Iterative Deployment Improves Planning Skills in LLMs

Augusto B. Corrêa
University of Oxford
United Kingdom

Yoav Gelberg
University of Oxford
United Kingdom

Luckeciano C. Melo
University of Oxford
United Kingdom

Ilia Shumailov
AI Sequrity Company
United Kingdom

André G. Pereira
UFRGS
Brazil

Yarin Gal
University of Oxford
United Kingdom

Abstract

We show that iterative deployment of large language models (LLMs), each fine-tuned on data carefully curated by users from the previous models' deployment, can significantly change the properties of the resultant models. By testing this mechanism on various planning domains, we observe substantial improvements in planning skills, with later models displaying emergent generalization by discovering much longer plans than the initial models. We then provide theoretical analysis showing that iterative deployment effectively implements reinforcement learning (RL) training in the outer-loop (i.e. not as part of intentional model training), with an implicit reward function. The connection to RL has two important implications: first, for the field of AI safety, as the reward function entailed by repeated deployment is not defined explicitly, and could have unexpected implications to the properties of future model deployments. Second, the mechanism highlighted here can be viewed as an alternative training regime to explicit RL, relying on data curation rather than explicit rewards.

1 Introduction

In this paper, we show that repeatedly deploying large language models (LLMs) and fine-tuning them on curated data from earlier deployments significantly improves their planning capabilities. This curation can be simply done by *validating* traces from previous generations, and *selecting* appropriate valid traces for future training. This mechanism produces a training process conceptually similar to RL fine-tuning but where the reward signal is left implicit. The core idea is simple: repeated deployment starts by users generating text with an LLM after its release. These texts go through a curation process, e.g. texts that do not capture user intent are rejected. Remaining texts are then shared to the web, with scrapes of the web including the curated text used to fine-tune the next generation of the LLM.

Iterative deployment is not a contrived setting: GPT-3.5 was trained on data scraped from the web following GPT-3's deployment, with data shared on the web at the time including curated texts generated by users using GPT-3 [2]. Similarly, GPT-4 was trained on data shared by users from GPT-3.5 and GPT-3 [17], and so on. With agent workflows becoming more commonplace, future training data will include agent traces from prior model generations, similarly leading to an iterative training on previous generation data. Figure 1 illustrates the basics of this mechanism.

We evaluate this mechanism in the well controlled setting of classical planning. Iterative deployment captures a pattern common to *planning*: when users use LLMs e.g. to review a product, help solve

Preprint. Preliminary version.

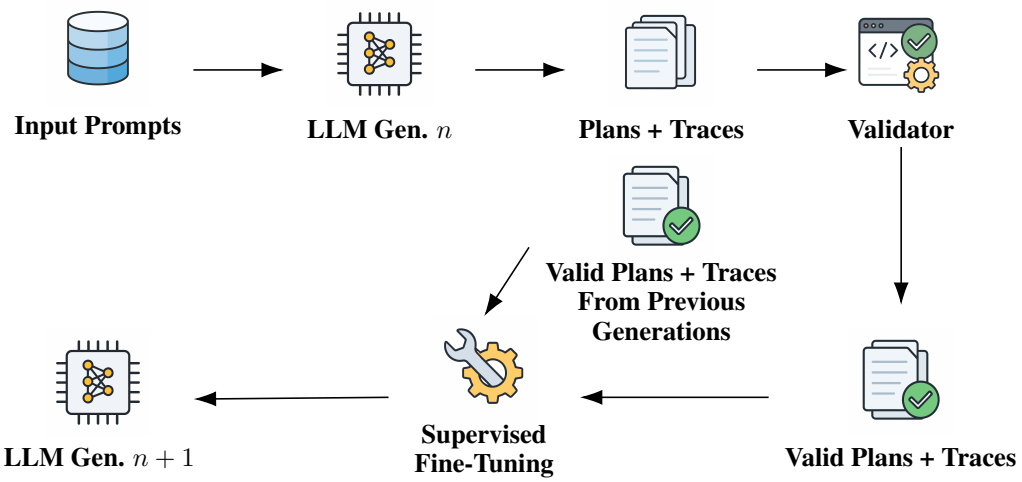


Figure 1: Single iteration of the iterative deployment mechanism for planning. Using a fixed a set of planning tasks, we prompt the current version of the LLM — referred to as the generation n of the model — to solve these tasks. An external validator (e.g. a human using a chatbot, or a computer programme in the case of planning) identifies the tasks solved correctly. Their traces and plans, together with the traces and plans from tasks solved by previous generations, are then used to fine-tune generation n, producing generation $n+1$ of the model.

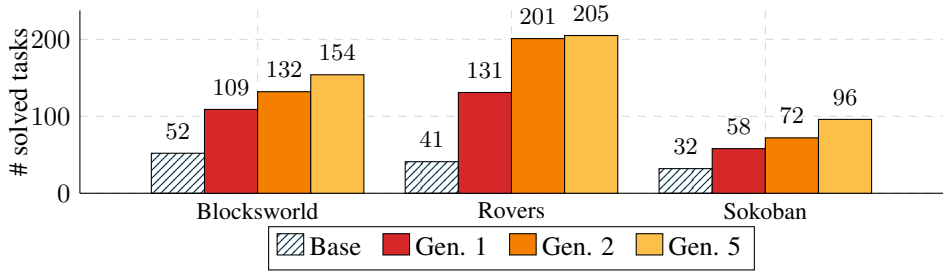


Figure 2: Summary of our main results. Number of solved tasks (with 1000 tasks per domain) for three different domains when comparing the base model with later deployed generations (generations 1, 2, and 5). Average over three separate runs. In all domains, the fifth generation more than doubles the performance of the base model.

a reasoning task, or plan a trip, they are more inclined to share LLM results publicly if they are correct. This works as a form of curation, where the users are selecting the correct 'solutions' before sharing them. Here, we want to study whether iteratively trained models can improve their planning capabilities having access only to their previously generated curated traces. We simulate the scenario just described in well-controlled environments of classical planning, and focus on the self-improving capabilities of LLMs. First, we prompt a base model to solve a diverse set of planning tasks, mixing together both short-horizon and harder long-horizon planning problems. Then, we discard traces that do not solve the task, mix the remaining into the original data, and fine-tune the next-gen model. The iteratively trained models essentially bootstrap each other's planning capabilities: each model attempts to generate solutions to the planning tasks, without relying on hand-designed prompts or on external planners. The simple plans solved in earlier generations are used as part of the training set of later generations, allowing the model to use these simple "building blocks" to solve more complicated planning problems.

In our experiments using Qwen3 4B [19], within five (deployment) generations, the latest model generation achieves more than *double* the base model performance in all tested domains. Figure 2 summarizes the results of our main experiment. In some cases, the performance increases by a factor of 5. Later generations of the model can also find much longer plans than the base model, showing that this mechanism allows for out-of-distribution generalisation. Moreover, there is no significant

difference in the average number of reasoning tokens produced by later generations, contrasting some results of RL fine-tuning [5].

We formally prove that iterative deployment is equivalent to a special case of REINFORCE [31] when the reward function is binary and traces are weighted according to importance sampling. This connection has two important implications: First, it highlights a big safety risk in the deployment of iteratively-trained models, as when the curation is done indirectly through user interactions during post-deployment, the next generation model would effectively be trained with an implicit reward function which can be difficult to control. That could have significant implications on the model's behaviour (e.g. the implicit reward could clash with safety training). Second, the mechanism highlighted here can be viewed as an alternative training regime to explicit RL, preserving generalisation but using a curation mechanism instead of defining an explicit reward function (which is often challenging for more open-ended, and less well-specified, tasks). The curation mechanism relies on post-processing validation as a signal: It works exclusively on the traces generated from prompts provided by the environment after deployment (e.g., user generated interactions with the LLM, or tool-usage traces). Our observations point the community to study the properties of these implicit reward functions as a top priority, especially as such iteratively-deployed AI systems are already being used through society.

Throughout the paper we refer to the starting model simply as the base model or as the *generation 0* of our model. Analogously, we call the method deployed after n iterations of our process as the n*-th generation* of the model.

2 Iterative Deployment Improves Planning Skills in LLMs

We study **Iterative Deployment**, a training mechanism that leads Large Language Models (LLMs) to bootstrap their planning capabilities without requiring external expert demonstrations or additional teacher models. The core intuition is that a model can improve by learning from the simple tasks it has already successfully solved, effectively using its own valid outputs as training data for subsequent generations (provided a reliable curation/verification mechanism is available).

We hypothesize that LLMs improve their planning abilities by simply fine-tuning over their own curated traces. In other words, they can bootstrap by first solving easy tasks themselves, and then being fine-tuned using the traces of these solved tasks. Later generations can then start solving larger tasks, and these new traces can be used so future generations solve even larger ones. Repeating this process many times can then gradually improve their planning skills. For example, if the current generation of a model can solve Sokoban problems with one single box, then it can learn from its own traces to solve problems with two boxes, and then with three boxes, etc. So by exploiting its own current capabilities, the model can improve and solve harder tasks in future generations.

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 $\mathcal{D}_{tasks}$ and a deterministic external validator $V(x, y)$ which returns true if trace y is a valid solution for task x, and false otherwise, with an ability to measure solution efficiency. This validator can be seen as a correction mechanism for reasoning tasks, or even simply as a proxy for user preferences. We evaluate model performance on $\mathcal{D}_{tasks}$ as a test-set as well. This is because we are interested to find how iterative deployment improves the model's ability to solve longer tasks with access only to its own previously curated solutions. The iterative deployment process then proceeds as follows:

- **Deployment and Trace Collection:** In each iteration n, we prompt the current model M_n to solve the tasks in $\mathcal{D}_{tasks}$. For each task input x, the model generates a trace y according to its policy $\pi_{\theta_n}(y|x)$. This trace includes the chain-of-thought and the tentative solution generated by the model. This emulates a standard deployment scenario where the model interacts with users or environments.
- **Validation:** The generated traces are passed to the external validator V. We filter the outputs to retain only the subset of valid traces, $\mathcal{D}^{(n)}_{valid} = \{(x, y)|V(x, y) = \text{True}\}$. Invalid plans, which certainly constitute the majority of outputs in early generations, are discarded.
- **Curation and Aggregation:** To prevent catastrophic forgetting, reduce model collapse, and further improve generalization, we aggregate the valid traces from the current generation

with those from all previous generations. The training dataset for the next step is $\mathcal{T}_{n+1} = \bigcup_{i=0}^{n} \mathcal{D}_{valid}^{(i)}$.

- During this aggregation, we apply a second curation step: a **selection mechanism** to ensure data quality. If multiple valid traces exist for the same task (e.g., from different generations), we retain only the highest-quality solution. In our experiments, quality is defined by plan efficiency – we select the trace with the shortest plan length, breaking ties by selecting the one with fewer reasoning tokens, but in principle other task-specific metrics can be used.
- **Supervised Fine-Tuning (SFT):** Finally, we produce the next generation M_{n+1} by fine-tuning M_n on the curated dataset $\mathcal{T}_{n+1}$ using the standard supervised learning objective (next token prediction).

The curation and aggregation phase is important here, as the likelihood of a user making an interaction with an LLM publicly available is not uniform, but depends on the nature of the interaction and the users' intention. For instance, when a user interacts with an LLM to solve a coding task, they are more likely to integrate a response that solves their tasks into their codebase. They are even more likely to use it if the solution is particularly elegant or simple, akin to our selection mechanism. This creates an effective curation mechanism controlled by the user's revealed preference (which is not necessarily their stated preference). This is a key difference to the assumptions used in the study of model collapse [23]. As LLMs are trained on more and more recursively generated data, their performance can deteriorate and the models eventually collapse. However, curation might delay or prevent this. In this work, we study the effect of this additional assumption compared to the model collapse assumptions, and argue for the importance of understanding its impact on future model generations.

Figure 1 illustrates how one iteration of this process works. When the n-th generation of an LLM is deployed, it is prompted by their users. The produced traces are filtered by an external *validator*, and those judged *valid traces* are used to fine-tune the $n+1$-th generation of the model. Note that we are *not* building a learning curriculum ourselves, but rather the LLM together with the validator are building one: we prompt the model with all tasks from the test set $\mathcal{D}_{tasks}$, discard the invalid traces, add the valid ones to the train set, and then fine-tune the next generation.

2.1 Formalizing the Connection to Reinforcement Learning

Iterative deployment can be interpreted as RL fine-tuning but with the reward signal left implicit. We prove next that SFT using only valid traces can be seen as a special case of REINFORCE [31] with an implicit binary reward function.

To show this, we start from the following result:

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

The proof is included in Appendix A.1.

In addition to the valid traces generated by the current model, we assume access to a collection of traces from previous generations of the model. We refer to the traces produced by current policy π_θ as the on-policy traces, and those produced by a behavior policy π_β (earlier generations or any other external source) as *off-policy* traces.

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 for Proposition 2 is also included in Appendix A.1.

In turn, this proves our original claim that SFT using only valid traces can be seen as a special case of REINFORCE [31].

Proposition 3. *SFT using only valid traces following the iterative deployment mechanism described in Section 2 is a special case of REINFORCE with implicitly defined binary rewards.*

Proof. Follows directly from Proposition 2. □

2.2 Implications to AI Safety

While iterative deployment shares links with RL fine-tuning, it also brings new concerns about AI safety. Unlike standard RL training, where reward functions can be explicitly designed to encode human preferences and safety constraints, iterative deployment of models in the wild relies on implicit signals from post-deployment usage. Curation done indirectly through user interactions with previously deployed models behaves as an opaque reward function which can be difficult to understand and control. In RL for alignment, reward functions are often used to align the model with safety constraints and goal specifications. In iterative model deployment, the implicit reward functions could clash with the explicitly specified rewards in model alignment. This could lead to unexpected problems in future model generations, as the indirect curation used might lead to large gradient steps in the opposite direction to the gradients induced by the safety training, for example. Overall, this raises new alignment challenges.

Another concern is bias during validation. If the external validator (e.g., a tool, or a human curator) has unintended or malicious biases, these biases might accumulate over the generations. Later models might then optimize for harmful properties that diverge from the original goals and also from safety constraints.

A final property to note is model collapse [23]. Data curation might delay model collapse by filtering for valid traces, it is still unknown whether this fully prevents collapse. If collapse occurs, important capabilities can degrade in ways that are difficult to detect until deployment. Note, however, that it is also unclear whether longer training periods using RL techniques, such as PPO or GRPO, lead to model collapse or not.

3 Empirical Validation

We evaluate iterative deployment on *classical planning* benchmarks: single-agent problems with deterministic actions in a fully-observable and discrete environment. We restrict ourselves to planning domains encoded using PDDL [15, 6], which is the description language used in planning competitions [26]. Classical planning using LLMs has been widely studied recently in different contexts [e.g., 27, 25, 12]. In *end-to-end planning* with LLMs, the model is prompt with a planning problem and asked for a solution—called a *plan*—for it. Shorter plans are usually preferred.

In the context of iterative deployment, we ask the model to solve a set of planning tasks. Then, we filter the traces that led to valid plans. These valid traces are then used for the fine-tuning of the next model generation. We include the valid traces produced by the previous generations as well.

But different model generations might solve the same tasks but with different traces or different plans. As we reuse valid traces from past generations, this means that the fine-tuning could contain several traces and plans for the same input prompt. We decided to have at most one training sample per task during fine-tuning, and so we use the following rules: [1]

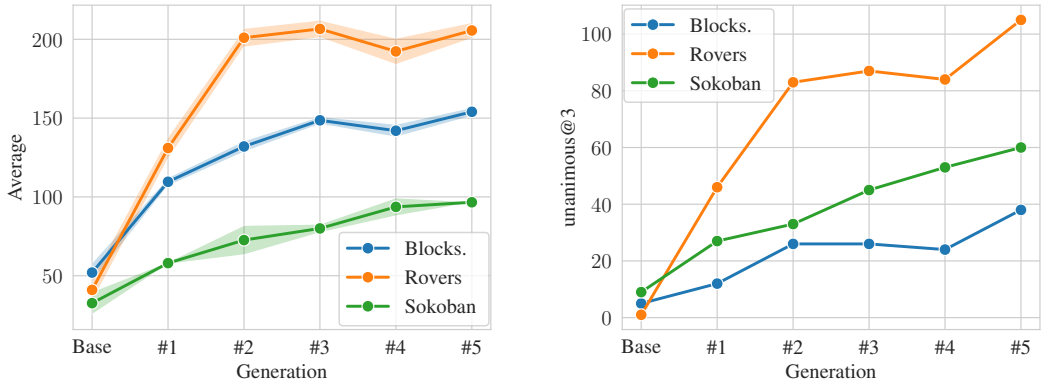
1. if there are several valid traces for the same task, we choose the one that yielded the shortest plan ('simplest', in number of steps)
2. if there is a tie, we select the one with the least amount of reasoning tokens

Additional experimental setup details are given in §A.2.

Benchmark. Our benchmark consists of three domains commonly used in classical planning. All three domains were recently used in the Learning Track of the International Planning Competition 2023 (IPC 2023) [26]:

Blocksworld: Several blocks are stacked on a table. The goal is to rearrange them to match a target configuration. The planning challenge comes from deciding which blocks to move first and which stacks must be "destroyed" in order to reach the target configuration. Any Blocksworld task can be solved with a simple polynomial strategy: put all blocks back on

[1] We experimented with alternative variations of these—use several training samples for the same task, breaking ties randomly—but these alternatives deteriorated performance.



(a) Average number of solved tasks

(b) Unanimous@3

Figure 3: Performance of different generations for the different domains tested.

the table and then rearrange them according to the goal. This guarantees that any task with n blocks can be solved within $2n$ steps.

Rovers: This domain is inspired by the 2003 Mars Exploration Rover (MER) missions and the planned 2009 Mars Science Laboratory (MSL) mission [14]. Rover robots (deployed in Mars) must perform tasks such as analyzing soil/rock, taking images with cameras (which may need calibration), and later communicating results-among waypoints and under visibility constraints. Plan existence for Rovers can be decided in polynomial time [7].

Sokoban: This is a PDDL version of the well-known puzzle: an agent in a grid must push boxes to their goal locations. The grid has several walls and obstacles that may prevent movement. The agent can only push but never pull boxes. This might lead to the case where boxes are "trapped" in corners which lead to dead-end states. Sokoban is **PSPACE**-complete [4], so plans might be exponentially long.

We generated 1000 instances per domain, varying the difficulty for each instance. This was done by varying specific domain parameters. Table 1 explains the most relevant parameter for each domain and shows their respective distributions. In this experiment, we train one model per domain; later, we show results for the case where the generations are fine-tuned over different domains.

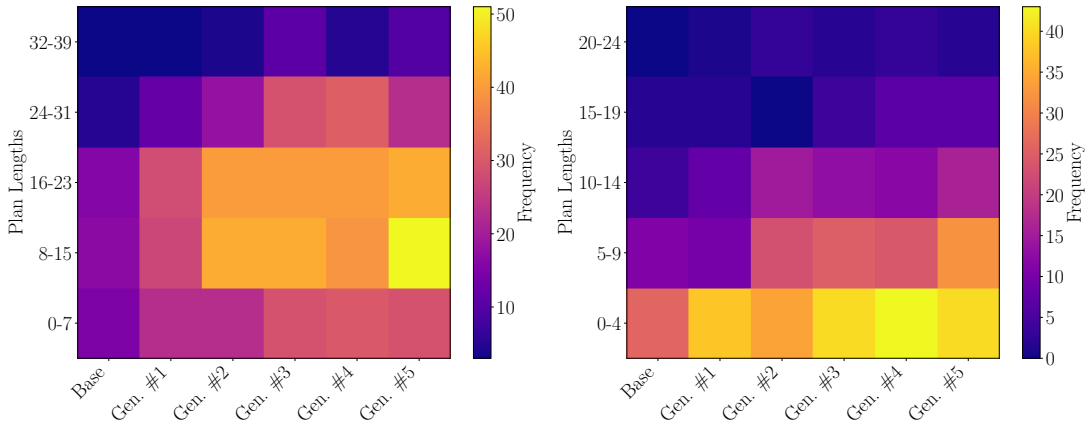
Domain	**Param. Dist.**	**Explanation**
Blocksworld	$n \in [2, 10]$	n blocks
Rovers	$r \in [1, 5]$	r rovers
Sokoban	$b \in [1, 7]$	b boxes

Table 1: Task distribution for each domain. Showing only the main parameters for each domain.

We emphasize that as we are analyzing how the model can self-improve, it is not relevant whether these domains were already seen during pre-training or not: our focus is on the relative increase in performance with respect to the base model and not to external baselines.

Metrics. The most relevant metric for our experiments is the number of solved tasks at each generation. This is our main estimate to know whether new generations are improving their planning skills or not. We analyze the average number of solved tasks over 3 complete runs, and also the number of tasks solved unanimously by all three runs, denoted as unanimous@3. Higher values for unanimous@3 indicate that the model is more robust, as it is more consistently able to solve certain tasks despite sampling noise during inference. We are also interested in the length of the computed plans, which represents the quality of the solutions, and the length of the reasoning traces, to see if our model's thinking behavior changes.

Results. Figure 3 shows the number solved tasks and unanimous@3 per domain for each generation. In all three domains, there is a substantial increase in performance for the first three generations. In



(a) Blocksworld

(b) Sokoban

Figure 4: Plan length frequency of different generations for the Blocksworld and Sokoban domain.

later generations, the speed of improvement decreases. After five generations, the average number of solved tasks in Blocksworld increased by 196%, in Rovers by 401%, and in Sokoban by 196%. Overall, this indicates that iterative deployment can bring a significant improvement in planning performance.

In later generations, the number of solved tasks fluctuates slightly between generations. But this is expected [e.g., 5], as the stochasticity in the inference process can cause tasks to be solved (or unsolved) simply by chance. Overall, still, the trend is clear: the last generations sustain better performance than earlier ones.

Later generations are also better at the unanimous@3 metric. In particular, the latest generation achieves the best value for all three domains. This shows that despite the slow down in performance in the later generations, the models are still learning to solve some tasks more consistently.

Appendix A.4 includes more details on these results.

Is the Model Only Learning Simple Tricks? One might wonder whether the models are learning to solve harder tasks or simply learning a few tricks (e.g., format answers correctly, avoid minor mistakes). Our experiments show that later generations are indeed solving longer horizon tasks. Figure 4 shows the plan length distribution for different model generations in the Blocksworld and Sokoban domains. It shows the distribution for one single pass over the benchmark. Both figures show a similar trend: 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.

Figure 5 shows the cumulative number of solved tasks per plan length for each generation for the Rovers domain. In this Figure, outliers are more visible than in Figures 4a and 4b. We can see that the last three generations find more plans with 10–30 steps, and generation 5 has significant outliers. However, when comparing the plans found by two different generations for a same task, there is no clear trend, and often the plans for a same task have the same lengths.

The last metric studied for this experiment is the amount of reasoning tokens produces by each generation. In RL fine-tuning, the model often increases the number of reasoning tokens produced as training progresses [5]. Figure 6 shows that this does not happen in our case. We average over both valid and invalid traces in this analysis. For Blocksworld and Sokoban, later generations have slightly fewer reasoning tokens in average, while for Rovers the result is the opposite. However, the difference between the base model and the last tested generation is not so significant: around 2000 tokens difference per domain. If we restrict the data to the valid traces, we still observe the same behaviour.

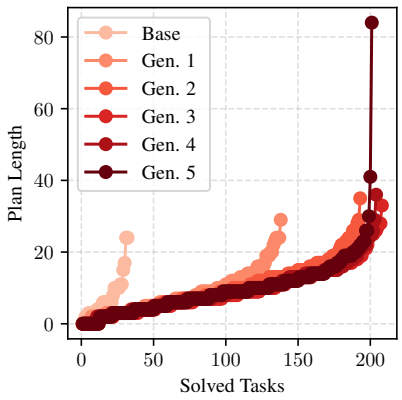


Figure 5: Performance across generations for Rovers. The y-axis represents the plan length of the plans found by each generation, while the x-axis shows the cumulative number of solved instances sorted by plan length.

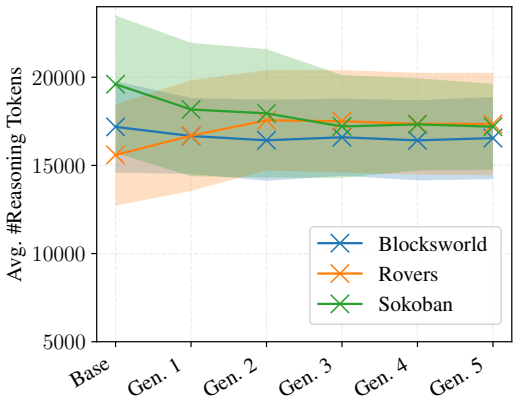


Figure 6: Average number of reasoning tokens per generation for each domain. Shaded area corresponds to one standard deviation of the mean. Considering both valid and invalid traces.

Does Curation Really Matter? We also studied how curation impacts our results. To analyze that, we repeated the experiment for the Blocksworld domain but *without* curation. In other words, we used all traces from previous generations to fine-tune the next one. (We do not include traces that reached the maximum token generation limit though.) Table 2 compares the average number of solved tasks for the case with and without curation. The version without curation also improves on the base model. While this might seem surprising at first, this is still expected: with more data, the version without curation is still able to acquire minor improvements, such as a better understanding of the PDDL syntax and more reliability at following instructions. The version with curation has a clear edge on the version without curation. This shows that, as we expected, the curation mechanism is crucial to achieve a significant improvement using iterative deployment. Furthermore, the version with curation requires less data. For example, to train the 5th generation of each model, the version with curation only used 356 traces, while the version with curation used 4017. So curation is capable of improving performance by 94% while using a fraction of the training data.

Gen.	w/ Curation	w/o Curation
Base	**52.0** $\pm$ **6.3**	**52.0** $\pm$ **6.3**
Gen. 1	**109.6** $\pm$ **2.8**	71.0 $\pm$ 3.2
Gen. 2	**132.0** $\pm$ **3.7**	82.0 $\pm$ 4.1
Gen. 3	**148.6** $\pm$ **2.6**	84.3 $\pm$ 4.9
Gen. 4	**142.0** $\pm$ **4.2**	84.0 $\pm$ 2.1
Gen. 5	**154.0** $\pm$ **2.9**	79.3 $\pm$ 5.2

Table 2: Average number of solved tasks in Blocksworld for iterative deployment with and without curation.

4 Related Work

Reasoning with LLMs. Chain-of-Thought (CoT) [29] uses "reasoning tokens" to improve the performance of LLMs in many problems. Many approaches build on this idea [e.g., 32, 21]s. For example, different methods explore multiple reasoning paths to find a valid solution [32, 24]. Others show that using more informative reasoning traces, such as those based on action sequence similarity, can also enhance planning performance [35]. However, these approaches often have limited success in generalizing to out-of-distribution reasoning tasks and longer-horizon planning problems. For instance, Stechly et al. [25] show that while CoT can help with planning problems, it does not consistently lead to generalization, particularly for out-of-distribution tasks.

Improving Reasoning Capabilities of LLMs. There are several methods in the literature that try to improve the reasoning capacities of LLMs [e.g., 34, 16, 30]. The iterative deployment mechanism we studied here shares conceptual similarities with the Self-Taught Reasoner (STaR) framework [33], which iteratively improves a model's reasoning ability by fine-tuning on traces that lead to valid

answers. However, while STaR tries to improve model performance, we focus on a very different question: we want to understand whether performance improvement could emerge *unintentionally* from repeated model deployments. We use planning problems to ground this question in a controlled setting which allows us to study this phenomenon, and prove that repeated deployment can be seen as a form of RL. On the technical level, there are key differences between the two as well. First, STaR has an extra-step for producing *rationales* for wrong answers, which are often not available from scraped user-curated traces. Second, we use valid traces from older generations as well as the current one emulating the property of web-scraped data containing traces from multiple former model generations, while STaR focuses on traces from the current model generation.

Reinforcement learning has been widely used to improve the reasoning capabilities of large language models [e.g., 22, 5]. Techniques such as group relative optimization (GRPO) allow us to fine-tune models without supervised data, using only an internal reward function [22]. By correctly modeling the reward functions, we can also influence the RL training so that the models better align with safety conditions or goal specifications.

Iterative deployment is also related to test-time scaling methods. For example, Muennighoff et al. [16] propose a simple test-time scaling technique where they fine-tune a small model on traces generated by a much larger, more capable teacher model—in their paper, the small model is a 32B model, while the teacher model is a model from the Gemini family. In contrast, iterative deployment does not rely on a separate, more powerful model. Instead, the LLM generates its own training data, bootstrapping its performance by solving progressively harder tasks. The model effectively becomes its own teacher, curating its own experience to improve in the next generations.

Model Collapse. One important connection to our analysis is 'model collapse'. Shumailov et al. [23] show that iteratively training models with their own synthetic output eventually collapses (i.e., the model's distribution shrinks so its tails disappear). The key difference in our analysis is the explicit curation step: only valid traces, as determined by an external validator, are used for fine-tuning. Shumailov et al. [23] lacks a curation step, and assumes all data is kept between generations. We show that curation can improve reasoning, but it is unknown whether the curation step completely avoids, or simply delays, model collapse. In our context, we have tested running our experiments above up to 10 generations and (despite the smaller improvement past the fifth generation) but we did not observe imminent hints of model collapse within planning domain.

Planning using LLMs. Valmeekam et al. [27] show that early LLMs do not reliably solve simple planning tasks. Kokel et al. [13] show that problems simpler than computing a plan, such as verifying if a given plan is valid, are already challenging for LLMs.

In the context of classical planning, several fine-tuning strategies have been explored. Rossetti et al. [20] train domain-specific GPT models for each planning domain, achieving strong performance but only for in-distribution problems. Bohnet et al. [1] report that supervised fine-tuning with optimal plans produced from an off-the-shelf planner [8] did not lead to out-of-distribution generalization. Huang et al. [11] apply RL to end-to-end plan generation, but observed only a small performance improvement. Our work differs from these, by showing that generalization can be achieved without access to an existing planner or to an explicit reward modeling, relying instead on an iterative process with a simple validation filter.

Verma et al. [28] propose an instruction-tuning framework that improves symbolic planning by teaching LLMs explicit logical CoT reasoning with feedback from VAL. While conceptually related to the iterative deployment mechanism, their method relies on carefully handcrafted training data and a multi-phase fine-tuning using structured reasoning feedback from a validator, whereas the mechanism studied here achieves self-improvement through repeated deployment and fine-tuning solely on valid traces, requiring just a binary signal.

5 Conclusion

In this work, we showed that iterative deployment of LLMs, where each generation is fine-tuned on curated traces from previous deployments, improves planning capabilities. Our experimental results on classical planning domains show that models trained under this paradigm more than doubled

their performance within five generations, with evidence of out-of-distribution generalization to longer-horizon plans.

Iterative deployment can be an effective alternative to RL to improve reasoning capabilities of LLMs. We proved theoretically that our method is equivalent to a special case of REINFORCE with binary rewards. Unlike RL fine-tuning, our approach does not rely on carefully designed reward functions. Instead, it uses external postprocessing tools to validate/curate LLM answers as an implicit signal. However, the absence of an internal reward function brings concern about potential biases and unintended effects caused during training. The use of external validators, that might be outside our control, can be damaging to AI safety.

Overall, our results suggest that iterative deployment is a viable method for self-improving LLMs. In future work, we plan to study how model collapsing [23] can affect iterative deployment, and also to develop theoretical results to connect our method with RL fine-tuning more explicitly.

References

[1] 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.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the Thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS 2020)*, pages 1877–1901, 2020.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[9] 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, editors, *Proceedings of the ICAPS 2003 Workshop on the Competition: Impact, Organisation, Evaluation, Benchmarks*, 2003.

[10] 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.

[11] 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, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2025)*. AAAI Press, 2025.

[12] Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi. Thought of search: Planning with language models through the lens of efficiency. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, pages 138491–138568, 2024.

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

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

[15] 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.

[16] 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.

[17] OpenAI. GPT-4 technical report. arXiv:2303.08774 [cs.CL], 2024.

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

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

[20] 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, editors, *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, pages 500–508. AAAI Press, 2024.

[21] Bilgehan Sel, Ahmad Al-Tawaha, Vanshaj Khattar, Ruoxi Jia, and Ming Jin. Algorithm of thoughts: enhancing exploration of ideas in large language models. In *Proceedings of the 41st International Conference on Machine Learning*. JMLR.org, 2024.

[22] 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.

[23] 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.

[24] Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more ffective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025.

[25] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of thoughtlessness? an analysis of CoT in planning. In *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, pages 29106–29141, 2024.

[26] 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.

[27] 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)*, pages 75993–76005, 2023.

[28] 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.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the Thirty-Sixth Annual Conference on Neural Information Processing Systems (NeurIPS 2022)*, pages 24824–24837, 2022.

[30] 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.

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

[32] 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. In *Proceedings of the Thirty-Seventh Annual Conference on Neural Information Processing Systems (NeurIPS 2023)*, pages 11809–11822, 2023.

[33] 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.

[34] 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.

[35] Xinran Zhao, Hanie Sedghi, Bernd Bohnet, Dale Schuurmans, and Azade Nova. Improving large language model planning with action sequence similarity. In *International Conference on Learning Representations*, 2025.

A Appendix

A.1 Proofs

In this section we present the proofs for our the theoretical results of Section 2.1. We use the same notation and terminology from the paper.

Proposition 1. *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).

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 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 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.*

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 is equivalent to REINFORCE with binary rewards augmented by importance-weighted contributions from the behavior policy.

The SFT objective over all valid trace 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 [18]. 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

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

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 trace, 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 corresponds to REINFORCE with binary rewards augmented by importance-weighted contributions from the behavior policy. In turn, this proves our original claim that SFT using only valid traces can be seen as a special case of REINFORCE [31]. □

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 [19]. Specifically, we use the Qwen3 4B Thinking 2507 version, which was better at following formatting instructions during our early experiments. We use temperature 0.6 and a maximum context length of 32 768 tokens during inference. To fine-tune our models, we use low-rank adaptation (LoRA) [10]. The hyperparameters used for fine-tuning can be found in Appendix A.3. To avoid stacking up or retraining existing adapters, when we train generation n, we fine-tune the base model Qwen3 4B with the traces from generations $0, 1, 2, \ldots, n-1$. To validate the computed plans, we use VAL [9], which is commonly used in planning competitions.

Prompt. Our prompt format is based on the prompt used by Corrêa et al. [3] 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.

A.3 Fine-Tuning Hyperparameters

Below we include the hyperparameters used during our supervised fine-tuning with LoRA. At each generation, we randomly select 10% of the traces to be used as validation set. The validation loss is evaluation every 10 steps.

Domain	Base	#1	#2	#3	#4	#5
Blocksworld	52.0 ± 6.3	109.6 ± 2.8	132.0 ± 3.7	148.6 ± 2.6	142.0 ± 4.2	$\mathbf{154.0 \pm 2.9}$
Rovers	41.0 ± 7.7	131.0 ± 7.2	201.0 ± 6.1	$\mathbf{206.6 \pm 5.7}$	192.33 ± 8.5	205.6 ± 5.2
Sokoban	32.6 ± 7.4	58.0 ± 0.8	72.6 ± 9.5	80.0 ± 2.9	93.67 ± 5.9	$\mathbf{96.6 \pm 0.4}$

Table 3: Average number of solved tasks per generation of the model. Average over three runs. Total of 1000 tasks per domain. Best value for each domain is highlighted.

Domain	Base	#1	#2	#3	#4	#5
Blocksworld (1000)	5	12	26	26	24	**38**
Rovers (1000)	1	46	83	87	84	**105**
Sokoban (1000)	9	27	33	45	53	**60**

Table 4: Number of solved tasks in unanimous@3 per generation of the model. Best value for each domain is highlighted.

Hyperparameter	**Value**
Maximum Context Length	32 768
Batch Size	1
Gradient Accumulation Steps	1
Epochs	2
Optimizer	AdamW (fused)
Learning Rate	1×10^{-5}
Learning Rate Scheduler	Cosine
Warm-up Ratio	0.02
Weight Decay	0.01
Max. Gradient Norm	1.0
LoRA Rank	16
LoRA Alpha	32
LoRA Dropout	0.05
LoRA Bias	None

A.4 Additional Results

Table 3 shows the number solved tasks per domain for each generation of our process. Table 4 shows the unanimous@3 for each generation.

A.5 Other IPC 2023 Domains

We briefly explain the seven domains used in our second experiment with 10 different domains. The other three domains — Blocksworld, Rovers, and Sokoban — where explained in the text. Table 5 shows the distribution of the main parameters for each domain used in this experiment.

Childsnack This domain involves making a number of sandwiches, some with gluten-free ingredients for children with allergies, placing them on trays, and delivering them to tables. Key constraints include production (gluten and non-gluten items), packaging (quantity of trays), and serving order.

Ferry Cars are distributed across locations, and a ferry (which can fit one car) must transport them to their target destinations. The domain requires deciding the ferry's schedule, routing, and when to move each car, all under capacity constraints.

Domain	Param. Dist.	Param. Meaning
Blocksworld	$n \in [2, 10]$	n blocks to be rearranged
Childsnack	$c \in [4, 12]$	c children to be served
Ferry	$a \in [3, 30]$	a cars to be transported
Floortile	$t \in [2, 25]$	t tiles to paint
Miconic	$p \in [1, 10]$	p passengers to deliver
Rovers	$r \in [1, 4]$	r rovers to drive and collect items
Satellite	$s \in [1, 8]$	s satellites to synchronize
Sokoban	$b \in [1, 4]$	b boxes to push to their goals
Spanner	$k \in [1, 20]$	k spanners to collect
Transport	$v \in [2, 10]$	v vehicles to drive

Table 5: Task distribution and plan lengths for each domain. Showing only the main parameter for each domain.

Floortile A grid of tiles must be painted by robots that can move in four directions via unpainted tiles. Robots can only move over tiles that are not yet painted, so they must paint tiles in a certain order.

Miconic Elevator domain: there are multiple floors, passengers on origin floors, and destinations; an elevator must move and pick up / drop off passengers, respecting boarding and alighting constraints. Though conceptually simple, it tests planner efficiency regarding sequencing and movement decisions, especially as passenger numbers and floor counts increase.

Satellite Satellites have instruments (which may need calibration) and different modes of observation; they must orient, turn, switch instruments, and gather images to satisfy a given goal. Only one instrument can be active at a time, and switching or calibrating requires actions.

Spanner An agent traverses a path where there are k spanners place along this path. At the end of the path there exists a gate with $m \leq k$ loose nuts. The agent must collect enough spanners to tighten all nuts and cross corridors to be able to cross the gate. Spanners break when used, thus the agent must collect enough spanners to open the gate.

Transport This is a traditional logistics problem with capacity constraints. Vehicles must deliver packages to certain locations on a graph, subject to vehicle capacities (i.e., number of packages that can be transported at once).