Chasing Progress, Not Perfection: Revisiting Strategies for End-to-End LLM Plan Generation

Sukai Huang, Nir Lipovetzky and Trevor Cohn*

School of Computing and Information Systems, The University of Melbourne, Australia
sukaih@student.unimelb.edu.au, {nir.lipovetzky, trevor.cohn}@unimelb.edu.au

Abstract

The capability of Large Language Models (LLMs) to plan remains a topic of debate. Some critics argue that strategies to boost LLMs' reasoning skills are ineffective in planning tasks, while others report strong outcomes merely from training models on a planning corpus. This study reassesses recent strategies by developing an end-to-end LLM planner and employing diverse metrics for a thorough evaluation. We find that merely fine-tuning LLMs on a corpus of planning instances does not lead to robust planning skills, as indicated by poor performance on *out-of-distribution* test sets. At the same time, we find that various strategies, including *chain-of-thought*, do enhance the probability of a plan being *executable*. This indicates progress towards better plan quality, despite not directly enhancing the final *validity rate*. Among the strategies we evaluated, reinforcement learning with our novel 'Longest Contiguous Common Subsequence' reward emerged as the most effective, contributing to both plan executability and validity. Overall, our research addresses key misconceptions in the LLM-planning literature; we validate incremental progress in plan executability, although plan validity remains a challenge. Hence, future strategies should focus on both these aspects, drawing insights from our findings.

1 Introduction

While latest Large Language Models (LLMs) have shown promise in areas like grade school math and code generation (Liu et al. 2023; Ye et al. 2024; Kumar et al. 2024; Madaan et al. 2024), their effectiveness in solving planning tasks remains a contentious issue. Numerous studies have voiced skepticism, questioning whether LLMs can truly conduct deliberative reasoning beyond statistical pattern matching (Huang et al. 2023; Kambhampati et al. 2024; Valmeekam et al. 2024b). However, the landscape is not uniformly skeptical, some studies have also made provocative claims that LLMs, when fine-tuned on paired datasets of planning problem descriptions and their corresponding plans, can generate correct plan sequences for new problems within the same domains (Shah et al. 2024; Rossetti et al. 2024).

Existing literature on *LLMs planning* shows a clear dichotomy, likely due to limitations in evaluation methodologies on both sides of the debate, as shown below:

1. Lack of OOD Evaluation: Optimistic studies often overlook out-of-distribution (OOD) evaluation, leading to models potentially simply recalling and adapting partial plan traces from the training data, rather than demonstrating genuine sequential reasoning (Mirzadeh et al. 2024). For instance, Shah et al. (2024) evaluated their model on Sudoku puzzles of the same size as those in the training set, potentially allowing the model to reuse familiar grid patterns. Similarly, Rossetti et al. (2024) evaluated their fine-tuned LLM planner on test instances generated using the same PDDL generator configurations as the training data. This lack of OOD evaluation can lead to an overestimation of the model's planning capabilities.

2. Insufficient Analysis of Failure Modes: Pessimistic studies (e.g., Valmeekam et al. (2024b)) typically focus solely on plan *validity*, an extremely stringent criterion that requires the entire plan to be perfectly correct. This restrictive metric fails to capture incremental improvements in planning capabilities. Moreover, these studies rarely conduct diagnostic experiments to identify specific reasoning bottlenecks, leading to a puzzling enigma in the planning community – Why do strategies that enhance reasoning capabilities in other tasks (e.g., math-solving (Wei et al. 2022)) appear ineffective in planning tasks? For instance, Stechly et al. (2024) found that chain-of-thought prompting did not improve the validity rate of LLM planners. By not considering more granular metrics and failing to investigate where the strategy fails, researchers miss opportunities to understand how different strategies contribute and which aspects of reasoning need the most attention.

To address these gaps, we **reassess** various strategies for training LLMs in *end-to-end plan generation*, where the LLM itself functions as a **black-box planner** with **no** explicitly modeled *search process*. These strategies include *permutation* (Allen-Zhu and Li 2023), *chain-of-thought* (Wei et al. 2022; Yao et al. 2024), *self-correction* (Madaan et al. 2024; Ye et al. 2024; Kumar et al. 2024), and *reinforcement learning* (RL) (Liu et al. 2024). Our evaluation extends beyond the plan *validity* metric to include plan *executability*[1] and employs a comprehensive suite of diagnostic experiments to identify where each strategy succeeds or

*Now at Google DeepMind

[1]An executable plan is one where the preconditions of all actions are satisfied by the state at execution (See Appendix A).

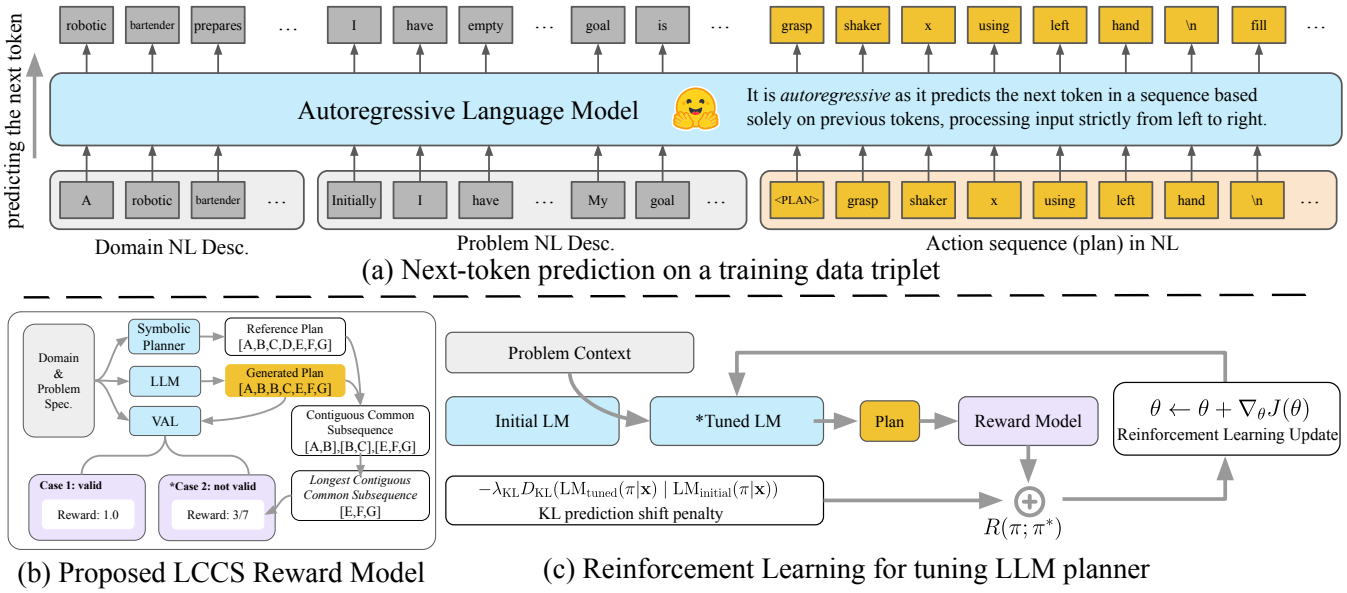

Figure 1: *(a) Next-token prediction:* The LLM is trained to predict the next token in corpus containing domain and problem details and their plans, proceeding left-to-right. (b) *Proposed LCCS Reward:* We use the length of LCCS as an auxiliary reward signal for RL. It provides granular feedback to fill the sparse reward gap. (See Section 3.2). (c) *Reinforcement Learning:* RL with LCCS Reward is shown to be the most effective strategy for enhancing end-to-end LLM planners among all tested strategies.

fails. Moreover, OOD test sets are deliberately designed to evaluate the model's generalization capabilities in planning.

We hope our work will address misconceptions in the LLM planning literature and provide a better understanding of the inherent planning capabilities of LLMs' *next-token prediction* mechanism. Our contributions are as follows: (1) We challenge the claim that *fine-tuning* LLMs simply on datasets containing problem contexts and reference plans acquire robust planning skills, by demonstrating their failure on OOD test sets. (2) We show that strategies like *CoT* lead to incremental improvements in plan quality by enhancing plan *executability*, even if they do not directly increase *validity* rates. (3) We show that RL with our proposed 'LCCS' reward emerges as the most effective strategy. In particular, it improves plan *validity* by 7% and *executability* by 9% in longer planning problems.

2 Background & Related Work

Classical Planning. We assume that readers are familiar with the standard planning language PDDL for representing deterministic, fully observable planning problems. A classical planning problem is a pair $P = \langle D, I \rangle$, where D represents the planning domain, consisting of a set of predicate symbols and action schemas; I denotes the problem instance, which includes the objects, initial state and goal. A sequence of grounded actions $\pi = (a_0, a_1, \ldots, a_n)$ is a *valid* plan *iff* it is *executable* and the goal G holds in the final state after executing the plan.

Next-token prediction. We fine-tune an end-to-end LLM-planner, LM_θ, using *next-token prediction* on a text sequence $\mathbf{x} = \langle \mathcal{Z}(D), \mathcal{Z}(I), \mathcal{Z}(\pi) \rangle$, where $Z(\cdot)$ refers to the serialization of the PDDL elements into natural language (as illustrated in Figure 1 part a). Let $\mathbf{x}_{<i}$ denote the first $i-1$ tokens of sequence $\mathbf{x}$, and x_i be the i-th token. Then, $\text{LM}_\theta(\hat{x}_i = x_i \mid \mathbf{x}_{<i})$ indicates the probability the model predicts the i-th token $\hat{x}_i$, to be equal to the actual token x_i, conditioned on the preceding tokens $\mathbf{x}_{<i}$. This is also known as *autoregressive modeling*. The training objective is to maximize the likelihood of the joint distribution of the training corpus $\mathcal{X}$, expressed as follows: $J(\theta) = \max_\theta \mathbb{E}_{\mathbf{x}\sim\mathcal{X}} \left[\sum_{i=1}^{|\mathbf{x}|} \log \text{LM}_\theta(\hat{x}_i = x_i \mid \mathbf{x}_{<i}) \right]$. Note that this approach trains the model to predict not only tokens in the output (i.e., the generated plans) but also tokens in the input query (i.e., the domain and problem description).

Scope. We examine the planning skills of LLMs in the paradigm of **end-to-end plan generation**. Our focus excludes certain approaches, such as the LLM-Modulo framework, which relies on an external verifier to validate generated plans (Kambhampati et al. 2024), nor the Thought of Search (Katz et al. 2024; Cao et al. 2024), which bypasses direct long-term planning by prompting LLMs to generate search functions instead of actual plans. We focus solely on paradigms that utilize basic next-token prediction during inference. This excludes hybrid models, such as AlphaMath (Chen et al. 2024), which explicitly model *search process* via Monte Carlo Tree Search (MCTS) to derive solutions. The most closely related work to ours is the *PlanGPT* model (Rossetti et al. 2024). They demonstrated that fine-tuning LLMs to predict next tokens on plan sequences, without access to the entire search trace, can effectively teach the model to generate plans for new problems.

However, our work differs in two key aspects: (1) We di-

rectly use natural language descriptions instead of PDDL snippets to train the model. This approach aligns more closely with how we expect to interact with LLMs in real-world scenarios. (2) As mentioned in § 1, we conduct a more comprehensive evaluation to examine the quality of the generated plans and identify the model's reasoning bottlenecks.

Criticism on LLM Planning. Criticisms of LLMs in planning tasks stem from both theoretical analysis and empirical observations. Theoretically, LLMs are machine learning-based *probabilistic models*, and the accuracy of the models' predictions decays exponentially over the length of the sequence, a phenomenon described as "Snowballing error due to autoregressive modeling" (Bachmann et al. 2024). For instance, even with a 99% correctness rate per step, the probability of a correct sequential prediction drops to about 36.6% over 100 steps. Therefore, one has to admit that there is no guarantee of soundness in long-term planning tasks due to its probabilistic nature.

Empirically, numerous studies have demonstrated the struggles of off-the-shelf LLMs in predicting valid plans for long-term tasks (Valmeekam et al. 2022, 2023; Huang et al. 2023). Even advanced proprietary models like OpenAI o1, which was designed for reasoning tasks, fail on long-term planning (Valmeekam et al. 2024b). In contrast to the pessimistic findings in the literature, we seek to better understand why LLMs often fail in planning tasks and provide insights into how to improve their planning capabilities.

3 Methodology & Experimental Design

3.1 Extended PlanBench Dataset

PlanBench (see Figure 2), as introduced by Valmeekam et al. (2024a), has been the most widely used benchmark for evaluating planning capabilities of LLMs. It provides a *template* to convert symbolic model information into natural language text that can be used to either train or test LLMs on plan generation. An innovation of PlanBench is its "obfuscated" planning domains, where names are replaced using misleading vocabulary (see Appendix C.2). This aims to assess whether LLMs can plan based solely on the logical structure of the planning problem without any linguistic cues.

To better support both training and evaluating LLMs' planning skills, we extend the dataset in two key ways[2]:

1. New Domains. We expand PlanBench to include additional domains from the IPC benchmarks, increasing the total from 2 to 8 domains. This allows for evaluating the model across diverse domains and complexities, providing a comprehensive assessment of its planning skills.

2. Longer-plan Problems. A significant limitation observed in *PlanGPT* (Rossetti et al. 2024) was that both the training and test data shared the same distribution, a limitation discussed in § 1. To address this, we have deliberately generated test instances with *longer plan-lengths*, which is arguably the most straightforward way to generate OOD performance. By doing it, we prevent LLMs from easily retrieving familiar plan trace patterns from the training data, forcing them to engage in actual planning.

[2]The dataset will be released via Huggingface Hub.

PlanBench Blockworld Example

```
Query:
[CONTEXT]
I am playing with a set of blocks where I need to
arrange the blocks into stacks.
[ACTION DESCRIPTION]
Here are the actions I can do
Pick up a block
Unstack a block from on top of another block
...
I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
...
[STATEMENT]
The initial state is that, the orange block is clear,
...
My goal: the blue block is on top of the orange block.
```

```
Response:
My plan is as follows:
[PLAN] unstack the orange block from top of the blue
block \n put down the orange block \n ...
[PLAN END]
```

Figure 2: Each planning problem instance in PlanBench is serialized into a single text block that presents the context, action description, initial and goal states, and the plan.

The updated benchmark consists of:

1. Train Set: 4000 instances per domain, with plan lengths from 3 to 16 actions. This contrasts with the previous training set, which consisted of 70,000 instances and had no restrictions on plan length.

2. Test Sets: 200 distinct instances per domain, all of which are not present in the training set, categorized into four groups: **(a) In-Distrib**: same plan length distribution as the training data. **(b) Long** (*new category*): longer plan length distribution ranging from 17 to 32 actions. **(c) Unseen** (*new category*): two novel domains not seen and not trained on by the model. **(d) Obfuscated**: two domains with obfuscated predicates, actions and objects.

To prevent the model from reusing memorized plan traces and achieving artificially high performance through pattern matching, we limit the training data to 4000 problem instances per domain. While the data size might still seem substantial, it is important to recognize that deep learning models, particularly modern LLMs, are data-hungry and are typically trained on trillions of tokens. This size represents only 5.7% of the data used in the *PlanGPT* work.

3.2 Strategies for Enhancing Reasoning

This section presents four strategies aiming to enhance LLMs' reasoning skills: *permutation augmentation*, *chain of thought*, *self-correction*, and *reinforcement learning*. We briefly explain the rationale and adaptation of each strategy to the planning domain.

Permutation Augmentation in Query Content

Original:	+ Permutation:
Query:	Query:
...	...
[ACTION DESCRIPTION]	[ACTION DESCRIPTION]
Action a	Action c ←
Action b	Action a ←
Action c	Action b ←
...	...
Action a's precon 1	Action a's effect 2 ←
Action a's precon 2	Action c's precon 1 ←
Action a's effect 1	Action b's effect 1 ←
Action a's effect 2	Action a's precon 2 ←
...	...
[STATEMENT]	[STATEMENT]
Initial state:	Initial state:
1. orange block is clear	1. hand is clear ←
2. hand is clear	2. orange block is clear ←
...	...
My goal is that:	My goal is that:
1. red on white	1. blue on orange ←
2. blue on orange	2. red on white ←
...	...

Figure 3: Permutation augmentation strategy shuffles the order of action descriptions (red arrows), condition and effect descriptions (blue arrows), and atoms in initial and goal statements (green arrows). The model is expected to learn underlying semantics through more diverse data representation, avoiding overfitting to superficial patterns.

CoT Prompts to the Plan Response

+ Goal CoT:	+ State CoT:
Response:	Response:
My plan is as follows:	My plan is as follows:
[PLAN]	[PLAN]
[GOAL]	[PRECON]
# Repeat the goal	# Provide grounded
My goal is ...	conditions for the action
[GOAL END]	I can only pick up a
[COUNT]	yellow block if the yellow
# Count the remaining step	block is on the table and
to the goal	the yellow block is clear.
10	...
[COUNT END]	[PRECON END]
put down the orange block	put down the orange block
# Next step in the plan	[EFFECT]
[GOAL]	# Provide grounded effects
My goal is ...	Once I put down a yellow
[GOAL END]	block, my hand becomes
[COUNT]	empty.
9	[EFFECT END]
...	...
[PLAN END]	[PLAN END]

Figure 4: Two types of CoT prompts are used in the plan response – *Goal CoT* and *State CoT*. *Goal CoT* prompts the agent to repeat the goal and count the remaining steps to the goal. *State CoT* prompts the agent to provide grounded conditions for the action and grounded effects. The model is expected to learn the world dynamics through these prompts.

Permutation Augmentation Early studies have shown that data augmentations enhancing input diversity can improve a model's generalization performance (Cubuk et al. 2019; Gontijo-Lopes et al. 2020). Recently, Allen-Zhu and Li (2023) discovered that permutation augmentation – randomly rearranging sentences in a query – significantly improved *Transformer* models for question-answering tasks. They posit that exposing models to varied expressions of the same knowledge encourages them to discover the underlying structure rather than overfitting to superficial word patterns. We hypothesize a more specific rationale for why permutation augmentation works: it enhances the *self-attention* mechanism in *Transformer* during training. By permuting sentences, it ensures that when a token *attends* to others, the attended tokens do not consistently occupy the same positions. This prevents the *attention* mechanism from exploiting shortcuts based on positional or syntactic cues, which could potentially skew the model's semantic understanding. Instead, permutation encourages each token to engage with semantically relevant tokens, enhancing the quality of each token's embedding. These improved embeddings enhance the model's overall ability to predict the answers.

As such, we apply it to the planning instance descriptions in the training data. Specifically, we permute the order of action descriptions and statement sentences, as illustrated in Figure 3. These permuted variants remain semantically equivalent, similar to how any permutation of predicates within conjunctions or disjunctions in PDDL expressions preserves their semantic meaning.

Chain of Thought (CoT) Producing intermediate steps before directly predicting the final output, is the core idea behind CoT prompting, a strategy that has shown promise in eliciting LLMs' reasoning, as demonstrated by numerous studies (Wang et al. 2023a; Shi et al. 2023; Yao et al. 2024). Complementing these empirical studies, Prystawski, Li, and Goodman (2023) provided a theoretical rationale for the effectiveness of chain-of-thought reasoning. Using a controlled Bayesian network environment, they showed that generating intermediate variables before predicting the target variable can improve a model's ability to match true conditional probabilities, particularly when relevant local variables are observed near each other in the network.

To design general CoT prompts for planning tasks, we draw inspiration from Decision Transformer (Chen et al. 2021), a RL work that uses LMs as policy models for Atari Benchmark. Both RL and planning address sequential decision problems, sharing key elements such as initial states, state transition functions, and reward/heuristic functions. In RL, Decision Transformer conditions action prediction on the provided remaining *return* and *state* information. Recognizing this common ground, we structured our CoT prompts to also include the information of the goal distance and the current state. We posit that training LLMs to predict these details before deciding on an action would help the model

Response with Erroneous Steps and Correction

```
+ Self Correction:
Response:
My plan is as follows:
# Assume the length of this reference plan is 7.
[PLAN]
1st action; 2nd action;
5th action [WRONG]; # Synthesized mistakes by selecting
an action that appears later in the plan
3rd action; 4th action;
7th action [WRONG]; # [WRONG] is special 'removal' token
5th action;
6th action; 7th action
[PLAN END]
```

Figure 5: Self-correction learning modifies plan responses to involve erroneous steps followed by their immediate corrections, with the incorrect step indicated by token `[WRONG]`.

learn world dynamics better and thus make better decisions.

But, in our context, serializing fully-observable states directly into natural language can lead to memory explosion, potentially overwhelming the model's capacity. Thus, we instead represent states using transition information (see Figure 4). In fact, by estimating these details, LLMs perform the roles of both state transition and heuristic functions in classical search algorithms. An orthogonal investigation by Katz et al. (2024) also explored leveraging LLMs to generate state transition and heuristic function code for planning tasks, but essentially not examining the plan generation skills.

Self-Correction Learning This strategy has shown effectiveness especially in grade school math: Kumar et al. (2024) demonstrated that a multi-turn online reinforcement learning approach, which trains the model to correct its own mistakes, improves the model's accuracy. Similarly, Ye et al. (2024) showed that training LLMs on data containing errors and their "immediate removal" can lead to higher accuracy compared to training on error-free data alone. Several theoretical frameworks support the effectiveness of self-correction learning, with the most intuitive being *contrastive learning*. By exposing the model to both mistakes and correct solutions, we establish a robust learning environment. In contrast, training solely on ground truth data can lead to *exposure bias*, where the model becomes overly confident in its predictions due to a limited range of correct examples. This overconfidence can hinder the model's ability to generalize effectively to unseen data (An et al. 2022).

This strategy has been largely overlooked in previous studies on training LLMs for plan generation. To explore its impact, we adopt an approach similar to Ye et al. (2024), designing synthesized error-correction procedure in the planning steps. Specifically, we randomly select an action that appears later in the plan sequence and insert it to the current step with a special 'removal' token, as illustrated in Figure 5.

Reinforcement Learning (RL) More recently, Liu et al. (2024) found that when applying RL specifically to reasoning tasks, such as math and code generation, yields superior performance gains compared to applying RL to more general tasks. This suggests that RL could be a particularly well-suited strategy for enhancing LLM planning skills.

However, applying RL training for LLM-based plan generation poses two key challenges: (1) Typical RL frameworks for LLM are designed to attribute rewards to individual tokens as they are generated. However, in plan generation, the validity of a plan can only be determined once the entire response is complete, which creates a mismatch with the token-level optimization paradigm. (2) Using plan validity as the reward signal – assigning $+1$ only when plans are valid, and 0 rewards otherwise – results in very sparse feedback. Such binary and infrequent rewards often lead to poor performance in RL algorithms (Ecoffet et al. 2019).

To address the first challenge, we leverage the recently proposed *rloo* framework by Ahmadian et al. (2024). *rloo* enables LLMs to optimize at the sequence level, providing delayed rewards associated with the entire output rather than individual tokens. This approach aligns better with planning tasks, where plan validity can only be assessed upon completion of the entire sequence.

To address the second challenge and create a more informative reward signal, we propose using the *Longest Contiguous Common Subsequence (LCCS)*. Let P_g be the generated plan and P_r be the reference plan. We define the LCCS reward function as:

$$R(P_g, P_r) = \begin{cases} 1 & \text{if } P_g \text{ is valid} \\ \frac{|\text{LCCS}(P_g, P_r)|}{|P_r|} & \text{otherwise} \end{cases} \quad (1)$$

When the plan is valid, we continue giving a reward of $+1$. However, when the plan is invalid, a supplementary reward is also provided based on the length of the LCCS between P_g and P_r (also see Figure 1 part b). Note that the current reward system has an inherent bias because it relies on a single reference plan, whereas there may be multiple valid plans for a given problem. However, this limitation can be effectively addressed in future work by considering distances to multiple reference plans from top-k planners, potentially providing a more robust measure. Despite this limitation, we found that *LCCS* serves as a good reward signal to fill the sparsity gap and provide granular feedback on the quality of the output. As such, it offers smoother gradients for the model to learn from, facilitating more effective training.

4 Evaluation Results

Model We fine-tuned the open-source LLM QWEN2-7B-INSTRUCT (Yang et al. 2024) on the extended PlanBench dataset. This model was pre-trained on general text but not on code. Note that QWEN2's architecture shares significant similarities with LLAMA 3 (Dubey et al. 2024), but was trained on a smaller dataset. It thus reduce the risk of having exposed to PlanBench or related data during pre-training. See training configurations and other details in Appendix B.

Evaluation Metrics In previous work, the assessment of LLM planners primarily focused on the *validity rate* of the generated plans (Kambhampati et al. 2024; Stechly et al.

Table 1: Performance of the fine-tuned LLM across various test sets with no additional strategies applied. Although the LLM performs well on in-distribution, it struggles to generalize to OOD cases.

Domain	In-Distrib.		Long	
	valid. rate	*exec. rate*	*valid. rate*	*exec. rate*
BLOCKSWORLD	98.5%	98.5%	13.5%	23.5%
LOGISTICS	**100%**	**100%**	14.0%	20.5%
BARMAN	**100%**	**100%**	25.0%	43.5%
CHILDSNACK	**100%**	**100%**	66.0%	67.0%
DEPOTS	98.5%	98.5%	31.0%	37.0%
DRIVERLOG	**100%**	**100%**	31.0%	50.7%
GRIPPERS	99.0%	99.0%	50.5%	76.0%
SATELLITE	99.0%	99.2%	51.5%	53.0%
Domain	**Unseen**			
	valid. rate		*exec. rate*	
HANOI	**0%**		35%	
STORAGE	**0%**		**1%**	
Domain	**Obfuscated**			
	valid. rate		*exec. rate*	
BLOCKSWORLD	**0%**		**0%**	
LOGISTICS	**0%**		**0%**	

2024). However, the validity rate alone is inadequate for differentiating between the performances of various strategies, as it is too stringent for evaluating incremental improvements in plan quality. Relying solely on this metric can obscure deeper insights into the planning process, preventing a thorough understanding of where the model faces reasoning bottlenecks and how to address them. To address this limitation, we introduce a complementary metric: *executability rate*. A plan is deemed *executable* if it satisfies all preconditions for the actions within the current state. This metric allows for a finer evaluation of plan quality, even if the plan does not completely achieve the goal state. To prevent trivially empty plans from being considered executable, we only count plans with more than 3 actions in the evaluation.

In the following sections, we present our experimental results, focusing on the effectiveness of the four strategies for enhancing LLM planning capabilities. We evaluate performance in both in-distribution and out-of-distribution test sets to provide a comprehensive assessment. To begin with, we establish a baseline by fine-tuning the LLM on the vanilla corpus. During the training phase, the model is tasked with predicting the next tokens in serialized planning instances consisting of domain descriptions, problem instance descriptions, and the associated plans. Then, in the testing phase, we prompt the model with only the domain and problem instance descriptions. The model is expected to continue the sequence by generating the plan[3].

4.1 LLMs Learn to Plan in Natural Language, but Struggle in OOD Scenarios

Previous studies have asserted the effectiveness of fine-tuning LLMs for plan generation (Rossetti et al. 2024; Shah et al. 2024). We revisit this claim, examining whether the statements hold true in our extended PlanBench dataset.

In-Distribution Test: A Promising Start Table 1 presents the performance of our fine-tuned LLM on the vanilla corpus. The model indeed achieved high performance across all domains in in-distribution tests, mirroring the positive result reported by *PlanGPT* (Rossetti et al. 2024). Remarkably, we attain comparable performance using only 5.7% of their training data and a more complex input format expressed in natural language, suggesting not only better data efficiency but also better capability to process less structured data than previously anticipated.

Longer Planning: A Drastic Decline The 'long' test set reveals a significant performance drop, particularly in the NP-hard BLOCKSWORLD domain (Chenoweth 1991), where the *validity rate* falls from 98.5% to 13.5%. This dramatic decline underscores the model's limitations in handling longer and more complex planning scenarios, suggesting that the planning capabilities acquired through end-to-end fine-tuning are not robust.

CHILDSNACK: Partial Success? In the CHILDSNACK domain, which involves preparing sandwiches for allergic and not allergic children, the model achieves the highest validity rate at 66.0% in the long' test set. However, the reasoning complexity of the tasks in this domain is not very sensitive to plan length. While the plan length expands with an increase in the number of children to feed, the order in which the children are addressed does not matter. Upon examining the generated plans, it appears that the model's major shortcoming is **not** its ability to handle food allergies, which it manages adeptly. Instead, it is the model's failure to adapt to updated world dynamics involving more children and objects, often resulting in the omission of several children from the plan. Therefore, it suggests that the model's reasoning abilities are insufficient for adapting to changes in out-of-distribution scenarios, even when the core task logic is well comprehended.

Generalization to Novel Domains: A Clear Failure The fine-tuned model utterly failed to perform in the "unseen" and "obfuscated" test sets, unable to generate either valid or executable plans. This performance breakdown was particularly striking in the "obfuscated" test set. Here, the LLM planner often resorted to repeating irrelevant actions from domains present in the training set, neglecting the actual planning context (see Appendix C for more details).

Overall, our results refute the claim that fine-tuning alone enables LLMs to master complex planning. Next, we will examine whether the purported strategies can turn the tide.

4.2 The Secret Help of Permutation

For the sake of brevity, Table 2 presents only the average performance across all domains in the ablation study. Our analysis reveals an intriguing finding regarding the impact of permutation augmentation. While this technique does not significantly improve the *validity rate*, it largely enhances the *executability rate* (see Table 2 row 2). In particular, we observe a remarkable 75.5% score in "unseen" test set, while

[3]Code available at https://anonymous.4open.science/r/official-misconcept-lm-plan-gen-D34B

Table 2: Ablation Study on Strategy Effectiveness in Planning. Validity rates (valid.) and the Executability rate (exec.) are analyzed. Strategies such as Permutation, CoT, and Self-Correct show no significant *validity*. improvements but enhance *executability* in 'long' and other OOD scenarios. Notably, 'Goal CoT' appears to hinder performance. We attribute this to the dual duties of generating plans and accurately estimating the heuristics of the state, which increases overall complexity and hinders the model's ability to effectively learn both aspects. RL emerges as the only strategy that enhances *validity* in OOD scenarios. Improvements of statistical significance are highlighted in green, while significant declines are highlighted in red.

Label	Strategies					In-Distrib.		Long		Unseen		Obfuscated	
	Perm.	Goal CoT	State CoT	Self Correct	RL	valid.	exec.	valid.	exec.	valid.	exec.	valid.	exec.
1						99.3%	99.8%	34.8%	42.3%	**0%**	20.1%	**0%**	**0%**
2	✓					99.5%	99.8%	35.0%	48.3%	**0%**	75.5%	**0%**	**0%**
3	✓	✓				96.8%	98.5%	12.1%	18.7%	5.5%	53.4%	**0%**	**0%**
4	✓		✓			98.9%	99.5%	29.5%	43.0%	**0%**	**100%**	**0%**	**0%**
5	✓	✓	✓			98.7%	99.0%	23.8%	39.3%	**0%**	90.8%	**0%**	**0%**
6	✓			✓		99.7%	99.9%	32.6%	50.6%	**0%**	70.9%	**0%**	**0%**
7	✓	✓		✓		97.3%	98.6%	14.9%	25.6%	**0%**	38.7%	**0%**	**0%**
8	✓		✓	✓		98.1%	99.3%	27.5%	49.4%	**0%**	94.5%	**0%**	**0%**
9	✓	✓	✓	✓		98.3%	99.1%	25.9%	30.4%	**0%**	90.6%	**0%**	**0%**
10⋆					✓	99.2%	99.6%	41.5%	51.3%	12.5%	23.0%	**0%**	**0%**
11⋆				✓	✓	99.7%	**100%**	36.3%	53.6%	**0%**	71.5%	**0%**	**0%**

Table 3: Probing test: LLM effectively identifies when it makes mistakes.

Row	Precision	Recall
6	87.5	98.1
7	89.3	99.2
8	89.2	97.4
9⋆	90.5	99.2

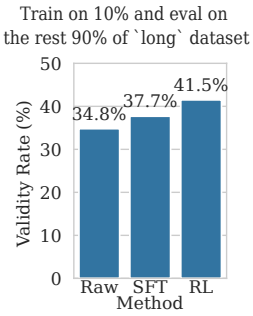

Figure 6: RL achieves higher gain than SFT in 'long' test.

the vanilla model only got 20.1% (row 1). This high performance suggests that permutation augmentation enables the model to effectively parse unseen problem content, which includes unseen actions, predicates and objects. This aligns with the findings of Allen-Zhu and Li (2023) and underscores the importance of data augmentation in enhancing LLMs capacity to interpret and adapt to new contexts.

4.3 Goal CoT: The Complexity Paradox and Overfitting Issue

Goal CoT is the only strategy that hinders planning performance among OOD cases, showing no improvement whatsoever (see Table 2 row 3, 5, 7, 9). We attribute the failure to two factors: **1. Complexity Paradox:** Estimating the goal distance adds complexity to the planning process. Although the intention was to provide heuristic guidance, this added layer paradoxically complicates the task. The requirement to predict the steps needed to achieve a goal demands precision but also restricts the model's flexibility during planning – By fixing the total number of action steps before planning begins, the model loses the ability to dynamically adjust its plan based on the evolving conditions. **2. Poor Generalization:** The model exhibits a noticeable bias towards estimating numbers within the range of plan lengths that it has previously encountered during the training stage. This aligns with the observable limitations of LLMs in generalizing to unseen formulas and number – an issue that has been extensively highlighted by Gorceix et al. (2024).

4.4 LLMs Recognize Mistakes But Fail to Correct Them

Despite high initial expectations for self-correction learning – stemming from its demonstrated effectiveness in solving maths, this recently proposed strategy did not improve *validity rates* (see Table 2 row 6, 7, 8, 9). To understand this outcome, we conducted *mistake identification probing tests* on test sets to assess the conditional probability of outputting the special 'removal' token when the model encounters a wrong step (see Appendix E for more details).

Results from Table 3 showed that the model is able to accurately identify errors, achieving particularly high precision (90.5%) and recall (99.2%) when all 4 strategies are combined (row 9). However, the detection capability does not lead to effective correction, indicating that future research should focus on how to leverage detected errors for correction. Nevertheless, the strategy gave a slight improvement in *executability rate* (e.g., row 10 vs. 11), suggesting its ability to enhance the *coherence* of generated sequences.

4.5 State CoT Boosts Executability with a Caveat: Efficacy Limited to Short Problems

We observed that *State CoT* does not improve plan *executability* within the 'long' test set, yet it significantly enhances performance within the 'unseen' test set (e.g., 100% in row 4). Importantly, the 'unseen' test set retains the same plan length distribution as the training set. Thus, we posit that the *State CoT*'s ability to enhance the model's understanding of state transition dynamics may likely be limited to the plan length distribution it encountered during training. Consequently, we do not observe an improvement in the 'long' test set. This also rationalizes why the *State CoT* demonstrates efficacy in other reasoning tasks (Wei et al. 2022; Yao et al. 2024), where these tasks typically require solution steps that align with the training data distribution. We shall verify this hypothesis in the next section through a 'plan continuation' experiment.

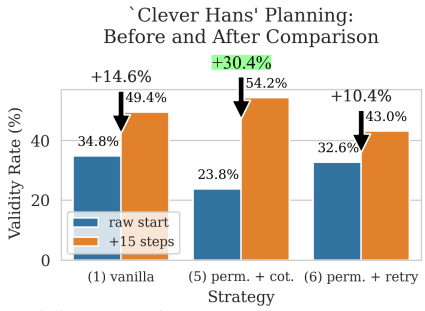

Figure 7: Validity rate of the LLM planner on the 'long' test set when provided with the first 15 actions. The model using CoT (Goal + State) showed the highest performance gain when hints were provided.

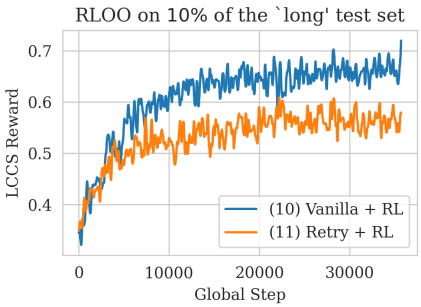

Figure 8: Reward curve of the RL training process. The LLM get further trained on the 10% of the 'long' test set with limited training steps. This however lead to a noticeable performance gain in OOD cases.

4.6 Familiar-Length Plan Continuation Experiments Reveal CoT's Potential

A critical question arises regarding the model's poor performance on longer problems within seen domains: Is this drop caused by distribution shift? Given that the model was trained on short-plan problem, it could have developed a bias towards shorter plans. To investigate this, we conducted a 'plan continuation' experiment, where we provided the first 15 true actions and asked the model to continue from there, ensuring that the remaining steps fell within the length distribution seen during training.

The results, as shown in Figure 7, reveal intriguing insights. Despite the significant hint provided, the model's performance still lags behind that of the in-distribution test set. This discrepancy is expected, as the model must infer the current world state from the given actions and then continue planning to reach the goal state. Even with the initial actions provided, predicting the world state remains challenging.

Interestingly, the model employing CoT (Goal + State) demonstrates the highest performance gain when provided with the hints. Its validity rate improves dramatically from the lowest (23.8%) to the highest (54.2%) among the compared strategies. This finding suggests that when CoT operates within its "comfort zone" (i.e., in-distribution scenarios), it begins to show its effectiveness in enhancing the model's planning, supporting the hypothesis presented in § 4.5. While this performance boost is encouraging, it also highlights a limitation: CoT appears to be overfit to in-distribution inference. This aligns with our earlier observation that the model faces difficulty estimating the goal distance that is not within the training distribution.

4.7 RL Enhances Model Performance

RL notably improves the performance under our end-to-end planning paradigm, especially on longer problems. Note that the model was trained on 10% of the 'long' test set with the proposed *LCCS*-based reward model, and evaluated on the 90% of the 'long' test set and other OOD test sets.

Despite the limited training data and suboptimal rewards achieved on this subset, RL boosted the validity rate on the 'long' test set from 34.8% to 41.5% (a 6.7% increase) and the executability rate from 42.3% to 53.6% (9.0%) (see Table 2, row 10). Interestingly, it also enabled the model to solve problems in the 'unseen' test set, achieving a 12.5% where it previously failed to generate any valid plans. The updated model does not exhibit overfitting, as indicated by the *LCCS* reward signal not reaching a perfect score of 1.0. Instead, the model has developed general planning strategies effective in unseen scenarios. To confirm that this improvement is not due to the additional training data, we also conducted supervised fine-tuning as described in Section 4.1 using the same training data. However, the outcomes were not as promising as those achieved with RL, as demonstrated in Figure 6. These results suggest that RL fosters more comprehensive planning skills compared to supervised fine-tuning (SFT), aligning with the findings of Liu et al. (2024).

Applying RL to the vanilla model led to faster convergence and improved results compared to its application to the model with self-correction skills, as illustrated in Figure 8 and row 10 and 11 in Table 2. We hypothesize that the self-correction strategy, by permitting repeated attempts at actions, effectively broadens the model's state space and thus poses a greater challenge to explore a valid solution.

5 Discussion

We investigate the enigma of why strategies aimed at improving LLM reasoning often fail to achieve expected results in planning tasks. Our findings reveal that, they do contribute to the overall plan quality, as reflected in the enhanced *executability rate*. This indicates progress towards more **coherent** plans, despite not directly leading to valid plans. We have limited our scope to the end-to-end plan generation paradigm, where the *search process* is not explicitly programmed. Within this context, we find that fine-tuning LLMs on datasets consisting solely of problems and corresponding reference plans struggles to foster robust planning skills beyond in-distribution instances. Nonetheless, our research reveals that RL stands out as the most effective strategy in this end-to-end paradigm, enhancing both the *validity* and *executability* rates on longer problems. This study provides a clear pathway for boosting the planning capability within the *next-token prediction* framework. Whereas previous work centered on pinpointing the shortcoming of LLMs

in plan generation, we show that the **direction** of improving LLM planning lies in increasing the likelihood of reaching goal state while preserving the high levels of executability that current strategies have already achieved.

References

Ahmadian, A.; Cremer, C.; Gallé, M.; Fadaee, M.; Kreutzer, J.; Üstün, A.; and Hooker, S. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*.

Allen-Zhu, Z.; and Li, Y. 2023. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*.

An, C.; Feng, J.; Lv, K.; Kong, L.; Qiu, X.; and Huang, X. 2022. Cont: Contrastive neural text generation. *Advances in Neural Information Processing Systems*, 35: 2197–2210.

Bachmann, G.; et al. 2024. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*.

Cao, D.; Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Automating Thought of Search: A Journey Towards Soundness and Completeness. *arXiv preprint arXiv:2408.11326*.

Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Sharukh Syed, M.; Tang, H.; Wu, Y.; Yan, Y.; Gil, T.; and Nir, L. 2020. Planimation. *arXiv preprint arXiv:2008.04600*.

Chen, G.; Liao, M.; Li, C.; and Fan, K. 2024. AlphaMath Almost Zero: Process Supervision without Process. In *Proceedings of the Neural Information Processing Systems (NeurIPS)*.

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34: 15084–15097.

Chenoweth, S. V. 1991. On the NP-Hardness of Blocks World. In *AAAI Conference on Artificial Intelligence*.

Cubuk, E. D.; Zoph, B.; Shlens, J.; and Le, Q. V. 2019. Randaugment: Practical data augmentation with no separate search. *arXiv preprint arXiv:1909.13719*, 2(4): 7.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ecoffet, A.; Huizinga, J.; Lehman, J.; Stanley, K. O.; and Clune, J. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Gontijo-Lopes, R.; Smullin, S. J.; Cubuk, E. D.; and Dyer, E. 2020. Affinity and diversity: Quantifying mechanisms of data augmentation. *arXiv preprint arXiv:2002.08973*.

Gorceix, A.; Le Chenadec, B.; Rammal, A.; Vadori, N.; and Veloso, M. 2024. Learning Mathematical Rules with Large Language Models. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; Muise, C.; Brachman, R.; Rossi, F.; and Stone, P. 2019. *An introduction to the planning domain definition language*, volume 13. Springer.

Howey, R.; et al. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI*, 294–301. IEEE Computer Society.

Huang, L.; Yu, W.; Ma, W.; Zhong, W.; Feng, Z.; Wang, H.; Chen, Q.; Peng, W.; Feng, X.; Qin, B.; and Liu, T. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ArXiv*, abs/2311.05232.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L. P.; and Murthy, A. B. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *Forty-first International Conference on Machine Learning*.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.

Kumar, A.; Zhuang, V.; Agarwal, R.; Su, Y.; Co-Reyes, J. D.; Singh, A.; Baumli, K.; Iqbal, S.; Bishop, C.; Roelofs, R.; et al. 2024. Training Language Models to Self-Correct via Reinforcement Learning. *arXiv preprint arXiv:2409.12917*.

Liu, A.; Feng, B.; Wang, B.; Wang, B.; Liu, B.; Zhao, C.; Dengr, C.; Ruan, C.; Dai, D.; Guo, D.; et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Liu, B.; Bubeck, S.; Eldan, R.; Kulkarni, J.; Li, Y.; Nguyen, A.; Ward, R.; and Zhang, Y. 2023. Tinygsm: achieving¿ 80% on gsm8k with small language models. *arXiv preprint arXiv:2312.09241*.

Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Mirzadeh, I.; Alizadeh, K.; Shahrokhi, H.; Tuzel, O.; Bengio, S.; and Farajtabar, M. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. *arXiv preprint arXiv:2410.05229*.

Prystawski, B.; Li, M.; and Goodman, N. D. 2023. Why think step by step? Reasoning emerges from the locality of experience. In *NeurIPS*.

Rossetti, N.; Tummolo, M.; Gerevini, A. E.; Putelli, L.; Serina, I.; Chiari, M.; and Olivato, M. 2024. Learning General Policies for Planning through GPT Models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 500–508.

Shah, K.; Dikkala, N.; Wang, X.; and Panigrahy, R. 2024. Causal Language Modeling Can Elicit Search and Reasoning Capabilities on Logic Puzzles. *arXiv preprint arXiv:2409.10502*.

Shi, F.; Suzgun, M.; Freitag, M.; Wang, X.; Srivats, S.; Vosoughi, S.; Chung, H. W.; Tay, Y.; Ruder, S.; Zhou, D.;

Das, D.; and Wei, J. 2023. Language models are multilingual chain-of-thought reasoners. In *ICLR*. OpenReview.net.

Stechly, K.; et al. 2024. Chain of thoughtlessness: An analysis of CoT in planning. *arXiv preprint arXiv:2405.04776*.

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2024a. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36: 75993–76005.

Valmeekam, K.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2022. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *Neural Information Processing Systems*.

Valmeekam, K.; Stechly, K.; Gundawar, A.; and Kambhampati, S. 2024b. Planning in Strawberry Fields: Evaluating and Improving the Planning and Scheduling Capabilities of LRM o1. *arXiv preprint arXiv:2410.02162*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023a. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR*. OpenReview.net.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR*. OpenReview.net.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Yang, A.; Yang, B.; Hui, B.; Zheng, B.; Yu, B.; Zhou, C.; Li, C.; Li, C.; Liu, D.; Huang, F.; et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Ye, T.; Xu, Z.; Li, Y.; and Allen-Zhu, Z. 2024. Physics of Language Models: Part 2.2, How to Learn From Mistakes on Grade-School Math Problems. *ArXiv*, abs/2408.16293.

The appendix consists of the following:

- Appendix A: Terminology Explanation: Executability and Validity of a Plan
- Appendix B: Implementation Details
- Appendix C: Planning Prompts and Responses
- Appendix D: Further Details on A Failure Case of Self-Correction Learning
- Appendix E: Further Details on Mistake Identification Probing Tests
- Appendix F: Additional Results: *pass@k* Validity Rate
- Appendix G: Additional Results: Goal Satisfiability Rate

Our codebase is available at https://anonymous.4open.science/r/official-misconcept-lm-plan-gen-D34B and the dataset will be released via Huggingface Hub once the review process is completed.

A Terminology Explanation: Executability and Validity of a Plan

The definition below can be found in Howey et al. (2004) and is also presented as DEFINITION 2.8 in the textbook (Haslum et al. 2019), defined as follows:

Definition A.1 (Executability of a Plan)**.** A plan is executable if it defines an action sequence $(a_0, a_1, \ldots, a_{n-1})$ with states $(s_0, s_1, \ldots, s_n)$. s_0 is the initial state and for each $i = 0, 1, \ldots, n-1$, s_{i+1} is the result of executing a_i in s_i, and the precondition of a_i must hold in s_i and s_{i+1} is the result of removing delete effects, adding add effects and applying numeric effects. The state s_n is called the final state produced by the plan and the state sequence $(s_0, s_1, \ldots, s_n)$ is called the trace of the plan. An executable plan produce a unique trace.

Definition A.2 (Validity of a Plan)**.** A plan is valid if and only if it is executable and the goal of the problem G holds in the final state s_n produced by the plan.

These are the formal definitions written in the PDDL textbook, and we can see that executability is a prerequisite for a plan to be valid. This underscores the importance of including the executability metric in our evaluation of plan quality.

B Implementation Details

We used 4 Nvidia A100 GPUs to fine-tune the QWEN2-7B-INSTRUCT model on the extended PlanBench dataset. Note that during the training process, especially during reinforcement learning, the machine occasionally encounter out of memory (OOM) errors, even when reducing the batch size to 1. This issue primarily arose because when applying Chain-of-Thought (CoT) prompts to the model, the length of the response sequences often exceeded 20,000 tokens, which is beyond what most open-source frameworks can handle.

Below are the detailed hyperparameter used in our experiments.

B.1 Hyperparameter

Topic	Hyperparameter	Value	Notes
Data generation	mistake rate	0.2	Follow the best practice in paper
	seed	1111	
	train size	4000	
	test size	200	
Fine-tuning	model name	Qwen2-7B-Instruct	
	stage	sft	
	deepspeed	zero 3	
	per device train batch size	1 - 4	Adaptive to the response length
	gradient accumulation steps	1 - 2	
	num train epochs	2.0	
	bf16	true	
	lr scheduler type	cosine	
	learning rate	1.0e-5	
RL	lora rank	128	
	lora alpha	256	
	per device train batch size	1	
	gradient accumulation steps	2-4	
	if qlora	false	
	seed	23	
	num ppo epochs	1	
	num mini batches	1	
	learning rate	3.0e-6	
	total episodes	500000	
	missing eos penalty	0.5	
	lora dropout	0.05	
Evaluation	k	1, 3, 5	Number of plan candidates to generate
	top p	0.93	Top P Sampling coefficient when $k > 1$, if P is set to 0.75, the model will include enough top tokens so that their combined probability equals or exceeds 75%.
	top k	50	If it is set to 50, only the 50 most likely tokens are considered for the next output token

C Problem Instances Prompts and Responses

C.1 Detailed Examples of Training Corpus from DRIVERLOG Domain

We will release the extended PlanBench dataset via Huggingface Hub after anonymous review. In this section, we provide several examples of problem instance context and responses text from the DRIVERLOG domain.

To illustrate modifications to the response section of the training corpus, we use row 9 from Table 2 as an example, where all four strategies are combined.

Figure 9 will display the problem context section of the example. Following that, Figure 10 presents a detailed response section that incorporates the *Goal CoT*, *State CoT*, and *Self-Correction* strategies.

Training Data Driverlog Query Content Example

```
Query:
[CONTEXT]
I have to plan how to transport objects between locations using trucks driven by drivers. The drivers can walk between
locations, board and disembark from trucks, and drive trucks between locations. The trucks can be loaded and unloaded with
objects.
[ACTION DESCRIPTION]
Here are the actions that can be performed:
Load an object into a truck.
Unload an object from a truck.
A driver boards a truck.
A driver disembarks from a truck.
A driver drives a truck from one location to another location.
A driver walks from one location to another location.

The following are the restrictions on the actions:
An object can be loaded into a truck only if the object is at the same location as the truck.
Once an object is loaded into a truck, the object is not at the location but is in the truck.
An object can be unloaded from a truck only if the object is in the truck.
Once an object is unloaded from a truck, the object is not in the truck but is at the location of the truck.
A driver can board a truck only if the driver is at the same location as the truck.
A driver can board a truck only if the truck is empty.
Once a driver boards a truck, we say the driver is driving the truck.
Once a driver boards a truck, the driver is no longer at the location.
Once a driver boards a truck, the truck is no longer empty.
A driver can disembark from a truck only if the driver is driving the truck.
Once a driver disembarks from a truck, we say the driver is no longer driving the truck.
Once a driver disembarks from a truck, the driver is at the location of the truck.
Once a driver disembarks from a truck, the truck is empty.
A driver can drive a truck from one location to another location only if the driver is driving the truck and the truck is at
the from-location.
A driver can drive a truck from one location to another location only if the from-location and to-location are linked.
Once a driver drives a truck from one location to another location, the truck is at the to-location and is no longer at the
from-location.
A driver can walk from one location to another location only if the driver is at the from-location and the from-location and
to-location are linked.
Once a driver walks from one location to another location, the driver is at the to-location and is no longer at the
from-location.

[STATEMENT]
As initial conditions I have that, place 1 and place 0 are linked, driver 2 is at place 1, path 1-0 and place 0 are connected,
path 1-0 and place 1 are connected, truck 1 is empty, place 1 and path 1-0 are connected, driver 1 is at place 1, package 1 is
at place 1, truck 1 is at place 1, place 0 and place 1 are linked, place 0 and path 1-0 are connected.

My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0.
```

Figure 9: The LLM planner is trained on these lengthy natural language queries that describe the context, action description, initial and goal states in the Driverlog domain. It's important to recognize that the next-token prediction training of LLMs during fine-tuning isn't limited to predicting just the response tokens. In fact, the model is trained to predict all tokens, including those in the query context, similar to the pretraining process. This aspect is often overlooked in the planning community, where people may assume that the model is only trained to generate the plan response.

Training Data Response Section in Driverlog Domain When Applied CoT and Self-Correction

```
Response:
My plan is as follows:
[PLAN]
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 6 </COUNT> <PRE
CON> A driver 1 can board a truck 1 only if the driver 1 is at the same place 1 as the truck 1. A driver 1 can board a truck 1
only if the truck 1 is empty. </PRECON> driver 1 boards truck 1 at place 1 <EFFECT> Once a driver 1 boards a truck 1, the truck
1 is no longer empty. Once a driver 1 boards a truck 1, we say the driver 1 is driving the truck 1. Once a driver 1 boards a
truck 1, the driver 1 is no longer at the place 1. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 5 </COUNT> <PRE
CON> An package 1 can be unloaded from a truck 1 only if the package 1 is in the truck 1. </PRECON> unload package 1 from truck
1 at place 0 <EFFECT> Once an package 1 is unloaded from a truck 1, the package 1 is not in the truck 1 but is at the place 0
of the truck 1. </EFFECT> WRONG
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 5 </COUNT>
<PRECON> An package 1 can be loaded into a truck 1 only if the package 1 is at the same place 1 as the truck 1. </PRECON> load
package 1 into truck 1 at place 1 <EFFECT> Once an package 1 is loaded into a truck 1, the package 1 is not at the place 1 but
is in the truck 1. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 4 </COUNT>
<PRECON> A driver 2 can walk from place 1 to path 1-0 only if the driver 2 is at the place 1 and the place 1 and path 1-0 are
linked. </PRECON> driver 2 walks from place 1 to path 1-0 <EFFECT> Once a driver 2 walks from place 1 to path 1-0, the driver 2
is at the path 1-0 and is no longer at the place 1. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 1 </COUNT>
<PRECON> A driver 2 can walk from path 1-0 to place 0 only if the driver 2 is at the path 1-0 and the path 1-0 and place 0 are
linked. </PRECON> driver 2 walks from path 1-0 to place 0 <EFFECT> Once a driver 2 walks from path 1-0 to place 0, the driver 2
is at the place 0 and is no longer at the path 1-0. </EFFECT> WRONG
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 3 </COUNT>
<PRECON> A driver 2 can walk from path 1-0 to place 0 only if the driver 2 is at the path 1-0 and the path 1-0 and place 0 are
linked. </PRECON> driver 2 walks from path 1-0 to place 0 <EFFECT> Once a driver 2 walks from path 1-0 to place 0, the driver 2
is at the place 0 and is no longer at the path 1-0. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 2 </COUNT>
<PRECON> A driver 1 can drive a truck 1 from place 1 to place 0 only if the driver 1 is driving the truck 1 and the truck 1 is
at the place 1. A driver 1 can drive a truck 1 from place 1 to place 0 only if the place 1 and place 0 are linked. </PRECON>
driver 1 drives truck 1 from place 1 to place 0 <EFFECT> Once a driver 1 drives a truck 1 from place 1 to place 0, the truck 1
is at the place 0 and is no longer at the place 1. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 1 </COUNT> <PRE
CON> An package 1 can be unloaded from a truck 1 only if the package 1 is in the truck 1. </PRECON> unload package 1 from truck
1 at place 0 <EFFECT> Once an package 1 is unloaded from a truck 1, the package 1 is not in the truck 1 but is at the place 0
of the truck 1. </EFFECT>
<GOAL> My goal is to have that truck 1 is at place 1, driver 2 is at place 0, package 1 is at place 0. <COUNT> 0 </COUNT>
<PRECON> A driver 1 can drive a truck 1 from place 0 to place 1 only if the driver 1 is driving the truck 1 and the truck 1 is
at the place 0. A driver 1 can drive a truck 1 from place 0 to place 1 only if the place 0 and place 1 are linked. </PRECON>
driver 1 drives truck 1 from place 0 to place 1 <EFFECT> Once a driver 1 drives a truck 1 from place 0 to place 1, the truck 1
is at the place 1 and is no longer at the place 0. </EFFECT>
[PLAN END]
```

Figure 10: The *Goal CoT* contains two components, one the the <GOAL> tag that prompts the agent to repeat the goal, and the other is the <COUNT> tag that prompts the agent to count the remaining steps to the goal. The *State CoT* contains two components, one is the <PRECON> tag that prompts the agent to provide grounded conditions for the action, and the other is the <EFFECT> tag that prompts the agent to provide grounded effects. Both the preconditions and effects are specifically tied to the actual objects present in the problem instance. In this example, there are two incorrect steps in the plan, which are marked as *WRONG*. The first incorrect step borrowed the second to last step from the reference plan. The second step provides an incorrect value for the remaining steps to the goal.

Additinoal Detail for the Error Correction Data Synthesization In our current setting, the erroneous action step is generated by randomly selecting an action from later in the reference plan sequence and inserting it to the current position, followed by the `[WRONG]` token, indicating that the current step is incorrect and should be retried. This method introduces an inductive bias that encourages the model not to skip steps. That is, we want the model to not taking shortcuts to the states that are closer to the goal state, but to strictly obey the state transition dynamics. In our experiments, we also explored an alternative approach that just select irrelevant actions from the action space. This alternative approach will generate the erroneous step by modifying the current correct action step based on two strategies: (1) either keeping the action constant while changing to other valid

objects or (2) keeping the object constant while changing to other valid actions. However, this alternative method produced worse results than our current approach. It suggests that the inductive bias introduced by our current method is beneficial for the model to learn the state transition dynamics.

C.2 Obfuscated Prompts: BLOCKSWORLD

Obfuscated domain replaces the name of the predicates, actions and objects with unfamiliar vocabulary. The following Figure 11 is a comparison of the original and obfuscated domain descriptions for the BLOCKSWORLD domain:

Original Prompts vs. Obfuscated Prompts in Blockworld Domain

Original:

```
Query:
I am playing with a set of blocks where I need to arrange the
blocks into stacks.
[ACTION DESCRIPTION]
Here are the actions that can be performed:
(a) Pick up a block
(b) Unstack a block from on top of another block
(c) Put down a block
(d) Stack a block on top of another block

The following are the restrictions on the actions:
(1) I can only pick up or unstack one block at a time.
(2) I can only pick up or unstack a block if my hand is empty.
(3) I can only pick up a block if the block is on the table and
the block is clear. A block is clear if the block has no other
blocks on top of it and if the block is not picked up.

(4) I can only unstack a block from on top of another block if
the block I am unstacking was really on top of the other block.
(5) I can only unstack a block from on top of another block if
the block I am unstacking is clear.
(6) Once I pick up or unstack a block, I am holding the block.

(7) I can only put down a block that I am holding.

(8) I can only stack a block on top of another block if I am
holding the block being stacked.
(9) I can only stack a block on top of another block if the
block onto which I am stacking is clear.
(10) Once I put down or stack a block, my hand becomes empty.
(11) Once you stack a block on top of a second block, the
second block is no longer clear.
[STATEMENT]
As initial conditions I have that
(i) the yellow block is on top of the red block,
(ii) the hand is empty,
(iii) the red block is on the table, the orange block is on the
table, the yellow block is clear, the orange block is clear.

My goal is to have that
(i) the red block is on top of the yellow block.
```

Obfuscated:

```
Query:
I am playing with a set of objects.

[ACTION DESCRIPTION]
Here are the actions that can be performed:
(a) Attack object
(b) Feast object from another object
(c) Succumb object
(d) Overcome object from another object

The following are the restrictions on the actions:
(1) I can only attack or feast one object at a time.
(2) I can only attack or feast an object if there is harmony.
(3) I can only attack an object if the object is a planet and
the object is a province. An object is a province if the object
does not crave any other object and if the object is not being
attacked.
(4) I can only feast an object from on top of another object if
the object I am feasting craves the other object.
(5) I can only feast an object from on top of another object if
the object I am feasting is a province.
(6) Once I attack or feast an object, there is pain with the
object.
(7) I can only succumb an object if there is pain with the
object.
(8) I can only overcome an object from another object if there
is pain with the object being overcome.
(9) I can only overcome an object from another object if the
object onto which I am overcoming the object is a province.
(10) Once I succumb or overcome an object, there is harmony.
(11) Once you overcome an object from a second object, the
second object is no longer a province.
[STATEMENT]
As initial conditions I have that
(i) object b craves object a,
(ii) harmony,
(iii) planet object a, planet object c, province object b,
province object c.

My goal is to have that
(i) object a craves object b.
```

Figure 11: Comparison of original and obfuscated prompts in the blockworld domain. Obfuscated vocabularies for actions are: *pick-up* → *attack*, *unstack* → *feast*, *put-down* → *succumb*, *stack* → *overcome*; Obfuscated vocabularies for objects are: *block* → *object*; Obfuscated vocabularies for predicates are: *on table* → *planet*, *clear* → *province*, *hand empty* → *harmony*, *holding* → *pain*, *on top of* → *craves*.

The LLM planner had no exposure to the obfuscated domain during training. Our results showed that achieved 0% validity and executability rates on the obfuscated test set. Upon examining the generated plans, we discovered an intriguing pattern: the

model tends to ignore the obfuscated domain context and instead produced actions from the original Blocksworld domain. This behavior persisted even when we provided the first action step as a hint, as illustrated in Figure 12.

Response Example for Obfuscated Blockworld Problem

```
Response:
My plan is as follows:
[PLAN]
# Provide the first action step as a hint
feast object a from object b
# The LLM planner starts from here
put down object 1 # object 1 comes from nowhere
unstack the blue block from the orange block
put down the blue block
unstack the orange block from the red block
put down the orange block
unstack the red block from the blue block
put down the red block
unstack the blue block from the yellow block
put down the blue block
unstack the yellow block from the red block
put down the yellow block
unstack the orange block from the blue block
put down the orange block
pick up the blue block
stack the blue block on top of the yellow block
pick up the orange block
stack the orange block on top of the blue block
unstack the white block from the orange block
put down the white block
unstack the orange block from on top of the blue block
put down the orange block
unstack the blue block from on top of the yellow block
put down the blue block
pick up the orange block
stack the orange block on top of the blue block
pick up the white block
stack the white block on top of the orange block
[PLAN END]
```

Figure 12: We observed an interesting pattern in the responses generated by the LLM planner for obfuscated blockworld problems. The planner tends to neglect the obfuscated domain context and continue to generate the actions based on the original domain context. This behavior remains even when we provide the first action step as a hint.

An optimistic interpretation of this behavior is that the model is capable of linking the obfuscated domain back to the original domain. But eventually the model treat the obfuscated domain as an outlier and revert back to the original domain context, demonstrating that LLMs are highly sensitive to the specific vocabulary used in their training data, and struggle to generalize its planning capabilities to scenarios that deviate from its training distribution.

Two noteworthy observations emerge from the generated plan which we often call it as *hallucination*:

1. The model creates actions like "put down object 1," where "object 1" doesn't exist in either the obfuscated or original problem descriptions.
2. Interestingly, the actions generated for the original domain context often form an executable plan sequence. However, there's no one-to-one correspondence between the objects in the original and obfuscated versions.

D Further Details on A Failure Case of Self-Correction Learning

We conducted qualitative analysis to understand the effect of self-correction learning on the model's planning capabilities. In the main text, we have stated that the model is able to identify errors in a high precision and recall rate, but fails to correct them effectively. We further verify this claim by examining the generated plan sequence when the model try to solve BLOCKSWORLD problem in the 'long' test set. The model we used is row 6 in Table 2, which is trained with the self-correction learning strategy and permutation augmentation.

Blockworld Problem Instance Query Context

```
Query:
I am playing with a set of blocks where I need to arrange
the blocks into stacks.
[ACTION DESCRIPTION]
Here are the actions that can be performed:
(a) Pick up a block
(b) Unstack a block from on top of another block
(c) Put down a block
(d) Stack a block on top of another block
The following are the restrictions on the actions:
...
[STATEMENT]
As initial conditions I have that
the green block is on top of the red block, the yellow block
is on top of the cyan block, the blue block is clear, the
orange block is on top of the magenta block, the white block
is on the table, the cyan block is on top of the orange
block, the blue block is on top of the black block, the
black block is on top of the green block, the magenta block
is on top of the white block, the red block is on top of the
yellow block, the hand is empty.

My goal is to have that
the cyan block is on top of the red block, the blue block is
on top of the cyan block, the green block is on top of the
magenta block, the orange block is on top of the blue block,
the red block is on top of the yellow block, the white block
is on top of the green block, the yellow block is on top of
the white block.
```

Response Generated By Trained LLM Planner

```
Response:
My plan is as follows: [PLAN]
unstack the blue block from on top of the black block
put down the blue block
unstack the blue block from on top of the yellow block WRONG
unstack the blue block from on top of the yellow block WRONG
unstack the blue block from on top of the blue block WRONG
unstack the yellow block from on top of the blue block WRONG
unstack the blue block from on top of the yellow block
put down the blue block
unstack the yellow block from on top of the blue block WRONG
unstack the yellow block from on top of the blue block WRONG
unstack the yellow block from on top of the blue block WRONG
unstack the blue block from on top of the blue block WRONG
put down the blue block
unstack the yellow block from on top of the blue block WRONG
unstack the yellow block from on top of the blue block WRONG
unstack the yellow block
put down the yellow block
pick up the blue block
stack the blue block on top of the yellow block
unstack the orange block from on top of the blue block WRONG
unstack the blue block from on top of the yellow block
put down the blue block
stack the orange block on top of the blue block
unstack the orange block from on top of the blue block WRONG
unstack the orange block from on top of the blue block WRONG
...
put down the orange block
unstack the blue block from on top of the yellow block
put down the blue block
pick up the orange block
stack the orange block on top of the blue block
[PLAN END]
```

Blockworld Problem Instance Reference Plan

```
(unstack blue black) (put-down blue) (unstack black green)
(put-down black) (unstack green red) (put-down green)
(unstack red yellow) (put-down red) (unstack yellow cyan)
(put-down yellow) (unstack cyan orange) (stack cyan red)
(unstack orange magenta) (stack orange blue) (unstack
magenta white) (put-down magenta) (pick-up green) (stack
green magenta) (pick-up white) (stack white green) (pick-up
yellow) (stack yellow white) (unstack cyan red) (put-down
cyan) (pick-up red) (stack red yellow) (pick-up cyan) (stack
cyan red) (unstack orange blue) (put-down orange) (pick-up
blue) (stack blue cyan) (pick-up orange) (stack orange blue)
```

To better understand the problem, we visualize both the initial state and the goal state using the Planimation Tool (Chen et al. 2020). successfully recognizes the first two steps: unstacking the blue block from the black block and then placing it down. However, it fails to identify the next step. Upon examining the visualization, it becomes evident that the next action should be to unstack the black block from the green block. Instead, the model does not acknowledge the presence of the green block in the scene. This oversight is reflected in the generated plan, where interactions with the green block are notably absent, indicating that the model struggles to accurately parse the initial state.

As a result, the LLM attempts to continue manipulating the blue block but quickly realizes this approach is incorrect, leading it to generate a `[WRONG]` token. This behavior aligns with the high precision and recall rates observed in probing tests.

Nevertheless, when retrying, the model remains focused on the blue block, suggesting it is unable to effectively correct its mistake. A possible remedy for this issue is to introduce a more sophisticated mechanism that allows external expert to provide detailed feedback on why the model's action is incorrect, enabling the model to learn from its mistakes more effectively.

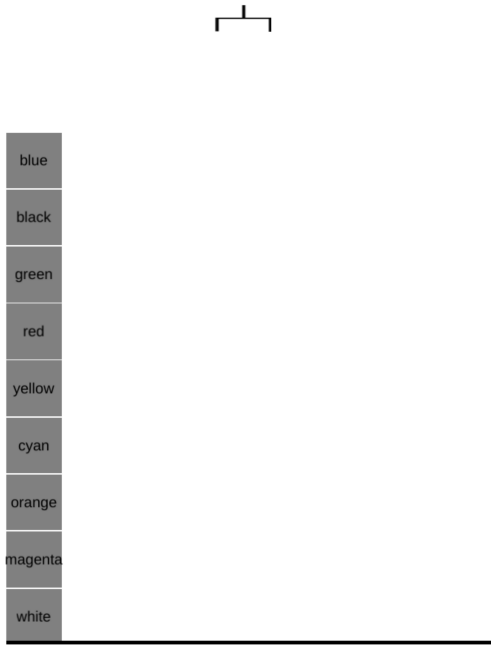

Figure 13: Initial state of the BLOCKSWORLD problem

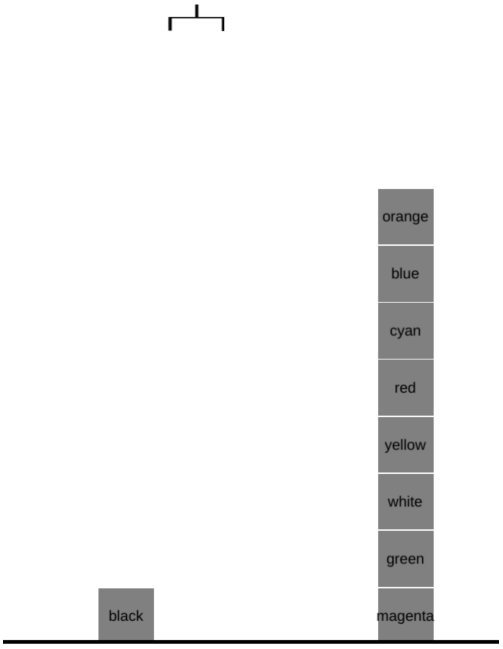

Figure 14: Goal state of the BLOCKSWORLD problem

E Further Details on Mistake Identification Probing Tests

Probing tests utilized the feature of the autoregressive generation process inherent in LLMs, wherein each token is generated by conditioning on all preceding tokens, producing *logits* scores across the vocabulary. For a greedy decoding, the model selects the token with the highest *logits* score. In our case, in order to train the model to have the self-correction skills, we introduced an additional special token – `[WRONG]` – to the vocabulary. This token serves as an indicator for the model to recognize that the current action step is incorrect and thus re-attempt a new action. Therefore, when the model reaches the end of every generated action step, it faces a binary decision for its subsequent token: if it deems the current step correct, it generates a new line token ($\backslash n$); otherwise, it generates the `[WRONG]` token. Both options are single-token choices, ensuring a fair comparison of their conditional probabilities.

For our probing tests, we selected the 'long' test set as our evaluation dataset. We generated synthesized plan sequences using the same methodology employed in our self-correction learning strategy – that is by randomly selecting an action that appears later in the reference plan sequence and inserting it to the current step followed by a *'special removal token'*. To create instances for probing test, we processed the plans in two ways: for correct examples, we truncated the plan at the end of a correct step sentence, while for incorrect examples, we truncated at the end of an erroneous step sentence. After freezing the fine-tuned LLM, we conducted a comprehensive evaluation by measuring four distinct conditional probabilities:

1. Generating the `[WRONG]` token when encountering a wrong step.
2. Generating the new line token ($\backslash n$) when encountering a wrong step.
3. Generating the `[WRONG]` token when encountering a correct step.
4. Generating the new line token ($\backslash n$) when encountering a correct step.

This 2×2 matrix of probabilities allows us to assess the model's ability to discriminate between correct and incorrect steps accurately. Therefore, the precision and recall value in Table 3 is calculated based as follows:

We considered a true positive (TP) when the probability of generating the `[WRONG]` token was higher than that of generating the new line token for a wrong step, and the other way around for a correct step. The total number of wrong steps in the test set represented all actual mistakes (TN + FP). We counted an identified mistake when the probability of generating the `[WRONG]` token exceeded that of the new line token, regardless of whether the step was actually wrong or correct (TP + FP). Finally, precision was then calculated as the ratio of true positives to all identified mistakes, while recall was computed as the ratio of true positives to all actual mistakes in the test set.

F Additional Results: *pass@k* Validity Rate

The *pass@k* metric is an important evaluation measure for assessing the performance of LLMs in reasoning. It provides insights into the model's ability to generate correct solutions across multiple attempts. The probabilistic nature of the generative process in LLMs results in the fact that the correct plan may not always be the most confident one. Due to this, employing a majority vote over multiple sampling outputs has become a common practice to enhance the robustness of model predictions (Wang et al. 2023b). In our context, *pass@k* metrics measure the validity rate by taking the best of k samples generated by the model. The sampling hyperparameter are shown in Appendix B.1. Results are shown in Table 4.

Table 4: Pass@k validity rates for different strategies across test sets. Results show consistent improvements for 'long' test sets as k increases, while 'unseen' and 'obfuscated' sets show no significant gains. Notably, the vanilla (1) and RL (10) strategies demonstrate the highest performance gains with multiple sampling

Row	Strategies					Long			Unseen			Obfuscated		
	Perm.	Goal CoT	State CoT	Self Correct	RL	pass@1	pass@3	pass@5	pass@1	pass@3	pass@5	pass@1	pass@3	pass@5
1⋆						34.8%	43.9%	49.0%	**0%**	12.2%	12.2%	**0%**	**0%**	**0%**
2	✓					35.0%	40.4%	42.9%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
3	✓	✓				12.1%	15.6%	17.8%	5.5%	6.0%	7.0%	**0%**	**0%**	**0%**
4	✓		✓			29.5%	35.3%	37.2%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
5	✓	✓	✓			23.8%	28.4%	34.1%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
6	✓			✓		32.6%	40.4%	42.3%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
7	✓	✓		✓		14.9%	21.0%	24.3%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
8	✓		✓	✓		27.5%	31.4%	35.1%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
9	✓	✓	✓	✓		25.9%	31.2%	36.5%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**
10⋆					✓	41.5%	49.2%	52.0%	12.5%	12.5%	12.5%	**0%**	**0%**	**0%**
11				✓	✓	36.3%	42.3%	45.0%	**0%**	**0%**	**0%**	**0%**	**0%**	**0%**

The results indicate that multiple sampling does not enhance the model's performance on the 'unseen' and 'obfuscated' domains, highlighting the model's limited generalization to unfamiliar contexts. In contrast, we observed a consistent improvement in the *pass@k* validity rate across all strategies for the 'long' test set. Among all the ablation strategies, the vanilla (row 1) and RL (row 10) models demonstrate the most significant improvements when utilizing multiple sampling. So, we actually observe this trend – strategies that increases the response length, such as incorporating Chain of Thought prompts (row 3 to 9), actually cannot benefit from multiple sampling. We attribute this phenomenon to the nature of autoregressive prediction in LLMs. That is, as the response length increases, the context provided by all preceding tokens becomes denser and more concrete. By conditioning on this context, the model will have more focused and less diverse set of plausible continuations for the next token. Consequently, the benefits of generating multiple samples are diminished for most strategies.

G Additional Results: Goal Satisfiability Rate

It is important to note that goal satisfiability is also not a standard term in the planning literature. Nevertheless, an informal definition of goal satisfiability, derived from the definition of executability, is as follows:

Definition G.1 (Goal-Satisfiability of a Plan)**.** A plan is goal-satisfiable if it defines an action sequence $(a_0, a_1, \ldots, a_{n-1})$ with states $(s_0, s_1, \ldots, s_n)$. s_0 is the initial state and for each $i = 0, 1, \ldots, n-1$, s_{i+1} is the result of executing a_i in s_i, removing delete effects, adding add effects and applying numeric effects. The state s_n is called the final goal state and the goal G holds in s_n. A goal-satisfiable plan may not be executable.

We also measured the *goal satisfiability rate* in the interest of completeness. However, there's a fundamental misalignment between this metric and the nature of autoregressive language models in end-to-end plan generation. Here's why:

- The sequential nature of autoregressive language models allows them to generate plans one token at a time, moving from left to right, similar to the forward progression of a plan sequence.
- Each new prediction by nature aims to maintain consistency with preceding ones.
- This inherently prioritizes local state transition coherence over goal satisfaction, just like how forward progression in a plan sequence will not jump to the goal state without ensuring the coherence of the preceding actions.
- Existing strategies, particularly *State CoT*, often emphasize the consistency of the local step transitions, therefore, pursuing goal satisfiability before ensuring executability conflicts with the idea of producing a plan sequence in a left-to-right manner.

Therefore, we say that the goal satisfiability metric fail to account for the characteristics of end-to-end plan generation in autoregressive language models. Nevertheless, this can be also seen as a limitation of the end-to-end left-to-right plan generation paradigm, as the trained model lack mechanisms for looking ahead to future states and conducting backward searches. It will eventually undermines the model's ability to generate a valid plan.

The results of the goal satisfiability rate are shown in Table 5.

Table 5: The results show a slight decrease in the goal satisfiability rate across the different strategies. This indicates that the strategies do focus on the sequential consistency due to the nature of autoregressive models and prioritize less on the goal satisfiability.

Row	**Strategies**					**In-Distrib.**	**Long**	**Unseen**	**Obfuscated**
	Perm.	Goal CoT	State CoT	Self Correct	RL	goal sat.	goal sat.	goal sat.	goal sat.
1⋆						**100%**	64.5%	63.5%	**0%**
2	✓					**100%**	62.0%	25.5%	**0%**
3	✓	✓				**100%**	35.0%	27.0%	**0%**
4	✓		✓			**100%**	57.0%	**0%**	**0%**
5	✓	✓	✓			97.5%	55.5%	8.0%	**0%**
6	✓			✓		**100%**	57.5%	28.5%	**0%**
7	✓	✓		✓		96.0%	46.5%	30.5%	**0%**
8	✓		✓	✓		**100%**	56.5%	**0%**	**0%**
9	✓	✓	✓	✓		98.5%	53.5%	**0%**	**0%**
10⋆					✓	98.5%	73.5%	53.5%	**0%**
11				✓	✓	**100%**	60.0%	29.0%	**0%**

We observed a decrease in the goal satisfiability rate across various strategies. This result is reasonable, as improvements in both the executability and goal satisfiability rates would naturally lead to an increase in the final validity rate, which is not the case in our experiments. As shown in Table 5, the vanilla model (row 1) achieved a high goal satisfiability rate across the test sets, though this did not correspond to a high validity rate, as indicated by our experiments discussed in the main text. The evaluated strategies maintained a considerable invariance in the goal satisfiability rate for the 'long' test set but experience a decrease in the 'unseen' test set. Specifically, the *State CoT* strategy shows the most significant drop, primarily due to its focus on maintaining the state transition consistency of the generated plans.