ITERATIVE DEPLOYMENT IMPROVES PLANNING SKILLS IN LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier LLMs have become much better at end-to-end planning across recent model generations, but their training pipelines are not public, so the exact cause of this progress is unknown. Earlier attempts to improve planning skills of LLMs relied on planning-specific supervision, such as optimal plans from a classical planner, carefully shaped rewards, or structured validator feedback. We test whether a much weaker signal suffices: the model's own attempts, filtered by a pass/fail check of each final plan. Starting from Qwen3 4B Thinking or Gemma 4 E4B, we repeatedly prompt the model on PDDL tasks, check each final plan with a symbolic validator, keep the traces whose plans pass, and fine-tune the next generation on the pool of successful traces. Across three different domains, later generations solve substantially more held-out test tasks, including tasks from a harder parameter distribution, and find valid plans with longer horizons. Over ten generations, performance plateaus without model collapse. Our study also shows that curation is essential: training on all generated traces performs much worse despite using more data. Because we hold the base model, tasks, validator, and tuning recipe fixed, these results show that this minimal mechanism is sufficient to produce large planning gains. It is therefore a candidate explanation for part of the progress of frontier models, although our experiments cannot show how those models were trained.

1 INTRODUCTION

LLM planning has changed quickly. Early evaluations found that even the strongest models rarely produced executable plans for simple PDDL tasks (Valmeekam et al., 2023), and Kambhampati et al. (2024) argued that LLMs cannot plan autonomously and should instead support external planners and verifiers. Later evaluations of reasoning models found much stronger planning (Valmeekam et al., 2024). On challenging PDDL problems, Corrêa et al. (2026) report that GPT-3.5 solves none, GPT-5 performs comparably to state-of-the-art classical planners, and Gemini 3.1 Pro solves more tasks than the strongest planner. The 2026 AI Index also reports this change (Sajadieh et al., 2026).

As the providers of these models do not disclose their training data or post-training procedures, the cause of this progress is unknown. Prior work suggests that planning gains require substantial planning-specific supervision. Supervised fine-tuning on optimal plans from a classical planner did not produce the longer-horizon generalization its authors sought (Bohnet et al., 2024), reinforcement learning with shaped rewards gave modest gains (Huang et al., 2025), and larger gains came from instruction tuning on given solution plans, with explanations and structured validator feedback (Verma et al., 2025).

In this paper, we test a much simpler hypothesis. Outputs of deployed models can enter training data for later generations, and correct outputs are more likely to be kept and reused than incorrect ones. We ask whether this loop alone improves planning when its only signal is a pass/fail check of each plan the model produces.

We call this mechanism *iterative deployment* (Figure 1). This has already been studied by previous work, such as STaR (Zelikman et al., 2022) and ReST^{EM} (Singh et al., 2024). Here, we show that iterative deployment is sufficient to improve the planning skills of large language models. Starting from a base model, we prompt it on a fixed set of PDDL tasks, check each final plan with a sound validator (Howey & Long, 2003), keep the traces whose plans pass, and fine-tune the next generation

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

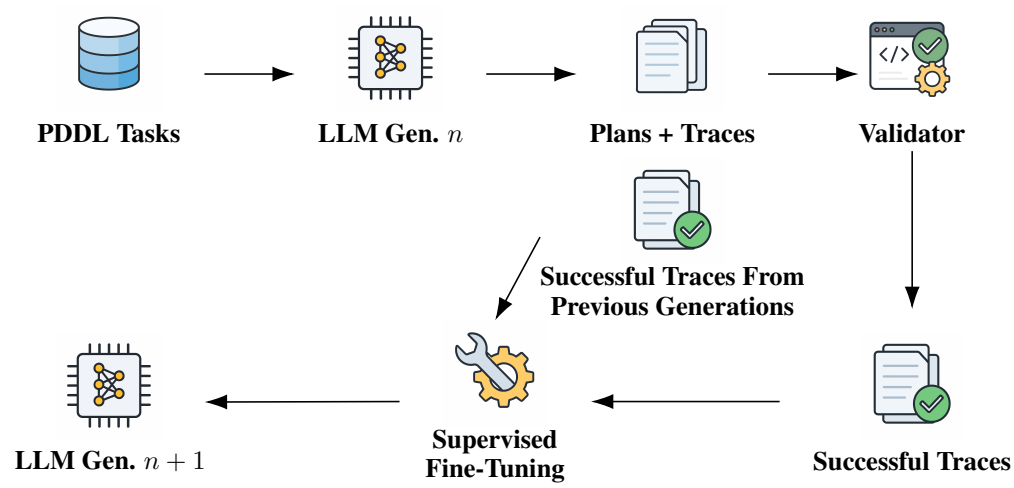


Figure 1: One round of iterative deployment. Generation n of the model attempts every task in a fixed set of PDDL tasks. A validator checks each final plan, and the traces whose plans pass join the successful traces from earlier generations. Generation $n + 1$ is fine-tuned from the base model on this pool. User curation motivates the abstraction but supplies no data to our experiments.

on the pool of successful traces collected so far. The deployment scenario motivates this abstraction but supplies none of our data. Every training trace comes from our own model on a fixed training task; no trace comes from users, an external planner, or a teacher model.

We test two model families, Qwen3 4B Thinking (Qwen Team, 2025) and Gemma 4 E4B (Gemma Team, 2026), and find large gains with both. By generation 5, Qwen solves more than three times as many held-out tasks as its base model in all three domains, and Gemma between 1.5 and 1.8 times as many. Over ten generations, performance plateaus after the first few generations but does not fall back toward the base model. Later generations also solve tasks for which they return much longer valid plans, so the gains are unlikely to come only from better formatting or better following of instructions. Importantly, curation also matters: fine-tuning on all generated traces works much worse than fine-tuning on the successful ones alone, although it uses many more traces.

Because we hold the base model, task distribution, validator, and tuning recipe fixed, our experiments isolate one mechanism and show that it is sufficient for large planning gains. They cannot show that frontier developers used this mechanism, and they do not rule out other explanations, such as scale, more planning data in pretraining, or transfer from reinforcement learning on mathematics and code. They do show that, in our setting, gains of this size need nothing beyond the model's own attempts and an exact check of each final plan. Iterative deployment is thus the simplest candidate explanation that we can test under full control.

We also include a direct connection to reinforcement learning. In the idealized on-policy case, supervised fine-tuning on successful traces has the same update direction, up to scaling, as REINFORCE (Williams, 1992) with binary rewards. Aggregating traces from earlier generations adds an off-policy component that admits an importance-weighted interpretation. We present this as an interpretation rather than a core contribution.

Contributions. Our contributions are:

- We test, under controlled conditions, whether an established generate, validate, and fine-tune loop improves end-to-end classical planning across model generations.
- We show that successful self-generated traces improve planning on held-out tasks from a harder parameter distribution, including tasks for which the model returns longer valid plans.
- With an ablation study, we show that the gains depend on keeping only traces whose final plans pass the validator.

- We provide a secondary theoretical interpretation that connects the iterative deployment mechanism to binary-reward policy optimization.

Scope. We study iterative deployment specifically for end-to-end classical planning. The same mechanism has been studied in other settings (Zelikman et al., 2022; Singh et al., 2024; Shao et al., 2024), but planning is harder in important respects: solutions are sequences of actions rather than single-shot answers, local errors compound because every action must be applicable in the current state, and plans can be exponentially long. Within classical planning, previous attempts have either fallen short or required substantial complexity: supervised fine-tuning on optimal plans from external planners did not yield longer-horizon generalization (Bohnet et al., 2024), RL fine-tuning with shaped rewards produced only incremental gains (Huang et al., 2025), and positive results have required explicit logical chain-of-thought instruction tuning with structured validator feedback (Verma et al., 2025). Therefore, we do not investigate these previously studied techniques in this paper. We show that iterative deployment, despite being substantially simpler, already produces large improvements.

Throughout the paper, we call the starting model the *base model* or *generation 0*. The model obtained after n rounds of iterative deployment is called the *n-th generation*.

2 Background and Related Work

We consider classical planning, where a single agent applies deterministic actions in a fully observable, discrete environment. Classical planning tasks are commonly described using the Planning Domain Definition Language (PDDL) (McDermott et al., 1998; Haslum et al., 2019). A PDDL task consists of objects, predicates that describe relations between objects, actions that change these relations, an initial state represented as a set of ground atoms, and a goal that lists the ground atoms that must hold at the end. PDDL usually separates these ingredients into two files: a domain file, which defines the predicates and actions shared by all tasks in a domain, and a problem file, which defines the objects, initial state, and goal for one task.

Each action specifies preconditions (atoms that must hold for the action to be applicable) and effects (atoms added to or deleted from the state when the action is applied). A plan is a finite sequence of actions $\pi = \langle a_1, \ldots, a_k \rangle$. The plan is valid for a task if each action a_i is applicable in the state reached after applying $a_1, \ldots, a_{i-1}$ from the initial state. The plan solves the task if the state reached after a_k satisfies the goal. In this paper, we only require satisficing plans: any valid plan that reaches the goal is acceptable regardless of its length, although shorter plans are traditionally preferred.

Given a candidate plan, an external validator can simulate the actions, check every precondition, and determine whether the final state reaches the goal. This gives an exact binary label for each plan without expert demonstrations, human preference labels, or a learned reward model. The label concerns only the plan; it says nothing about the reasoning that produced it.

2.1 Related Work

Planning with LLMs. Valmeekam et al. (2023) show that early LLMs are unreliable even on simple planning tasks, and Kokel et al. (2025) show that related subproblems such as plan validation are also hard for LLMs. Kambhampati et al. (2024) argue that LLMs should work inside frameworks with external planners and verifiers. However, recent evaluations of reasoning models report much stronger results (Corrêa et al., 2026). These studies show a change in capability but do not identify its training cause, which is the question we study.

Within classical planning, several papers train models on plans. Rossetti et al. (2024) train domain-specific GPT models on planner-generated data and obtain strong in-distribution performance. Bohnet et al. (2024) report that supervised fine-tuning on optimal plans from an off-the-shelf planner failed to produce the longer-horizon generalization they sought. Huang et al. (2025) use sequence-level RL with a shaped LCCS reward and report incremental improvements in validity and executability on longer problems. Lehnert et al. (2024) train Transformers from scratch on A* search traces for Sokoban and mazes, then repeatedly fine-tune them on their own shorter traces that still reach optimal plans. We start from a pretrained LLM and use no plans or search traces from a planner, no shaped rewards, and no error feedback. Because these studies differ from ours in model, tasks, supervision, and metrics, we consider their published numbers are context rather than baselines.

Previous work also combine LLMs with PDDL and symbolic validators to produce policies or heuristic functions in format of code (Silver et al., 2024; Tuisov et al., 2025; Corrêa et al., 2025). In this line of research, the model is prompt to write Python code (e.g., a generalized policy, a heuristic function, or an entire planner), using validator feedback to debug the program in context. However, none of these works train or fine-tune the weights of the models. We instead generate one plan per task and update the model's weights on validated traces across generations.

Self-Improvement from Model-Generated Traces. The Self-Taught Reasoner (STaR) (Zelikman et al., 2022) fine-tunes a model on its own rationales that lead to correct answers, restarting from the base model at every round. Singh et al. (2024) scale this recipe as ReST^{EM}, which repeatedly generates samples, filters them with binary feedback, and fine-tunes on the accepted outputs for mathematics and code. Our loop follows the same recipe, with two design choices of its own: it aggregates successful traces across generations, and it keeps one trace per task, the one with the shortest plan. Lee et al. (2025) show that transformers trained on their own filtered outputs generalize to longer arithmetic problems and longer maze paths, and that the loop fails on the harder tasks without filtering. Their models mostly learn each task from scratch with a curriculum of increasing difficulty, whereas we start from a pretrained LLM and keep the task pool fixed. We study what this recipe does in end-to-end PDDL planning, where a validator labels each final plan exactly. The label removes noise about plan validity but does not verify the reasoning that precedes the plan.

Other work improves reasoning with better rationales, structured feedback, or larger teacher models (e.g., Zhang et al., 2024; Muennighoff et al., 2025; Wen et al., 2025); Verma et al. (2025), for example, improve symbolic planning with instruction tuning and structured validator feedback. Our signal is narrower: a binary check of each final plan.

Reinforcement Learning and Reasoning. Reinforcement learning is now a common tool for improving reasoning and decision-making in LLMs (e.g., Shao et al., 2024; DeepSeek-AI et al., 2025). Methods such as GRPO optimize an explicit reward function without requiring supervised targets (Shao et al., 2024). We do not position our method as a replacement for RL in general, although in classical planning specifically RL has already been applied to end-to-end plan generation with only modest gains (Huang et al., 2025). Instead, we show that in the planning setting, successful-trace fine-tuning is a special case of binary-reward policy optimization. This connection helps explain why such a simple procedure can work, even though we implement it as supervised fine-tuning.

Model Collapse and Recursive Training. Shumailov et al. (2024) show that recursively training on model outputs in place of real data can lead to model collapse. Later work shows that data management can stabilize this loop: accumulating real and synthetic data (Gerstgrasser et al., 2024), keeping enough real data with a good initial model (Bertrand et al., 2024), verifying synthetic samples (Feng et al., 2025), correcting them (Gillman et al., 2024), or curating them, which amounts to implicit preference optimization (Ferbach et al., 2024). That curation can stabilize recursive training is therefore known. Our result is specific to planning: with an exact check of each final plan, recursive training on the model's own traces improves held-out planning without planner-generated plans, error feedback, or explicit rewards, and our matched ablation isolates the role of the filter. We do not claim that curation always prevents collapse.

3 Iterative Deployment for Planning

Iterative deployment repeatedly trains a model on its own successful attempts. The intuition is that a model can learn from the tasks it already solves and use those solutions as building blocks for harder tasks. This requires a reliable way to separate successful attempts from failed ones, which a symbolic validator provides in planning.

Formally, let M_n denote the model at generation n, parameterized by θ_n. We assume access to a dataset of planning tasks without solutions, $\mathcal{D}_{\text{train}}$, and to a deterministic external validator $V(x, y)$ that returns true if trace y contains a valid solution for task x, and false otherwise. In our implementation, V is VAL, which checks only the extracted final plan and not the reasoning that precedes it. We call a trace *successful* if its final plan is valid; malformed outputs and outputs without an extractable plan are unsuccessful. The validator also lets us compare successful traces by plan length.

The iterative deployment mechanism then proceeds as follows:

- **Deployment and Trace Collection.** In each iteration n, we prompt the current model M_n on the tasks in $\mathcal{D}_{\text{train}}$. For each task x, the model samples exactly one trace $y \sim \pi_{\theta_n}(\cdot \mid x)$ containing its reasoning trace (i.e., chain-of-thought) and a candidate plan.
- **Validation.** We check each trace with the validator and keep only successful traces: $\mathcal{D}^{(n)}_{valid} = \{(x, y) \mid V(x, y) = \text{True}\}$. We discard the rest.
- **Curation and Aggregation.** We aggregate the successful traces from the current generation with those from all previous generations into a candidate pool $\mathcal{P}_{n+1} = \bigcup_{i=0}^{n} \mathcal{D}^{(i)}_{valid}$, so that newly solved tasks are added without discarding earlier successes. We do not ablate aggregation separately.
- During aggregation, we apply a second curation step, a **selection mechanism**. If several successful traces exist for the same task (e.g., from different generations), we keep only the best one. In our experiments, we select the trace with the shortest plan and break ties by fewer reasoning tokens, but other task-specific metrics are possible.

  We define a selection function Sel that applies the select mechanism to a set of candidate traces, and define the training set for the next generation as $\mathcal{T}_{n+1} = \mathrm{Sel}(\mathcal{P}_{n+1})$.
- **Supervised Fine-Tuning (SFT).** Finally, we produce the next generation M_{n+1} by fine-tuning the base model M_0 on $\mathcal{T}_{n+1}$ with the standard next-token prediction objective, using LoRA adapters (Hu et al., 2022); see Appendix A.2 for details. As in STaR (Zelikman et al., 2022), every generation starts from M_0. This avoids stacking adapters and keeps the base checkpoint fixed, so that, apart from the randomness of training and sampling, the training pool is the only thing that changes between generations. Starting each round from a newer or stronger checkpoint would mix the effect of the loop with capability supplied by a different pretrained model.

The loop is intentionally simple. We use no plans from an external planner, no hand-designed curricula, and no explicit reward optimization; the only component beyond standard SFT is a validator that separates successful traces from unsuccessful ones. Every trace in the training pool comes from our own model on a training task. The prompt contains two fixed in-context examples (Appendix A.2), which never enter the pool.

We do not construct a curriculum manually, but the model and validator may induce one. Early generations may solve easier tasks and contribute short successful traces, and later generations may use those traces as building blocks for harder tasks from the same domain. For example, traces from Sokoban instances with one box may help later generations solve instances with two or three boxes.

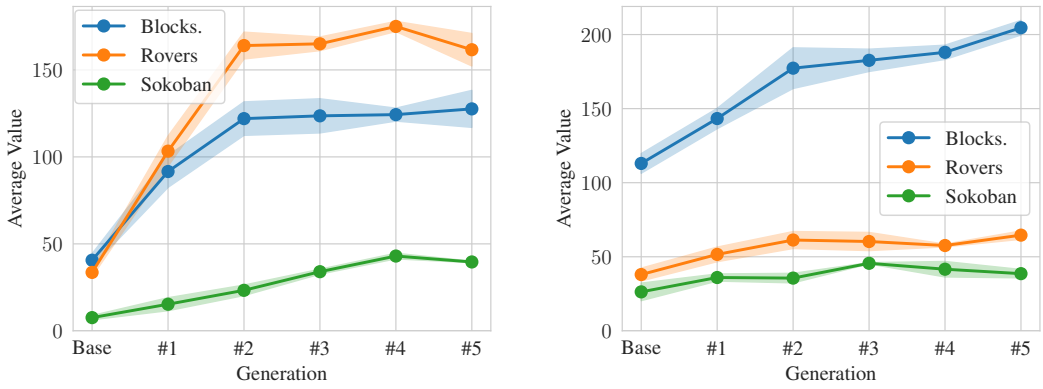
The validator is the only training signal in the loop: a trace is kept if its final plan is valid, and among valid traces for the same task, the selection rule keeps the one with the shortest plan. This is a controlled version of deployment, where correct and simple outputs are more likely to be kept and reused.

Without curation, recursive training on generated data can degrade performance and eventually lead to collapse (Shumailov et al., 2024); later work shows that verification and curation can stabilize it (Section 2.1). We ask whether, in planning, validation, selection, and aggregation turn recursive training into a source of improvement.

4 Empirical Validation

We evaluate iterative deployment on *classical planning domains* encoded in PDDL (McDermott et al., 1998; Haslum et al., 2019), the standard description language used in planning competitions (Taitler et al., 2024). These experiments study iterative capability bootstrapping on fixed task pools within each planning domain. We ask whether successful traces from earlier generations help later generations solve more tasks, solve them more consistently, and reach longer plans.

Experimental protocol. For each domain, we split the generated tasks into a training pool $\mathcal{D}_{\text{train}}$ and a disjoint test pool $\mathcal{D}_{\text{test}}$. Iterative deployment uses only $\mathcal{D}_{\text{train}}$: we prompt the model on training tasks, validate the generated traces, and fine-tune the next generation on the curated successful traces



(a) Qwen3 4B Thinking

(b) Gemma 4 E4B

Figure 2: Average number of solved test tasks across generations in the three domains. Shaded areas show one standard deviation.

as described in Section 3. We evaluate every generation on $\mathcal{D}_{\text{test}}$, which is never used for fine-tuning or trace selection. We use two base models, Qwen3 4B Thinking (Qwen Team, 2025) and Gemma 4 E4B (Gemma Team, 2026), with the same inference and fine-tuning settings, and train one model per domain. When multiple generations solve the same task, we keep only one training sample, choosing the sample based on the selection method described above. We give additional setup details in §A.2. We will release the task generators, prompts, generated traces, training and evaluation code, and trained LoRA adapters upon acceptance.

Benchmark. Our benchmark consists of three domains commonly used in classical planning. All three domains appeared in the Learning Track of the International Planning Competition 2023 (IPC 2023) (Taitler et al., 2024):

Blocksworld: Several blocks are stacked on a table. The goal is to rearrange them to match a target configuration. Planning requires deciding which stacks to dismantle and which blocks to move first. A simple polynomial strategy exists, but the model must still carry out nontrivial sequential reasoning to execute it reliably.

Rovers: This domain draws from Mars rover planning (Long & Fox, 2003). Rovers must navigate among waypoints, calibrate cameras, collect soil or rock samples, and communicate results while satisfying visibility and movement constraints. Plan existence for this domain can be decided in polynomial time (Helmert, 2006).

Sokoban: This PDDL version of Sokoban requires an agent to push boxes to designated goal cells on a grid with walls and obstacles. Boxes can be pushed but not pulled, which creates dead ends and makes the domain combinatorially difficult. Sokoban is **PSPACE**-complete (Culberson, 1997), so plans may be exponentially long.

For each domain, we generated separate training and test pools, each with 1000 instances. The test pool is disjoint from the training pool and drawn from a parameter distribution shifted toward harder tasks, with larger means and wider ranges for the main difficulty parameters. The ranges overlap, but part of the test pool lies beyond the largest training instances. The split therefore tests generalization to unseen and harder tasks, beyond the instances used to collect traces. Table 4 in Appendix A.4 reports both distributions.

Metrics. Our main metric is the number of solved tasks at each generation. We train one model per generation and report the average over three inference runs on the test pool. Finally, we analyze the length of the returned plans.

Planning performance improves. Figure 2 shows the average number of solved test tasks for each generation of both models. For Qwen3 4B Thinking (Figure 2a), performance improves sharply in

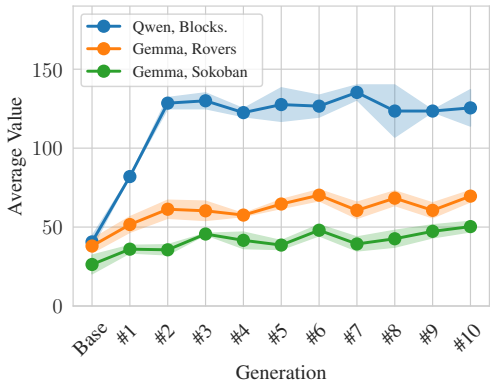


Figure 3: Average number of solved test tasks over ten generations for Qwen3 4B Thinking in Blocksworld and Gemma 4 E4B in Rovers and Sokoban. Shaded areas show one standard deviation.

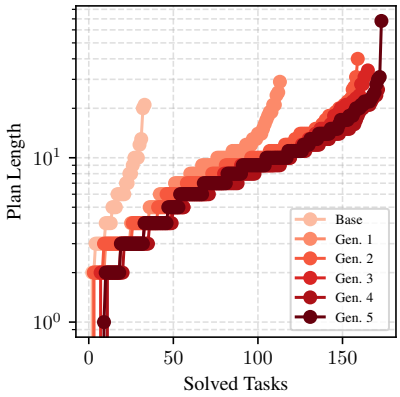


Figure 4: Performance across generations for Rovers on the test set. The y-axis shows plan length on a log scale, while the x-axis shows the cumulative number of solved instances sorted by plan length.

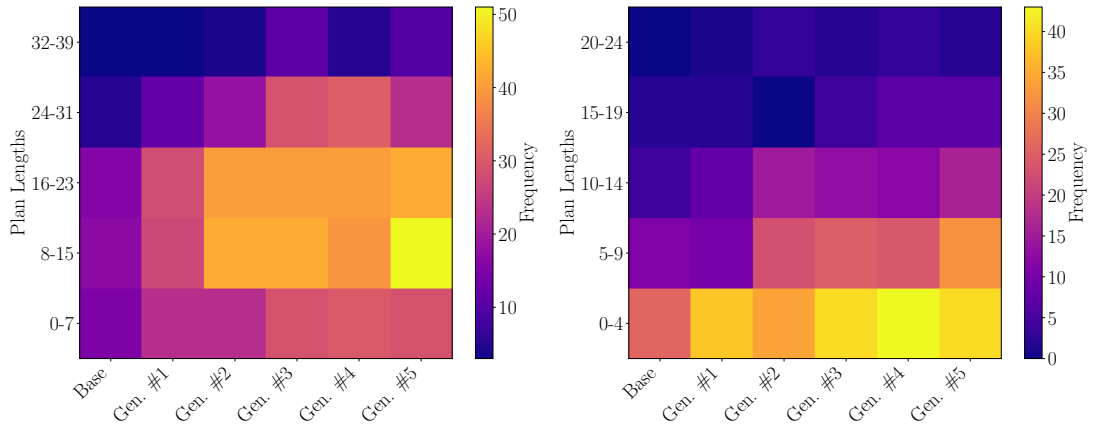
the first generations in all three domains, even though the test tasks are never used for fine-tuning. By generation 5, the average number of solved test tasks is 214% higher in Blocksworld, 381% higher in Rovers, and 421% higher in Sokoban. Gemma 4 E4B behaves similarly (Figure 2b). It starts from a stronger base model in all three domains, and by generation 5 it improves from 113.0 ± 6.6 to 204.6 ± 4.9 solved tasks in Blocksworld, from 38.0 ± 4.3 to 64.6 ± 2.8 in Rovers, and from 26.3 ± 5.9 to 38.6 ± 2.8 in Sokoban.

The trajectory is not strictly monotonic. Later generations fluctuate slightly, which is expected because inference is stochastic and the same task may succeed in one run but not another. Even so, the trend is clear: all deployed generations outperform the base model.

Appendix A.4 reports the full numeric results, including the number of tasks solved in all three runs.

What happens in more generations? To see whether the gains continue, level off, or reverse, we extended three runs to ten generations: Blocksworld for Qwen3 4B Thinking, and Sokoban and Rovers for Gemma 4 E4B. Figure 3 shows the results. For Qwen, most of the gain comes in the first two generations: solved tasks rise from 40.6 ± 3.7 for the base model to 128.5 ± 3.5 at generation 2, and then fluctuate between 122.5 ± 2.5 and 135.3 ± 4.7 up to generation 10 (125.5 ± 11.5). For Gemma in Sokoban, performance rises more gradually, from 26.3 ± 5.9 to 50.3 ± 3.3 at generation 10, with dips along the way. In Rovers, Gemma's pool grows to only 159 traces by generation 10, and solved tasks stay between 57.6 ± 0.9 and 70.2 ± 3.5 from generation 2 on. Slow pool growth does not explain this plateau, however: Qwen's Blocksworld pool grows from 356 to 588 traces between generations 5 and 10, and its performance levels off as well. In none of the three runs does performance fall back toward the base model: over ten generations, iterative deployment reaches a plateau and does not collapse. Table 7 in Appendix A.4 lists all values.

Later generations return longer plans. One might wonder whether the models learn to solve harder tasks or only learn a few tricks (e.g., format answers correctly, avoid minor mistakes). Later generations solve tasks for which they return longer plans. Returned plan length is an imperfect proxy for horizon because the plans need not be optimal. Figure 5 shows the plan length distribution for different model generations in the Blocksworld and Sokoban domains using Qwen 3 4B Thinking as the base model. A similar trend happens when using Gemma 4 E4B. It shows the distribution for one single pass over the benchmark. Both figures show a similar behavior: as the model evolves, later generations often find longer plans. For example, in Blocksworld, the base model finds mostly plans up to 20 steps, while generation 5 finds many plans up to 35 steps.



(a) Blocksworld

(b) Sokoban

Figure 5: Plan-length frequencies across generations of Qwen3 4B Thinking for Blocksworld and Sokoban.

Figure 4 shows the cumulative number of solved Rovers tasks by plan length, for a single pass over the test set. This view makes outliers more visible than the heatmaps in Figures 5a and 5b. Later generations solve more tasks with plans in the 10–30 step range, and generation 5 reaches larger outliers. When two generations solve the same task, however, their plan lengths often remain similar. Thus, later models mostly solve additional instances with longer plans, rather than writing longer plans for the same instances.

Is curation necessary? To test whether filtering successful traces really matters, we repeated the Blocksworld experiment with Qwen 3 4B Thinking but without curation. In this variant, we used all traces from previous generations for fine-tuning, except those that hit the maximum token limit.

Table 1 shows that this weaker variant still improves over the base model, presumably because additional data helps the model follow the prompt format more reliably. However, it is much worse than the curated version at every generation. By generation 5, the curated procedure solves 127.6 tasks, compared with 79.0 without curation, while using far fewer training traces. This ablation supports the main hypothesis of the paper: the quality of the self-generated training signal matters more than the number of model outputs. It does not show that curation prevents model collapse in general, only that, in an otherwise matched planning loop, keeping traces with valid final plans gives a stronger training signal than keeping nearly all outputs.

Gen.	w/ Curation	w/o Curation
Base	**40.6 ± 3.7**	**40.6 ± 3.7**
Gen. 1	**91.6 ± 9.1**	63.7 ± 6.6
Gen. 2	**122.0 ± 9.6**	79.0 ± 2.4
Gen. 3	**123.6 ± 9.8**	74.0 ± 3.3
Gen. 4	**124.3 ± 3.7**	71.0 ± 2.4
Gen. 5	**127.6 ± 10.6**	79.0 ± 3.0

Table 1: Average number of solved test tasks in Blocksworld for Qwen3 4B Thinking with and without curation.

Furthermore, the version with curation requires less data. For example, to train the 5th generation of the model, the version with curation only used 356 traces, while the version without curation used 4017. So curation improves performance by 61.5% while using a fraction of the training data.

5 Connection to Binary-Reward Policy Optimization

The results above show that iterative deployment can substantially improve end-to-end planning without planner demonstrations or explicit reward optimization. This is already meaningful for classical planning with LLMs, where prior work has found robust generalization difficult. The

connection to reinforcement learning below is therefore not the source of the empirical claim, but an interpretation of the training signal produced by validation and curation.

We first show the basic on-policy correspondence.

Proposition 1. *The update directions of the gradients for SFT using only valid traces and REINFORCE with binary rewards are identical.*

The proof appears in Appendix A.1. Invalid traces have reward zero, so they contribute no policy-gradient signal. The resulting update is the same, up to a positive scaling factor, as maximizing the log-likelihood of the successful traces.

Iterative deployment also aggregates successful traces from previous generations. This introduces an off-policy component, which can be expressed through importance weighting in an idealized analysis.

Proposition 2. *SFT on a mixture of on- and off-policy valid traces is equivalent to REINFORCE with binary rewards augmented by importance-weighted contributions from the behavior policy.*

The proof is also in Appendix A.1. The two propositions relate our analysis to successful-trace self-training. STaR (Zelikman et al., 2022) and ReST^{EM} (Singh et al., 2024) fine-tune a model on its own successful samples, which corresponds to the on-policy case of Proposition 1; Zelikman et al. (2022) already note this connection to policy gradients. Our training pool also keeps successful traces from earlier generations, which Proposition 2 covers. Together, these results give the following interpretation for the iterative deployment:

Proposition 3. *Iterative deployment using only valid traces, as described in Section 3, is a special case of REINFORCE with implicitly defined binary rewards.*

This interpretation also clarifies the limits of the analogy. In explicit RL, the training algorithm specifies the reward function. In iterative deployment, the effective reward is implicit: validators, user choices, and data curation decide which traces enter future training. Our implementation does not estimate or apply explicit importance weights; the off-policy result is an interpretation of the training signal induced by aggregated validated traces. The shortest-plan selection rule also adds a task-specific preference beyond binary success. This can be useful when the curation signal is reliable, as in our planning setting, but it can also amplify biases in validators or user behaviour. It also changes the usual model-collapse picture (Shumailov et al., 2024): recursively generated data is not used uniformly, but filtered through a success signal. Later work shows that such curation can stabilize recursive training; we relate our results to it in Section 2.1.

6 Discussion

Frontier LLMs have become much better planners, but the cause of this progress is unknown. We tested the simplest hypothesis we could control: iterative deployment, where each generation is fine-tuned only on the model's own attempts whose final plans pass a validator. With two model families and three classical planning domains, this signal alone was sufficient for large gains on held-out tasks, including tasks from a harder distribution. In our ablation study, training on all generated traces worked much worse, despite using more data.

Longer runs show where the mechanism stops being efficient. In our longer runs with 10 geenrations, performance plateaus after the first few generations without falling back toward the base model, and it does so whether the pool of successful traces grows quickly or slowly. Our supervised fine-tuning only learns from successful traces, so failed ones are discarded. Other training mechanisms that can learn from such failes traces may avoid the plateau further.

Limitations. Our pipeline needs a sound and cheap verifier. Planning has access to such verifiers, but many tasks do not. Iterative deployment could still benefit from signals coming from human or AI feedback, but we did not evaluate those. Moreover, we study two small open models, so we cannot predict how the gains change with model scale.

The general value of verification in recursive training is known. Our results show what it does for planning: a model's own validated plans are enough to make the next generation a better planner. This makes iterative deployment a candidate explanation for part of the recent progress in LLM planning, and a controlled setting in which to study it.

AI USE STATEMENT

Qwen3 4B Thinking and Gemma 4 E4B are the subject of our experiments and generate all traces used for training and evaluation. We also used AI assistants to check the literature, review and revise the manuscript, and write plotting code. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

The supplementary material contains the code for inference, plan validation, trace selection, and fine-tuning, the training and test tasks of all three domains, and the curated training set of the fifth Qwen generation in each domain. Appendix A.2 describes the prompt, inference settings, and validator, and Table 2 lists all fine-tuning hyperparameters. Both base models are publicly available. We will release all generated traces and trained adapters upon acceptance.

REFERENCES

Quentin Bertrand, Avishek Joey Bose, Alexandre Duplessis, Marco Jiralerspong, and Gauthier Gidel. On the stability of iterative retraining of generative models on their own data. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. Exploring and benchmarking the planning capabilities of large language models. arXiv:2406.13094 [cs.CL], 2024.

Augusto B. Corrêa, André G. Pereira, and Jendrik Seipp. Classical planning with LLM-generated heuristics: Challenging the state of the art with Python code. In *Proceedings of the Thirty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2025)*, 2025.

Augusto B. Corrêa, André G. Pereira, and Jendrik Seipp. Frontier large language models rival state-of-the-art planners. arXiv:2511.09378v2 [cs.AI], 2026.

Joseph C. Culberson. Sokoban is PSPACE-complete. Technical Report TR 97-02, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada, 1997.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z.F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, and Guanting Chen et al. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv:2501.12948 [cs.CL], 2025.

Yunzhen Feng, Elvis Dohmatob, Pu Yang, Francois Charton, and Julia Kempe. Beyond model collapse: Scaling up with synthesized data requires verification. In *Proceedings of the Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025.

Damien Ferbach, Quentin Bertrand, Avishek Joey Bose, and Gauthier Gidel. Self-consuming generative models with curated data provably optimize human preferences. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, 2024.

Gemma Team. Gemma 4 technical report, 2026. URL `https://arxiv.org/abs/2607.02770`.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, Daniel A. Roberts, Diyi Yang, David L. Donoho, and Sanmi Koyejo. Is model collapse inevitable? Breaking the curse of recursion by accumulating real and synthetic data. arXiv:2404.01413 [cs.LG], 2024.

Nate Gillman, Michael Freeman, Daksh Aggarwal, Chia-Hong Hsu, Calvin Luo, Yonglong Tian, and Chen Sun. Self-correcting self-consuming loops for generative model training. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024.

Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*, volume 13 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, 2019.

Malte Helmert. New complexity results for classical planning benchmarks. In Derek Long, Stephen F. Smith, Daniel Borrajo, and Lee McCluskey (eds.), *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 52–61. AAAI Press, 2006.

Richard Howey and Derek Long. VAL's progress: The automatic validation tool for PDDL2.1 used in the International Planning Competition. In Stefan Edelkamp and Jörg Hoffmann (eds.), *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*, 2003.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the Tenth International Conference on Learning Representations (ICLR 2022)*. OpenReview.net, 2022.

Sukai Huang, Trevor Cohn, and Nir Lipovetzky. Chasing progress, not perfection: Revisiting strategies for end-to-end LLM plan generation. In Daniel Harabor and Miquel Ramirez (eds.), *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2025)*. AAAI Press, 2025.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. ACPBench: Reasoning about action, change, and planning. In Julie Shah and Zico Kolter (eds.), *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2025)*. AAAI Press, 2025.

Nayoung Lee, Ziyang Cai, Avi Schwarzschild, Kangwook Lee, and Dimitris Papailiopoulos. Self-improving transformers overcome easy-to-hard and length generalization challenges. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, 2025.

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. In *Proceedings of the First Conference on Language Modeling (COLM 2024)*, 2024.

Derek Long and Maria Fox. The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University, 1998.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. arXiv:2501.19393 [cs.CL], 2025.

Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In Pat Langley (ed.), *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, pp. 759–766. Morgan Kaufman, 2000.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, Mattia Chiari, and Matteo Olivato. Learning general policies for planning through GPT models. In Sara Bernardini and Christian Muise (eds.), *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, pp. 500–508. AAAI Press, 2024.

Sha Sajadieh, Loredana Fattorini, Raymond Perrault, Yolanda Gil, Vanessa Parli, Lapo Santarlasci, Juan Pava, Nestor Maslej, Russ Altman, Erik Brynjolfsson, Carla Brodley, Jack Clark, Virginia Dignum, Vipin Kumar, James Landay, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Elham Tabassi, Russell Wald, Toby Walsh, and Dan Weld. The AI index 2026 annual report. Technical report, AI Index Steering Committee, Institute for Human-Centered AI, Stanford University, 2026.

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:2402.03300 [cs.CL], 2024.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross J. Anderson, and Yarin Gal. AI models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.

Tom Silver, Soham Dan, Kavitha Srinivas, Josh Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in PDDL domains with pretrained large language models. In Jennifer Dy and Sriraam Natarajan (eds.), *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI 2024)*, pp. 20256–20264. AAAI Press, 2024.

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. *Transactions on Machine Learning Research*, 2024.

Ayal Taitler, Ron Alford, Joan Espasa, Gregor Behnke, Daniel Fišer, Michael Gimelfarb, Florian Pommerening, Scott Sanner, Enrico Scala, Dominik Schreiber, Javier Segovia-Aguas, and Jendrik Seipp. The 2023 International Planning Competition. *AI Magazine*, 45(2):280–296, 2024. doi: 10.1002/aaai.12169.

Alexander Tuisov, Yonatan Vernik, and Alexander Shleyfman. LLM-generated heuristics for AI planning: Do we even need domain-independence anymore? arXiv:2501.18784 [cs.AI], 2025.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models – a critical investigation. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, pp. 75993–76005, 2023.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. Planning in strawberry fields: Evaluating and improving the planning and scheduling capabilities of LRM o1. arXiv:2410.02162 [cs.CL], 2024.

Pulkit Verma, Ngoc La, Anthony Favier, Swaroop Mishra, and Julie A. Shah. Teaching LLMs to plan: Logical chain-of-thought instruction tuning for symbolic planning. arXiv:2509.13351 [cs.AI], 2025.

Jiaxin Wen, Zachary Ankner, Arushi Somani, Peter Hase, Samuel Marks, Jacob Goldman-Wetzler, Linda Petrini, Henry Sleight, Collin Burns, He He, Shi Feng, Ethan Perez, and Jan Leike. Unsupervised elicitation of language models. arXiv:2506.10139 [cs.CL], 2025.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022.

Edwin Zhang, Vincent Zhu, Naomi Saphra, Anat Kleiman, Benjamin L. Edelman, Milind Tambe, Sham M. Kakade, and Eran Malach. Transcendence: Generative models can outperform the experts that train them. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, 2024.

A Appendix

A.1 Proofs

In this section we present the proofs for our theoretical results of Section 5. We use the same notation and terminology from the paper.

Proposition A1. *The update directions of the gradients for SFT using only valid traces and REINFORCE with binary rewards are identical.*

Proof. Let x denote an input (e.g., a prompt), and let $y = (y_1, \ldots, y_T)$ denote a complete trace produced by a policy $\pi_\theta(y \mid x)$ with parameters θ (e.g. the LLM above), where T denotes the length of the trace under consideration.

We assume a binary reward function $R(x, y) \in \{0, 1\}$, where $R(x, y) = 1$ if y is a valid trace for x, and 0 otherwise.

The learning objective of REINFORCE is the expected reward:

$$J(\theta) = \mathbb{E}_{x\sim\mu}\, \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[R(x, y)].$$

The gradient $\nabla_\theta J(\theta)$ is

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y}\big[R(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x)\big].$$

Using the factorization

$$\log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \log \pi_\theta(y_t \mid y_{<t}, x),$$

we obtain

$$\nabla_\theta \log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid y_{<t}, x).$$

Thus, we can rewrite $\nabla_\theta J(\theta)$ as

$$\nabla_\theta J(\theta) = \mathbb{E}_{x,y}\left[R(x, y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid y_{<t}, x)\right].$$

Suppose we sample a dataset $\mathcal{D} = \{(x_i, y_i, R_i)\}_{i=1}^{N}$ for $N \in \mathbb{N}$ with $R_i \in \{0, 1\}$. A Monte Carlo estimate of the REINFORCE gradient is

$$\widehat{\nabla_\theta J} = \frac{1}{N} \sum_{i=1}^{N} R_i \sum_t \nabla_\theta \log \pi_\theta(y_{i,t} \mid y_{i,<t}, x_i).$$

Let $\mathcal{D}_+ = \{(x_i, y_i) : R_i = 1\}$ be the set of valid traces, and $\mathcal{D}_- = \{(x_i, y_i) : R_i = 0\}$ be the set of invalid traces. As our reward function $R(x, y)$ is binary, invalid traces do not affect the gradient (i.e., they contribute zero). Therefore, we can simplify the gradient:

$$\widehat{\nabla_\theta J} = \frac{1}{N} \sum_{i\in\mathcal{D}_+} \sum_t \nabla_\theta \log \pi_\theta(y_{i,t} \mid y_{i,<t}, x_i).$$

Let $N_+ = |\mathcal{D}_+|$, then

$$\widehat{\nabla_\theta J} = \frac{N_+}{N}\left[\frac{1}{N_+} \sum_{i\in\mathcal{D}_+} \sum_t \nabla_\theta \log \pi_\theta(y_{i,t} \mid y_{i,<t}, x_i)\right]. \tag{1}$$

On the flip side, SFT on the valid traces uses the loss

$$\mathcal{L}_{\text{SFT}}(\theta) = -\frac{1}{N_+} \sum_{i\in\mathcal{D}_+} \sum_{t=1}^{T_i} \log \pi_\theta(y_{i,t} \mid y_{i,<t}, x_i).$$

Its gradient is

$$\nabla_\theta \mathcal{L}_{\text{SFT}}(\theta) = -\frac{1}{N_+} \sum_{i \in \mathcal{D}_+} \sum_t \nabla_\theta \log \pi_\theta(y_{i,t} \mid y_{i,<t}, x_i).$$

We can then rewrite (1) in terms of $\nabla_\theta \mathcal{L}_{\text{SFT}}(\theta)$:

$$\widehat{\nabla_\theta J} = -\frac{N_+}{N} \nabla_\theta \mathcal{L}_{\text{SFT}}(\theta).$$

Thus, a gradient ascent step on $J(\theta)$,

$$\theta \leftarrow \theta + \eta \widehat{\nabla_\theta J},$$

is equivalent (up to the positive scaling factor N_+/N) to a gradient descent step on $\mathcal{L}_{\text{SFT}}$:

$$\theta \leftarrow \theta - \big(\eta\, N_+/N\big) \nabla_\theta \mathcal{L}_{\text{SFT}}(\theta).$$

Hence, the update directions are identical. □

Proposition A2. *SFT on a mixture of on- and off-policy valid traces is equivalent to REINFORCE with binary rewards augmented by importance-weighted contributions from the behavior policy.*

Proof. Let $p_\pi^+(x, y)$ denote the distribution over valid traces induced by sampling $x \sim \mu(x)$, then sampling $y \sim \pi_\theta(\cdot \mid x)$, and retaining the pair only if $R(x, y) = 1$:

$$p_\pi^+(x, y) \;\propto\; \mu(x)\, \pi_\theta(y \mid x)\, \mathbf{1}\{R(x, y) = 1\}.$$

Similarly, let

$$p_\beta^+(x, y) \;\propto\; \mu(x)\, \pi_\beta(y \mid x)\, \mathbf{1}\{R(x, y) = 1\}$$

denote the distribution over valid off-policy traces.

The dataset of valid traces can be modeled as a mixture

$$p_{\text{data}}(x, y) = (1 - \lambda)\, p_\pi^+(x, y) + \lambda\, p_\beta^+(x, y), \qquad \lambda \in [0, 1],$$

where λ reflects the proportion of off-policy data.

We can then show that SFT on a mixture of on- and off-policy valid traces can be written as a REINFORCE-like update with binary rewards and importance-weighted contributions from the behavior policy.

The SFT objective over all valid traces is

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim p_{\text{data}}}\big[\log \pi_\theta(y \mid x)\big].$$

Using the mixture form of p_{data}, this decomposes as

$$\mathcal{L}_{\text{SFT}}(\theta) = -(1 - \lambda)\, \mathbb{E}_{p_\pi^+}\big[\log \pi_\theta(y \mid x)\big] - \lambda\, \mathbb{E}_{p_\beta^+}\big[\log \pi_\theta(y \mid x)\big].$$

Differentiating $\mathcal{L}_{\text{SFT}}(\theta)$ we obtain the following decomposition of the gradient:

$$\nabla_\theta \mathcal{L}_{\text{SFT}}(\theta) = -(1 - \lambda)\, \mathbb{E}_{p_\pi^+}\big[\nabla_\theta \log \pi_\theta(y \mid x)\big] - \lambda\, \mathbb{E}_{p_\beta^+}\big[\nabla_\theta \log \pi_\theta(y \mid x)\big].$$

The first term, corresponding to the on-policy traces, is identical (up to a scalar factor) to REINFORCE with binary reward $R(x, y) \in \{0, 1\}$ under the current policy:

$$\mathbb{E}_{p_\pi^+}\big[\nabla_\theta \log \pi_\theta(y \mid x)\big] \;\propto\; \mathbb{E}_{x\sim\mu,\, y\sim\pi_\theta}\big[R(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x)\big].$$

To express the second term, corresponding to the off-policy traces, as an expectation under the current policy, we apply importance sampling (Precup et al., 2000). For any valid trace (x, y),

$$\frac{p_\beta^+(x, y)}{p_\pi^+(x, y)} = \frac{\pi_\beta(y \mid x)}{\pi_\theta(y \mid x)},$$

so we can rewrite the off-policy expectation as

$$\mathbb{E}_{p_\beta^+}\left[\nabla_\theta \log \pi_\theta(y \mid x)\right] = \mathbb{E}_{p_\pi^+}\left[\frac{\pi_\beta(y \mid x)}{\pi_\theta(y \mid x)} \nabla_\theta \log \pi_\theta(y \mid x)\right].$$

Substituting this expression produces the fully on-policy-looking form

$$\nabla_\theta \mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{p_\pi^+}\left[\left((1-\lambda) + \lambda \, \tfrac{\pi_\beta(y|x)}{\pi_\theta(y|x)}\right) \nabla_\theta \log \pi_\theta(y \mid x)\right].$$

Thus the SFT update may be interpreted as a REINFORCE-like gradient step under the current policy, restricted to valid traces, with an *effective reward*

$$R_{\mathrm{eff}}(x, y) \;\propto\; (1-\lambda) + \lambda \, \frac{\pi_\beta(y \mid x)}{\pi_\theta(y \mid x)}.$$

This shows that SFT on a mixture of on- and off-policy valid traces admits a REINFORCE-like interpretation with binary rewards and importance-weighted contributions from the behavior policy. In turn, this supports our claim that SFT using only valid traces can be viewed as binary-reward policy optimization (Williams, 1992). □

A.2 Experimental Setup

All our experiments were run on Intel Xeon Gold 6248R processors running at 3.00 GHz with an NVIDIA A100 80 GiB GPU. We use the Qwen3 4B as the base model (Qwen Team, 2025). Specifically, we use the Qwen3 4B Thinking 2507 version, which was better at following formatting instructions during our early experiments. We also use Gemma 4 E4B (Gemma Team, 2026) with the same inference and fine-tuning settings. During inference, we use temperature 0.6, top-$p = 0.95$, and a maximum of 32 768 new tokens. Each call returns one sample. We train one model per generation and report averages over three inference runs. The traces added to the training pool at each generation come from the first of these runs. To fine-tune our models, we use low-rank adaptation (LoRA) (Hu et al., 2022). The hyperparameters used for fine-tuning can be found in Appendix A.3. To avoid stacking adapters, each deployed generation is trained from its base model on the curated aggregate collected up to that point. To validate the computed plans, we use VAL (Howey & Long, 2003), which is commonly used in planning competitions.

Prompt. Our prompt format is based on the prompt used by Corrêa et al. (2025) for end-to-end PDDL planning. To solve a task T from a domain D, the prompt contains the following components:

1. The following instruction: "*You are generating plans for PDDL tasks. You will be given the PDDL domain and the PDDL instance, and you need to return the plan.*"
2. Two examples of PDDL domains, tasks, and their respective plans.
3. The PDDL domain D and the PDDL task T.

We use tasks from the traditional Gripper and Logistics domains as examples. These domains are not included in our experiments. The plans used to illustrate the examples are not optimal plans.

The two in-context examples are fixed across domains, generations, and runs. The target domain and task are inserted after the examples using the same tagged structure:

```
Domain 3:  <domain-3> ...</domain-3>
Task 3:  <task-3> ...</task-3>
Plan 3:
```

The model output after `Plan 3:` is treated as its final solution. VAL validates only the extracted final plan, not the reasoning trace. If the output is malformed or the final plan cannot be extracted in the expected format, we count the attempt as invalid. We will release the task generators, prompts, generated traces, training and evaluation code, and trained LoRA adapters upon acceptance.

Hyperparameter	Value
Maximum Context Length	32 768
Batch Size	1
Gradient Accumulation Steps	1
Epochs	2
Optimizer	AdamW (fused)
Learning Rate	1×10^{-4}
Learning Rate Scheduler	Cosine
Warm-up Ratio	0.01
Weight Decay	0.01
Max. Gradient Norm	1.0
LoRA Rank	16
LoRA Alpha	32
LoRA Dropout	0.05
LoRA Bias	none
LoRA Target Modules	all linear*
Fully Trained Modules	LM head, embeddings

Table 2: Hyperparameters used for fine-tuning. *For Gemma 4 E4B, LoRA targets only the attention and MLP projections of the language model and excludes the vision and audio encoders.

Model	Domain	SFT 1	SFT 2	SFT 3	SFT 4	SFT 5
Qwen3 4B Thinking	Blocksworld	56	132	210	290	356
	Rovers	32	150	252	303	333
	Sokoban	43	72	99	121	148
Gemma 4 E4B	Blocksworld	136	259	385	481	555
	Rovers	41	70	86	102	112
	Sokoban	48	85	120	139	152

Table 3: Number of curated traces used in each supervised fine-tuning step.

A.3 Fine-Tuning Hyperparameters

Table 2 lists the hyperparameters used for supervised fine-tuning with LoRA. At each generation, we randomly select 10% of the traces as a validation set. We evaluate validation loss every 10 steps. Table 3 reports the number of curated traces used for each fine-tuning step after validation, aggregation, and per-task trace selection.

A.4 Additional Results

Table 4 shows the train and test parameter distributions. Table 5 shows the average number of solved test tasks per domain and generation for both models. Table 6 shows unanimous@3, the number of test tasks solved in all three runs, which measures robustness to sampling noise during inference. Table 7 shows the ten-generation runs.

Domain	Parameter	Train ($\mathcal{D}_{\text{train}}$)	Test ($\mathcal{D}_{\text{test}}$)
Blocksworld	Blocks	2–10 (mean 7.3)	3–12 (mean 8.2)
Rovers	Rovers	1–5 (mean 3.8)	1–7 (mean 4.3)
	Waypoints	2–8 (mean 4.8)	2–12 (mean 5.9)
	Cameras	1–4 (mean 2.4)	1–6 (mean 2.9)
	Objectives	1–10 (mean 5.1)	1–15 (mean 6.6)
Sokoban	Grid size	5–10 (mean 9.0)	7–12 (mean 9.7)
	Boxes	1–7 (mean 4.3)	2–10 (mean 5.1)

Table 4: Training and test task distributions. Each pool contains 1000 instances per domain, and all test instances are disjoint from training instances.

Model	Domain	Base	#1	#2	#3	#4	#5
Qwen3 4B Thinking	Blocksworld	40.6 ± 3.7	91.6 ± 9.1	122.0 ± 9.6	123.6 ± 9.8	124.3 ± 3.7	$\mathbf{127.6 \pm 10.6}$
	Rovers	33.6 ± 2.4	103.3 ± 8.6	164.0 ± 7.7	165.0 ± 3.9	$\mathbf{175.0 \pm 2.9}$	161.6 ± 9.3
	Sokoban	7.6 ± 1.2	15.3 ± 3.7	23.3 ± 3.0	34.0 ± 1.4	$\mathbf{43.0 \pm 1.6}$	39.6 ± 0.4
Gemma 4 E4B	Blocksworld	113.0 ± 6.6	143.3 ± 6.8	177.3 ± 13.7	182.6 ± 7.5	188.0 ± 4.8	$\mathbf{204.6 \pm 4.9}$
	Rovers	38.0 ± 4.3	51.6 ± 4.9	61.3 ± 5.7	60.3 ± 6.1	57.6 ± 0.9	$\mathbf{64.6 \pm 2.8}$
	Sokoban	26.3 ± 5.9	36.0 ± 2.4	35.6 ± 3.2	$\mathbf{45.6 \pm 0.4}$	41.6 ± 5.2	38.6 ± 2.8

Table 5: Average number of solved test tasks per generation. Averages are over three runs; each domain has 1000 test tasks. Best value within each row is highlighted.

Model	Domain	Base	#1	#2	#3	#4	#5
Qwen3 4B Thinking	Blocksworld	3	18	26	26	**30**	27
	Rovers	1	28	74	80	**95**	81
	Sokoban	0	2	8	**15**	11	10
Gemma 4 E4B	Blocksworld	18	**35**	26	33	**35**	**35**
	Rovers	7	21	28	**30**	22	29
	Sokoban	6	12	10	**18**	11	12

Table 6: Number of test tasks solved in all three runs (unanimous@3) per generation. Each domain has 1000 test tasks. Best value within each row is highlighted.

	Qwen, Blocksworld		Gemma, Rovers		Gemma, Sokoban	
Gen.	Solved	Traces	Solved	Traces	Solved	Traces
Base	40.6 ± 3.7	—	38.0 ± 4.3	—	26.3 ± 5.9	—
Gen. 1	82.0	56	51.6 ± 4.9	41	36.0 ± 2.4	48
Gen. 2	128.5 ± 3.5	132	61.3 ± 5.7	70	35.6 ± 3.2	85
Gen. 3	130.0 ± 5.0	210	60.3 ± 6.1	86	45.6 ± 0.4	120
Gen. 4	122.5 ± 2.5	290	57.6 ± 0.9	102	41.6 ± 5.2	139
Gen. 5	127.6 ± 10.6	356	64.6 ± 2.8	112	38.6 ± 2.8	152
Gen. 6	126.6 ± 6.9	403	70.2 ± 3.5	121	48.0 ± 3.5	162
Gen. 7	135.3 ± 4.7	466	60.6 ± 5.1	134	39.3 ± 4.4	173
Gen. 8	123.5 ± 16.5	524	68.3 ± 4.6	145	42.6 ± 5.3	181
Gen. 9	123.5 ± 0.5	551	60.6 ± 4.7	154	47.3 ± 4.1	187
Gen. 10	125.5 ± 11.5	588	69.6 ± 3.6	159	50.3 ± 3.3	198

Table 7: Average number of solved test tasks over ten generations for Qwen3 4B Thinking in Blocksworld and Gemma 4 E4B in Rovers and Sokoban, and the number of curated traces used to train each generation.