2607.04332*, 2026.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

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

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Xie. Self-evaluation guided beam search for reasoning. *Advances in Neural Information Processing Systems*, 36:41618–41650, 2023.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A Environment Details

For each domain, we generate 20k instances for SFT training, a separate 20k instances for RL, and 1k for evaluation. SFT traces are generated by running best-first search with the rule-based heuristic described below, and instances whose SFT trace exceeds 16k tokens are filtered out.

Blocks World. The agent must transform an initial configuration of stacked blocks into a target stacking by moving a block with nothing on top of it. We generate instances with three stacks and between 4 and 10 blocks, by randomly sampling initial and goal states and filtering out instances that can be solved in fewer than 4 steps. The ratio of instances with different numbers of blocks in the training and testing datasets is weighted by the square of the number of blocks. The state is serialized as an ordered list of stacks, e.g., "`S{ 0:A<B ; 1:C ; 2:D }`", where each stack is written from bottom to top. An action moves one clear block from its current support to another stack or to the table, written as "`B:A->TABLE`" (moving B from the top of A to TABLE). The rule-based heuristic is the negated number of "good" blocks in a state, where a block is considered "good" if it is already on its goal support and every block beneath it is also "good."

Grid Navigation. The agent moves on a 10×10 2D grid from a randomly selected starting cell to a randomly selected goal cell while avoiding obstacles. Each cell is an obstacle with probability 35%. We generate instances by randomly sampling obstacles in the map and retain only instances whose optimal path length is at least 6 and whose number of best-first expansions is at least 1.3 times the resulting plan length. When using the LLM reasoning model, the full grid is given in the input prompt, and during search the state records only the agent location, written as "`A=(r,c)`". The action space is the four cardinal moves `U`, `D`, `L`, and `R`. The rule-based heuristic is the Manhattan distance to the goal. This heuristic can be misleading in this domain because minimizing Manhattan distance can easily lead to dead ends, leaving room for the model to learn a stronger search heuristic.

Sokoban. The player navigates a 2D grid and pushes boxes onto goal squares. We use the unfiltered split from Boxoban levels (Guez et al., 2018) for training and testing. The original Boxoban instances have four boxes each, for which the search can have more than 5000 expansions. To make search manageable for the LLM, we randomly remove two boxes and two goal positions in each instance. The full grid configuration, including the initial box, wall, and goal locations, is given in the prompt. During search, the state is represented by the player position together with the set of box positions, written as "`P=(r,c) B={(r,c),...}`". The action space is again `U`, `D`, `L`, and `R`, which moves the player. The rule-based heuristic (Junghanns & Schaeffer, 1997) first estimates, for each box-goal pair, the number of moves required to push the box to the goal while ignoring the other boxes, and then uses the minimum bipartite matching cost between boxes and goals as the heuristic value.

B Training Details

The trace-conditioned reasoning policy uses Qwen3-0.6B (Yang et al., 2025) or Qwen3.5-4B as the backbone, with the same hyperparameters for both. SFT is run for 4 epochs with learning rate 1e-5. GRPO samples $N = 8$ rollouts per instance for the advantage estimate, and is run for 1 epoch with learning rate 1e-5, with $\lambda = 0.005$ and $\gamma = 0.99$ in Eq. 1. The local-state LLM heuristic uses the same backbone with an additional MLP head, and is trained for 1 epoch of SFT and 1 epoch of RL on the same SFT and RL instances, with a search budget of 500 expansions during RL training and 200 during evaluation. All training runs in this paper are performed on one NVIDIA H100 GPU.

C Results with a Larger Base Model

Table 6 repeats the comparisons of Tables 2 and 3 with Qwen3.5-4B as the base model for both the trace-conditioned policies and the local-state LLM heuristic, using the same data, training pipeline, and hyperparameters as in Section 4. The pattern matches the Qwen3-0.6B results. At the SFT stage, explicit annotation raises solve rate in all three domains, with the largest gain in Sokoban (88.3 vs. 79.7), while search efficiency is unchanged or worse. After GRPO, the explicit model outperforms the implicit model on both solve rate and expansions in every domain, and uses fewer expansions than BFS (RL) in every domain. As with the smaller model, the explicit model's solve rate in Sokoban is

below that of BFS (RL) under free-form generation (92.4 vs. 99.4) and matches it once the generation constraint is applied (99.5).

Table 6: Implicit versus explicit trace annotations with Qwen3.5-4B as the base model. Lower is better for expansions. Subscripts show standard errors computed over evaluation instances.

Blocks World			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$98.2_{\pm 0.42}$	$9.67_{\pm 0.29}$
With parent pointer	SFT	$99.3_{\pm 0.26}$	$11.26_{\pm 0.62}$
No parent pointer	GRPO	$98.3_{\pm 0.41}$	$8.24_{\pm 0.16}$
With parent pointer	GRPO	$100.0_{\pm 0.00}$	$7.27_{\pm 0.10}$
–	BFS (pretrained)	$100.0_{\pm 0.00}$	$9.62_{\pm 0.32}$
–	BFS (RL)	$100.0_{\pm 0.00}$	$9.01_{\pm 0.22}$
Sokoban			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$79.7_{\pm 1.27}$	$61.21_{\pm 1.38}$
With parent pointer	SFT	$88.3_{\pm 1.00}$	$62.54_{\pm 1.37}$
No parent pointer	GRPO	$86.3_{\pm 1.09}$	$58.12_{\pm 1.26}$
With parent pointer	GRPO	$92.4_{\pm 0.84}$	$51.04_{\pm 1.60}$
With parent pointer	GRPO + generation constraint	$99.5_{\pm 0.22}$	$53.43_{\pm 1.53}$
–	BFS (pretrained)	$95.0_{\pm 0.69}$	$73.30_{\pm 1.40}$
–	BFS (RL)	$99.4_{\pm 0.24}$	$63.20_{\pm 1.04}$
Navigation			
Method	Training	Success ↑	Expansions ↓
No parent pointer	SFT	$98.7_{\pm 0.36}$	$15.09_{\pm 0.18}$
With parent pointer	SFT	$98.9_{\pm 0.33}$	$15.12_{\pm 0.18}$
No parent pointer	GRPO	$98.8_{\pm 0.34}$	$14.46_{\pm 0.17}$
With parent pointer	GRPO	$99.6_{\pm 0.20}$	$13.94_{\pm 0.18}$
–	BFS (pretrained)	$100.0_{\pm 0.00}$	$20.03_{\pm 0.41}$
–	BFS (RL)	$100.0_{\pm 0.00}$	$16.94_{\pm 0.37}$

D SFT Trace Examples

We show one short instance for each domain and trace format, drawn from the training split. Within each domain, the implicit and explicit examples describe the same underlying instance and search but use different trace serializations.

Note that the Navigation traces explicitly include `BLOCKED` entries for actions that walk into walls, rather than silently pruning them. We found that without these entries, both the implicit and explicit models fail to learn a valid search policy during SFT. A plausible explanation is that the Manhattan-distance heuristic gives a strong directional preference over actions, so the model expects the highest-scoring candidate to be expanded next; if blocked candidates are silently removed from the trace, the visible expansion order no longer matches the heuristic's ranking, which may make the learning harder.

D.1 Blocks World

Implicit format

```
INSTANCE_BEGIN
BLOCKS A,B,C,D
INIT S{ 0:B<A<C ; 1:D }
GOAL S{ 0:B<A ; 1:D<C }
INSTANCE_END
SEARCH_TRACE_BEGIN
```

```
EXPAND S{ 0:B<A<C ; 1:D }
EXPAND ACT C:A->D -> S{ 0:B<A ; 1:D<C }
GOAL_REACHED
PLAN_LEN 1
PLAN_STEP 1 C:A->D
SEARCH_TRACE_END
```

Explicit format

```
INSTANCE_BEGIN
BLOCKS A,B,C,D
INIT S{ 0:B<A<C ; 1:D }
GOAL S{ 0:B<A ; 1:D<C }
INSTANCE_END
SEARCH_TRACE_BEGIN
EXPAND sid=0 S{ 0:B<A<C ; 1:D }
EXPAND sid=0 ACT C:A->D -> sid=1 S{ 0:B<A ; 1:D<C }
GOAL_REACHED
PLAN_LEN 1
PLAN_STEP 1 C:A->D
SEARCH_TRACE_END
```

D.2 NAVIGATION

Implicit format

```
INSTANCE_BEGIN
GRID_SIZE 10
MAP
# # # # # # # # # #
# . . # # . # . . #
# . # . # . . . . #
# # # . . # . . . #
# # . . . . . # . #
# . G . . # . . . #
# . . . . . . . # #
# . # . . . S . # #
# . . . . . . # . #
# # # # # # # # # #
START (7,6)
GOAL (5,2)
INSTANCE_END
SEARCH_TRACE_BEGIN
EXPAND A=(7,6)
EXPAND ACT U -> A=(6,6)
EXPAND ACT U -> A=(5,6)
BLOCKED ACT L -> (5,5) WALL
EXPAND ACT L -> A=(6,5)
BLOCKED ACT U -> (5,5) WALL
EXPAND ACT L -> A=(6,4)
EXPAND ACT U -> A=(5,4)
BLOCKED ACT R -> (5,5) WALL
EXPAND ACT L -> A=(5,3)
EXPAND ACT L -> A=(5,2)
GOAL_REACHED
PLAN_LEN 6
PLAN_STEP 1 U
PLAN_STEP 2 L
PLAN_STEP 3 L
```

```
PLAN_STEP 4 U
PLAN_STEP 5 L
PLAN_STEP 6 L
SEARCH_TRACE_END
```

Explicit format

```
INSTANCE_BEGIN
GRID_SIZE 10
MAP
# # # # # # # # # #
# . . # # . # . . #
# . # . # . . . . #
# # # . . # . . . #
# # . . . . . # . #
# . G . . # . . . #
# . . . . . . . # #
# . # . . . S . # #
# . . . . . . . # #
# # # # # # # # # #
START (7,6)
GOAL (5,2)
INSTANCE_END
SEARCH_TRACE_BEGIN
EXPAND sid=0 A=(7,6)
EXPAND sid=0 ACT U -> sid=1 A=(6,6)
EXPAND sid=1 ACT U -> sid=2 A=(5,6)
BLOCKED ACT L -> (5,5) WALL
EXPAND sid=1 ACT L -> sid=3 A=(6,5)
BLOCKED ACT U -> (5,5) WALL
EXPAND sid=3 ACT L -> sid=4 A=(6,4)
EXPAND sid=4 ACT U -> sid=5 A=(5,4)
BLOCKED ACT R -> (5,5) WALL
EXPAND sid=5 ACT L -> sid=6 A=(5,3)
EXPAND sid=6 ACT L -> sid=7 A=(5,2)
GOAL_REACHED
PLAN_LEN 6
PLAN_STEP 1 U
PLAN_STEP 2 L
PLAN_STEP 3 L
PLAN_STEP 4 U
PLAN_STEP 5 L
PLAN_STEP 6 L
SEARCH_TRACE_END
```

D.3 SOKOBAN

The map is rendered with `#` walls, `.` floor, `G` goal, `B` box, and `P` player. The two examples below describe the same instance and the same underlying search (plan length 5, 13 expansions).

Implicit format

```
INSTANCE_BEGIN
SIZE 10 10
BOARD
# # # # # # # # # #
# . . . . . . . . #
# . . . . . B G . #
# . . . B P . . . #
# . # # G . . # # #
```

```
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
GOALS {(2,7),(4,4)}
INIT P=(3,5) B={(2,6),(3,4)}
INSTANCE_END
SEARCH_TRACE_BEGIN
EXPAND P=(3,5) B={(2,6),(3,4)}
EXPAND ACT U -> P=(2,5) B={(2,6),(3,4)}
EXPAND ACT R -> P=(2,6) B={(2,7),(3,4)}
EXPAND ACT U -> P=(1,6) B={(2,7),(3,4)}
EXPAND ACT D -> P=(3,6) B={(2,7),(3,4)}
EXPAND ACT L -> P=(2,5) B={(2,7),(3,4)}
EXPAND ACT L -> P=(1,5) B={(2,7),(3,4)}
EXPAND ACT R -> P=(1,7) B={(2,7),(3,4)}
EXPAND ACT D -> P=(4,6) B={(2,7),(3,4)}
EXPAND ACT L -> P=(3,5) B={(2,7),(3,4)}
EXPAND ACT R -> P=(3,7) B={(2,7),(3,4)}
EXPAND ACT L -> P=(2,4) B={(2,7),(3,4)}
EXPAND ACT D -> P=(3,4) B={(2,7),(4,4)}
GOAL_REACHED
PLAN_LEN 5
PLAN_STEP 1 U
PLAN_STEP 2 R
PLAN_STEP 3 L
PLAN_STEP 4 L
PLAN_STEP 5 D
SEARCH_TRACE_END
```

Explicit format

```
INSTANCE_BEGIN
SIZE 10 10
BOARD
# # # # # # # # # #
# . . . . . . . . #
# . . . . . B G . #
# . . . B P . . . #
# . # # G . . # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
GOALS {(2,7),(4,4)}
INIT P=(3,5) B={(2,6),(3,4)}
INSTANCE_END
SEARCH_TRACE_BEGIN
EXPAND sid=0 P=(3,5) B={(2,6),(3,4)}
EXPAND sid=0 ACT U -> sid=1 P=(2,5) B={(2,6),(3,4)}
EXPAND sid=1 ACT R -> sid=2 P=(2,6) B={(2,7),(3,4)}
EXPAND sid=2 ACT U -> sid=3 P=(1,6) B={(2,7),(3,4)}
EXPAND sid=2 ACT D -> sid=4 P=(3,6) B={(2,7),(3,4)}
EXPAND sid=2 ACT L -> sid=5 P=(2,5) B={(2,7),(3,4)}
EXPAND sid=3 ACT L -> sid=6 P=(1,5) B={(2,7),(3,4)}
EXPAND sid=3 ACT R -> sid=7 P=(1,7) B={(2,7),(3,4)}
EXPAND sid=4 ACT D -> sid=8 P=(4,6) B={(2,7),(3,4)}
```

```
EXPAND sid=4 ACT L -> sid=9 P=(3,5) B={(2,7),(3,4)}
EXPAND sid=4 ACT R -> sid=10 P=(3,7) B={(2,7),(3,4)}
EXPAND sid=5 ACT L -> sid=11 P=(2,4) B={(2,7),(3,4)}
EXPAND sid=11 ACT D -> sid=12 P=(3,4) B={(2,7),(4,4)}
GOAL_REACHED
PLAN_LEN 5
PLAN_STEP 1 U
PLAN_STEP 2 R
PLAN_STEP 3 L
PLAN_STEP 4 L
PLAN_STEP 5 D
SEARCH_TRACE_END
```

E Proof of Proposition 1

Proposition 1 (restated). *Let $0 < \gamma < 1$ and $0 < \lambda < 1 - \gamma$. Then the reward in Eq. 1 satisfies the lexicographic ordering property for correctness and number of expansions: for any two traces τ and τ', (i) if τ is valid and correct and τ' is not, then $R(\tau) > R(\tau')$; (ii) if both are valid and correct and $N_{\text{exp}}(\tau) < N_{\text{exp}}(\tau')$, then $R(\tau) > R(\tau')$.*

Proof. Case (i) is the first level of the order, correct above incorrect, and case (ii) is the second level, shorter above longer among correct traces. Let $S(N) = \sum_{t=0}^{N-1} \gamma^t = (1 - \gamma^N)/(1 - \gamma)$, which is strictly increasing in N and bounded above by $1/(1 - \gamma)$. For a valid and correct trace, $R(\tau) = 1 - \lambda S(N_{\text{exp}}(\tau)) > 1 - \lambda/(1 - \gamma) > 0 = R(\tau')$ for any τ' that is not valid and correct, which gives (i). For (ii), both traces are valid and correct, so $R(\tau) - R(\tau') = \lambda \left(S(N_{\text{exp}}(\tau')) - S(N_{\text{exp}}(\tau))\right) > 0$ since S is strictly increasing. □

F Ordering over Policies

Proposition 1 concerns individual traces. A non-deterministic policy is optimized through its expected reward, and a scalar expectation can trade success probability against search length. Write $c = \lambda/(1 - \gamma)$, so that a valid and correct trace with N expansions scores $R(N) = (1 - c) + c\,\gamma^N \in (1 - c, 1]$ and an incorrect trace scores 0. Consider a policy A that succeeds with probability p_A using N_A expansions and a policy B that succeeds with probability p_B using N_B expansions. B has higher expected reward than A when $p_B R(N_B) > p_A R(N_A)$, that is, when $p_B/p_A > R(N_A)/R(N_B)$. The right-hand side is at least $1 - c$, so the trade is bounded: B can be preferred only if its success probability is at least a $1 - c$ fraction of A's, and in practice only if it also searches much less. The constant c therefore balances the two incentives. A smaller λ brings $1 - c$ closer to 1, so that almost no success probability can be traded away, at the cost of a weaker efficiency signal; a larger λ strengthens the efficiency signal but widens the room for such a trade.

G Alternative Reward Designs

Here we discuss alternative reward designs we considered, in terms of the lexicographic ordering property of Section 4.1. The natural first choice, a constant per-step penalty, violates the property; the geometric decay satisfies it but with a margin between correct and incorrect traces that shrinks to zero. Both were unstable in our experiments, which motivated the bounded discounted form of Eq. 1.

Constant per-step penalty. The simplest way to reward shorter searches is to subtract a constant penalty per expansion from the correctness reward,

$$R_{\text{const}}(\tau) = \mathbf{1}[\text{valid}(\tau) \wedge \text{correct}(\tau)]\,(1 - \alpha\, N_{\text{exp}}(\tau))\,, \tag{6}$$

where $\alpha > 0$ is the per-step penalty. This reward does not satisfy the lexicographic ordering property: a correct trace with more than $1/\alpha$ expansions scores below zero, and therefore below every incorrect trace, so a sufficiently long search is penalized more for succeeding than it would be for failing. Capping the reward at zero,

$$R_{\text{cap}}(\tau) = \mathbf{1}[\text{valid}(\tau) \wedge \text{correct}(\tau)]\ \max(1 - \alpha\, N_{\text{exp}}(\tau),\, 0)\,, \tag{7}$$

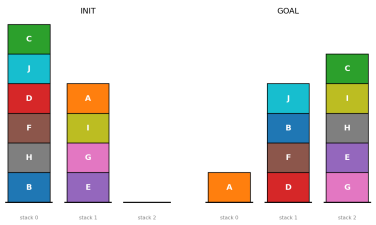
removes the negative rewards but still violates the property: once $N_{\text{exp}}(\tau) \geq 1/\alpha$ the reward saturates at zero, so a long correct trace ties with every incorrect trace and correct traces beyond the cap are no longer distinguished by length. Both versions are sensitive to the choice of α, since $1/\alpha$ sets the length beyond which the ordering breaks. While tuning α can mitigate this issue, the training was still unstable and tended to collapse.

Geometric decay applied to the correctness reward. The second alternative multiplies the correctness reward by a geometric decay factor:

$$R_{\text{decay}}(\tau) = \mathbf{1}[\text{valid}(\tau) \wedge \text{correct}(\tau)]\, \gamma^{N_{\text{exp}}(\tau)}, \tag{8}$$

which satisfies the lexicographic ordering property, since $\gamma^N > 0$ for every N and is decreasing in N. However, the margin between a correct trace and an incorrect one, γ^N, shrinks to zero as the search grows, whereas Eq. 1 keeps it above $1 - \lambda/(1-\gamma)$. This reward still suffered from unstable training in our experiments.

H VISUALIZATION OF REASONING ENVIRONMENTS



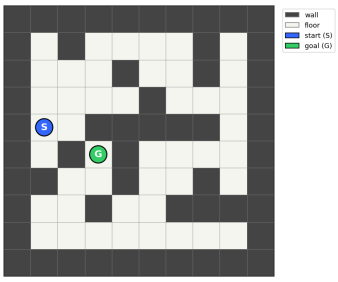
Prompt

```
INSTANCE_BEGIN
BLOCKS A,B,C,D,E,F,G,H,I,J
INIT S{ 0:B<H<F<D<J<C ; 1:E<G<I<A }
GOAL S{ 0:A ; 1:D<F<B<J ; 2:G<E<H<I<C
}
INSTANCE_END
SEARCH_TRACE_BEGIN
```

State representation in search

```
"S{ 0:B<H<F<D<J<C ; 1:E<G<I<A }"
```

(a) Blocks World



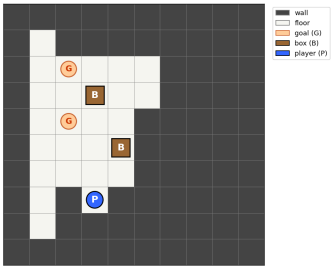
Prompt

```
INSTANCE_BEGIN
GRID_SIZE 10
MAP
# # # # # # # # # #
# . # . . . . # . #
# . . . # . . # . #
# . . . . # . . . #
# S . # # # # # . #
# . # G # . . . . #
# # . . # . . # . #
# . . # . . # # # #
# . . . . . . . . #
# # # # # # # # # #
START (4,1)
GOAL (5,3)
INSTANCE_END
SEARCH_TRACE_BEGIN
```

State representation in search

```
"A=(4,1)"
```

(b) Navigation



Prompt

```
INSTANCE_BEGIN
SIZE 10 10
BOARD
+ + + + + + + + + +
+ . + + + + + + + +
+ . O . . . + + + +
+ . . . . . + + + +
+ . O . . + + + + +
+ . . . . + + + + +
+ . . . . + + + + +
+ . + . + + + + + +
+ . + + + + + + + +
+ + + + + + + + + +
GOALS {(2,2),(4,2)}
INIT P=(7,3) B={(3,3),(5,4)}
INSTANCE_END
SEARCH_TRACE_BEGIN
```

State representation in search

```
"P=(7,3) B={(3,3),(5,4)}"
```

(c) Sokoban

Figure 1: Examples of the three domains used in our experiments. Each subfigure shows a visualization of one problem instance on the left and the corresponding textual serialization on the right.