Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

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

Sukai Huang, Trevor Cohn* and Nir Lipovetzky

School of Computing and Information Systems, The University of Melbourne, Australia
sukaih@student.unimelb.edu.au, {trevor.cohn, nir.lipovetzky}@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 paper revisits these claims by developing an end-to-end LLM-based planner and evaluating a range of reasoning-enhancement strategies — including fine-tuning, Chain-of-Thought (CoT) prompting, and reinforcement learning (RL) — across multiple dimensions of plan quality: *validity*, *executability*, *goal satisfiability*, and more. Our findings reveal fine-tuning alone is insufficient, especially on out-of-distribution tasks. Strategies like CoT prompting primarily enhance local coherence, yielding higher executability rates — a necessary prerequisite for validity — but provide only incremental gains and struggle to ensure global plan validity. Notably, RL guided by a novel *Longest Contiguous Common Subsequence* reward significantly enhances both executability and validity, particularly on longer-horizon problems. Overall, our research addresses key misconceptions in the LLM-planning literature and underscores reward-driven RL optimization as a promising direction for advancing robust LLM-based planning by jointly improving executability and validity.

Code — https://github.com/Sino-Huang/official-misconcept-lm-plan-gen

1 Introduction

While latest 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. 2024). 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 (Pallagani et al. 2023; 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

2. Insufficient Analysis of Failure Modes: Pessimistic studies (e.g., Valmeekam et al. (2024)) typically focus solely on plan *validity*, an extremely stringent criterion that requires the entire plan to be perfectly correct. 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. 2023), *self-correction* (Madaan et al. 2024; Ye et al. 2024; Kumar et al. 2024), and *reinforcement learning* (RL) (DeepSeek-AI et al. 2025). Our evaluation extends beyond the plan *validity* metric to include plan *executability*, which, refers to whether the preconditions of all actions are satisfied by the state at execution. and employs a comprehensive suite of diagnostic experiments to identify where each strategy succeeds or fails. Moreover, OOD test sets are deliberately designed to evaluate the model's generalization capabilities in planning.

This work aims to clarify 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

*Now at Google DeepMind

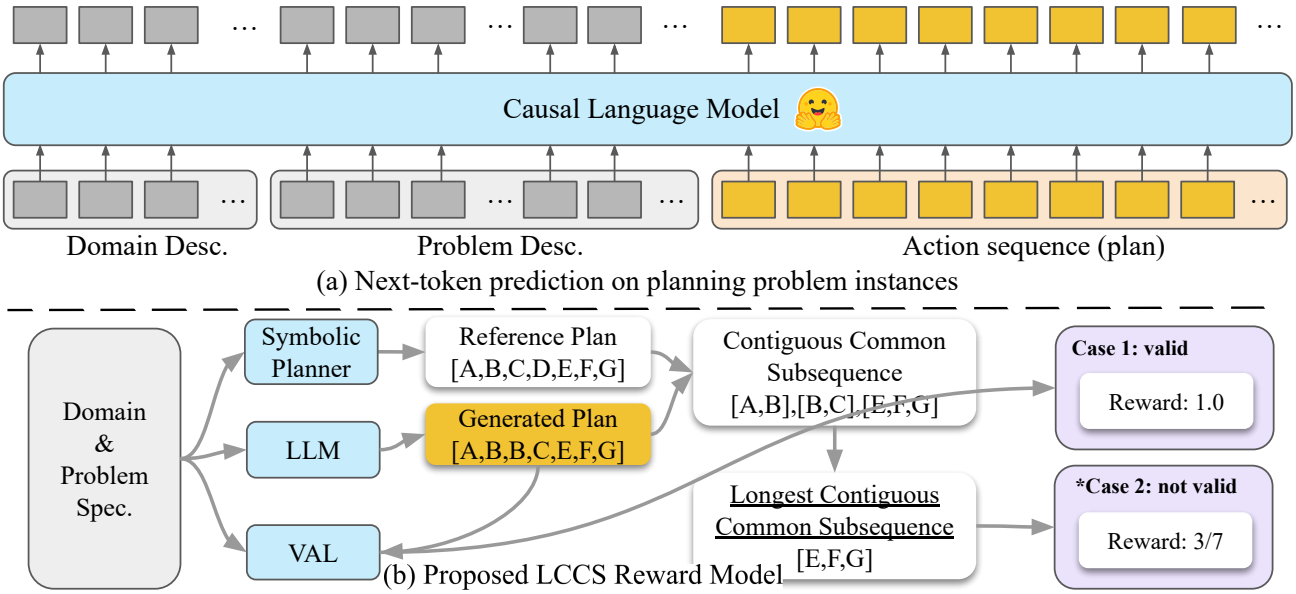

Figure 1: *(a) Next-token prediction:* The LLM is trained to predict the next token in corpus containing domain and problem details and their plans, proceeding left-to-right. (b) *Proposed LCCS Reward:* We use the length of LCCS as an auxiliary reward signal for RL. It provides granular feedback to fill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

longer planning problems.

2 Background & Related Work

Classical Planning. PDDL is the standard language for representing deterministic and fully observable planning problems. A classical planning problem modeled in PDDL is a tuple $\langle \mathcal{D}, \Pi_{\mathcal{D}} \rangle$, where $\mathcal{D}$ represents the planning domain, consisting of a set of predicate symbols and action schemas; $\Pi_{\mathcal{D}}$ 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 $\mathcal{G}$ 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). 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).

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 and Nagarajan 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. 2023a,b; Huang et al. 2023).

3 Methodology & Experimental Design

3.1 Extended PlanBench Dataset

PlanBench (see Figure 2), as introduced by Valmeekam et al. (2023a), 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. To better support both training and evaluating LLMs' planning skills, we extend the dataset as follows:

1. New Domains. We expand PlanBench to include additional domains from the IPC benchmarks, increasing the total from 2 to 8 domains.

2. Longer-plan Problems. We have deliberately generated test instances with *longer (optimal) plan-lengths*[1], which is arguably the most straightforward way to generate

[1]The problem's (optimal) plan-length is approximated.

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.

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.

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.

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 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.

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. 2023; Shi et al. 2023; Yao et al. 2023). 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 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.

CoT Prompts to the Plan Response

+ Goal CoT:

```
Response:
My plan is as follows:
[PLAN]
[GOAL]
# Repeat the goal
My goal is ...
[GOAL END]
[COUNT]
# Count the remaining step
to the goal
10
[COUNT END]
put down the orange block
# Next step in the plan
[GOAL]
My goal is ...
[GOAL END]
[COUNT]
9
...
[PLAN END]
```

+ State CoT:

```
Response:
My plan is as follows:
[PLAN]
[PRECON]
# Provide grounded
conditions for the action
I can only pick up a
yellow block if the yellow
block is on the table and
the yellow block is clear.
...
[PRECON END]
put down the orange block
[EFFECT]
# Provide grounded effects
Once I put down a yellow
block, my hand becomes
empty.
[EFFECT 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.

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) Reinforcement learning (RL) has emerged as a promising approach to enhance the reasoning capabilities of LLMs. As of early 2025, *DeepSeek R1*[2] (DeepSeek-AI et al. 2025) demonstrated 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 finding, from a model rivaling state-of-the-art systems like *OpenAI o1*, highlights RL's potential to refine LLM skills in sequential reasoning. Given our focus on improving LLMs for SDM tasks, *DeepSeek R1*'s success offers valuable insight into tailoring RL strategies to enhance LLM planning skills. However, applying RL training to LLM-based plan generation poses two 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).

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]`.

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) for *reward shaping*. 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 $+1$.

[2]https://huggingface.co/deepseek-ai/DeepSeek-R1

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.

We present the LCCS-based reward shaping as one viable solution to reward sparsity in RL for LLMs in plan generation. Designing improved or alternative reward shaping approaches remains an open direction for future research.

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.

Evaluation Metrics We use three key metrics. *Executability* assesses whether each action within a plan sequence is feasible to perform given the state preceding it, meaning all preconditions are met at each step. *Goal satisfiability* checks if the final state produced by executing the plan's action sequence successfully achieves the desired goal, regardless of whether the intermediate steps were executable. A plan is considered *valid* only if it satisfies both conditions: it must be fully executable from the initial state to the final state, and the final state must meet the goal requirements. Thus, both executability and goal satisfiability are prerequisites for a plan to be deemed valid.

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 (Pallagani et al. 2023; Rossetti et al. 2024; Shah et al. 2024). We revisit this claim, examining whether the statements hold true in our extended PlanBench dataset.

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.

Domain	In-Distrib.		Long	
	valid. rate	*exec. rate*	*valid. rate*	*exec. rate*
BLOCKS	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*
BLOCKS	**0%**	**0%**
LOGISTICS	**0%**	**0%**

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.

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.

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

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 the vanilla model only got 20.1% (row 1).

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 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. Label 1 methodologically mirrors the setups of *Plansformer* and *PlanGPT*.

This high performance suggests that permutation augmentation enables the model to effectively parse unseen contexts, 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 generalization capabilities.

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 6-9). To gain insight into this outcome, we conducted a qualitative analysis to understand how self-correction learning affects the model's planning behavior. The experimental setting corresponds to row 6 in Table 2, which combines the self-correction strategy with permutation augmentation.

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

Table 3: Probing test: LLM effectively identifies mistakes.

Mistake Identification Probing Tests The preceding analysis, along with the results in Table 2 (rows 6-9), shows that the self-correction strategy does not lead to improved overall performance. To better understand this outcome, we decompose the failure into two possible causes: (1) the model fails to identify its own mistakes, or (2) it detects mistakes but is unable to correct them. To investigate the first possibility, we conducted *mistake identification probing tests* to assess the model's ability to recognize incorrect action steps.

We used the 'long' test set for evaluation. Probing instances were constructed by truncating plans immediately after either a correct (positive example) or incorrect (negative example) action. This setup frames mistake identification as a binary classification task. Precision and recall metrics were computed accordingly. The results, shown in Table 3, demonstrate that the model can reliably detect errors. Notably, when all four strategies are applied (row 9), the

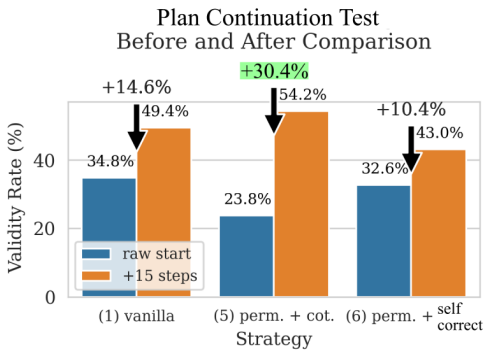

Figure 6: Validity rate of the LLM planner on the 'long' test set when provided with the first 15 actions. The model using CoT (Goal + State) showed the highest performance gain when hints were provided.

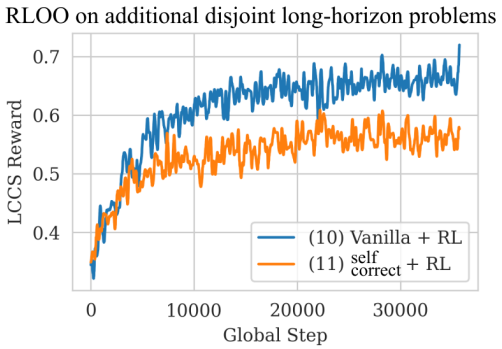

Figure 7: Reward curve of the RL training process. The LLM is further trained on 20 *disjoint* long-horizon OOD problem instances per domain. These examples are not part of the 'long' OOD test set. Despite the small training size and limited training steps, this leads to noticeable performance gains in the OOD test set.

model achieves high precision (90.5%) and recall (99.2%), indicating strong mistake recognition capabilities.

However, this detection ability does not translate into effective correction. This finding contrasts with prior findings from Ye et al. (2024), where self-correction improved model performance in math problem solving by enabling both recognition and correction of errors.

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. 2023), 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.

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 6, 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.

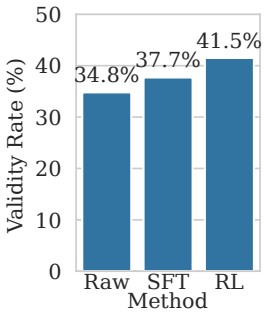

Figure 8: Comparison between RL and supervised fine-tuning (SFT), both trained on the same 20 *disjoint* long-horizon OOD instances and evaluated on the same test set. The results show that RL achieves higher validity rates.

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 Section 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

L notably improves performance under our end-to-end planning paradigm, especially on longer problems. In our setup, the RL phase was applied to already fine-tuned models, following standard practice in prior work (e.g., (DeepSeek-AI et al. 2025) and Huggingface blog[3]).

Importantly, we chose to train the model using 20 additional long-horizon OOD problem instances per domain, which were *disjoint* from the 'long' test set, rather than continuing using in-distribution, short-horizon training data. This decision was motivated by the observation that the model had already saturated on in-distribution data after SFT (see Table 2). Further training on the same distribution would provide minimal reward signal differences, yielding negligible learning gradients and ineffective RL. Moreover, this small subset — just 5% of the original 4000-instance training set — is designed to address the concern that observed improvements might simply stem from an increase in data quantity, rather than from the effectiveness of the reward-driven RL optimization.

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 stems from reward-driven optimization rather than a data advantage, we conducted an additional supervised fine-tuning (SFT) run using the same long-horizon OOD training subset. As shown in Figure 8, the RL-trained model significantly outperforms the SFT model in terms of validity rates. This finding suggests that RL fosters more comprehensive planning skills compared to supervised fine-tuning (SFT), aligning with the findings of Xiao et al. (2025).

An interesting observation is that applying RL to the vanilla model (without self-correction) resulted in faster convergence and better outcomes compared to applying RL to the self-correcting model (see Figure 7 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.

4.8 Beyond Executability: Why LLMs Still Fail to Plan?

While *executability* provides insight into the local consistency and coherence of generated plans, high executability does not guarantee that the plan achieves the desired goal — nor does it reflect the coverage of the solution space of the model, which is crucial for eventually *searching* for a valid plan. To further investigate this issue, we analyze the *goal satisfiability rate*, which offers additional insight into how well the model generates goal-directed plans. To recap, a plan is *goal-satisfiable* if the cumulative application of its actions' effects results in a state where the goal condition holds, regardless of whether the plan is fully executable.

We compute this metric alongside executability and validity in order to understand their interaction. Interestingly, the results reveal a consistent pattern: strategies that improve *executability* do not necessarily improve — and in some cases even reduce — the *goal satisfiability rate*. This observation provides a key insight into the behavior of the model under the end-to-end plan generation paradigm.

Trade-Off Between Executability and Search Coverage The discrepancy between executability and goal satisfaction reflects a deeper trade-off between local consistency and global coverage. Autoregressive language models generate plans from left to right, prioritizing sequential coherence and local state transitions. This is especially true when strategies like *State CoT* are employed, which explicitly condition each action on the preceding state to ensure realistic transitions. While such techniques improve executability, i.e., the likelihood that each step follows from the previous one, they also constrain the model's exploration of the state space.

This results in a form of **local optimization bias**: the model becomes adept at generating locally coherent steps but overly cautious, sacrificing the broader exploration needed to discover potentially less obvious trajectories that reach the global goal.

Issues of End-to-End Plan Generation These observations highlight a fundamental limitation of the end-to-end plan generation paradigm when applied to autoregressive LLMs. Unlike traditional planning systems, which explicitly construct and traverse a *search tree*, LLMs generate a single linear plan without any form of explicit backtracking, goal-checking, or search tree expansion. As a result, they lack the core capabilities provided by explicit search: they cannot systematically explore and backtrack to find alternative paths or revise their plans. This limitation highlights why higher executability does not always result in improved plan validity — because the model still lacks the ability to perform a structured search over the solution space.

5 Conclusion

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 enhance the *local consistency* of the generated plans, as reflected in the improved *executability rate*. This indicates progress towards more coherent plans that better aligned with the state transition dynamics of the world model, despite not directly leading to valid plans. We have limited our scope where the *search process* is not explicitly programmed. Within this context, we find that fine-tuning LLMs on a corpus of planning task instances struggles to foster robust planning skills beyond in-distribution instances. Nonetheless, we show that the **direction** of improving LLM planning lies in reward-driven RL optimization rather than in the design of complex prompting strategies.

[3]https://huggingface.co/blog/trl-peft

References

Ahmadian, A.; Cremer, C.; Gallé, M.; Fadaee, M.; Kreutzer, J.; Pietquin, O.; Üstün, A.; and Hooker, S. 2024. Back to Basics: Revisiting REINFORCE-Style Optimization for Learning from Human Feedback in LLMs. In *ACL (1)*, 12248–12267. Association for Computational Linguistics.

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.; and Nagarajan, V. 2024. The Pitfalls of Next-Token Prediction. In *ICML*. OpenReview.net.

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. In *NeurIPS*, 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.

DeepSeek-AI; Guo, D.; Yang, D.; Zhang, H.; et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948.

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*.

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. *CoRR*, abs/2311.05232.

Kambhampati, S.; Valmeekam, K.; Guan, L.; Verma, M.; Stechly, K.; Bhambri, S.; Saldyt, L.; and Murthy, A. 2024. Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. In *ICML*. OpenReview.net.

Katz, M.; Kokel, H.; Srinivas, K.; and Sohrabi, S. 2024. Thought of Search: Planning with Language Models Through The Lens of Efficiency. In *NeurIPS*.

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, 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.

Pallagani, V.; Muppasani, B.; Srivastava, B.; Rossi, F.; Horesh, L.; Murugesan, K.; Loreggia, A.; Fabiano, F.; Joseph, R.; and Kethepalli, Y. 2023. Plansformer Tool: Demonstrating Generation of Symbolic Plans Using Transformers. In *IJCAI*, 7158–7162. ijcai.org.

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 *ICAPS*, 500–508. AAAI Press.

Shah, K.; Dikkala, N.; Wang, X.; and Panigrahy, R. 2024. Causal language modeling can elicit search and reasoning capabilities on logic puzzles. In *NeurIPS*.

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.

Valmeekam, K.; Marquez, M.; Hernandez, A. O.; Sreedharan, S.; and Kambhampati, S. 2023a. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In *NeurIPS*.

Valmeekam, K.; Marquez, M.; Sreedharan, S.; and Kambhampati, S. 2023b. On the Planning Abilities of Large Language Models - A Critical Investigation. In *NeurIPS*.

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

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

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E. H.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*.

Xiao, T.; Yuan, Y.; Li, M.; Chen, Z.; and Honavar, V. G. 2025. On a Connection Between Imitation Learning and RLHF. In *The Thirteenth International Conference on Learning Representations*.

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. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *NeurIPS*.

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.