Enhancing the Planning Capabilities of Large Language Models by Building External World Models

Edwin Chen [1] **Xiaoyan Li** [1] **Colin Bellinger** [2] **Yunli Wang** [2*]
[1] University of Toronto, Toronto, ON, Canada
[2] National Research Council Canada, Ottawa, ON, Canada
edwinhy.chen@mail.utoronto.ca, xiaoy.li@mail.utoronto.ca
Colin.Bellinger@nrc-cnrc.gc.ca, *Yunli.Wang@nrc-cnrc.gc.ca

Abstract

Large Language Models (LLMs) possess a huge amount of knowledge but struggle with multi-step planning even in toy environments due to the limitations of their static internal world model. We introduce a novel approach where an LLM serves as a "world model builder", constructing and iteratively refining an explicit, external world model. The core of our approach is a state transition function, that is initially generated by the LLM and is refined using feedback from interactions with the environment. This refinement is made possible by accumulating test cases from past experiences allowing us to treat the construction of the world model as a program synthesis problem. We demonstrate the efficacy of our method on the Blocksworld benchmark and introduce a novel ColorMixing dataset that is designed to evaluate multi-step reasoning and planning. Our experimental results show that our method, using GPT-4 and LLaMA3-70B, achieves perfect accuracy on Blocksworld tasks and significantly outperforms baseline methods, especially in terms of planning success and LLM queries. This paper presents a robust methodology for enhancing LLM planning via a learnable external world model and contributes a new benchmark for evaluating such capabilities.[1]

1 Introduction

Large Language Models (LLMs), trained on extensive internet data, have acquired broad commonsense knowledge that enables their application across diverse domains. These models are increasingly employed in critical areas such as medical diagnosis, autonomous driving, chemical experimentation, and intelligent assistance systems. Despite their versatility, reasoning remains a fundamental limitation for LLMs, particularly in complex, multi-step decision-making tasks (Pallagani et al., 2024; Kambhampati et al., 2024). To address this, various prompt-based methods, including Chain-of-Thought (CoT)(Wei et al., 2022), Self-consistency CoT(Wang et al., 2023), Tree of thoughts (Yao et al., 2023a), ReAct (Yao et al., 2023b), and Reflexion (Shinn et al., 2023), have been developed to enhance LLMs' reasoning capabilities. These approaches have demonstrated significant improvements in structured tasks like arithmetic reasoning. However, prompt-based reasoning methods lack the ability to explicitly predict future states, which is essential for effective planning.

LLMs still struggle with tasks requiring multi-step planning or domain-specific knowledge, exhibiting several key limitations (Xiang et al., 2023; Huang et al., 2024). First, they frequently generate plans containing non-existent objects or impermissible actions, as they lack specific knowledge about the target environment. Second, their plans often prove suboptimal due to insufficient understanding of the underlying task mechanisms. Multiple planning benchmarks have revealed limitations in LLMs' performance across diverse problem domains (Valmeekam et al., 2023; Xie et al., 2024).

To enhance both the feasibility and optimality of generated plans, researchers have increasingly adopted world models to capture system dynamics. These models enable the prediction of action outcomes, which can be systematically integrated into the planning process to generate more reliable solutions. This capability is especially crucial for long-horizon decision-making tasks, where world models can be iteratively refined through accumulated experience to adapt to environmental changes.

Some studies utilize pre-existing simulators as world models (Liu et al., 2023), while others leverage LLMs as commonsense world models (Hao

[1]The code for our method and the ColorMixing dataset is available at https://github.com/edweenie123/WorldModelBuilder

et al., 2023; Zhao et al., 2023) or construct the world model (Guan et al., 2023).

Inspired by using LLM as the world model (Hao et al., 2023), we propose a novel approach that learns an external world model from LLM interaction trajectories. This model encodes past experiences as reusable functions, enabling more efficient planning. The framework specifically learns state transition dynamics and action prediction mechanisms from historical interactions. Through progressive refinement from simple to complex scenarios, the world model mimics human-like learning and adaptation in novel environments.

This work makes two key contributions: First, we develop and validate an effective world model learning methodology, demonstrating its performance on the established Blocksworld benchmark. Second, we introduce a new ColorMixing dataset specifically designed to evaluate LLM planning capabilities in complex, multi-step scenarios.

Our method can be used for scenario planning in hospital resource management, as well as other real-world scenario planning problems. Scenario planning involves creating a set of plausible but distinct future "scenarios" based on key uncertainties, trends, and drivers of change. Our approach takes these distinct scenarios as initial conditions and goals and uses the dynamics model to develop plans to achieve these goals.

2 Related work

LLM-based planning systems face three core challenges: grounding, plan generation, and adaptability. For grounding, agents utilize the LLM's inherent commonsense knowledge (Huang et al., 2022) to bridge abstract concepts with environmental specifics. In plan generation, LLMs typically function as policy networks that propose contextually appropriate next actions (Hao et al., 2023; Zhao et al., 2023). Planning can rely on the inherent reasoning capabilities of LLMs (Krishna et al., 2023) or enhance these abilities by combining ReAct and Reflexion prompting while retrieving relevant examples from memory (Zhao et al., 2024). To improve planning efficacy, researchers often employ LLMs as world models for state prediction (Hao et al., 2023) and integrate Monte Carlo Tree Search (MCTS) to efficiently explore large action spaces (Zhao et al., 2023; Zhou et al., 2024), while skill transfer from past experiences helps reduce computational complexity (Wang et al., 2024; Sun et al., 2023). The system's adaptability emerges through continuous plan refinement based on environmental feedback (Sun et al., 2023; Zhou et al., 2024).

The importance of world models in planning tasks has been recognized in various studies. For instance, Mind's Eye (Liu et al., 2023) employs a simulator as its world model, while RAP (Hao et al., 2023) leverages the world model in LLMs for simple reasoning tasks. When presented with a physical reasoning question, Mind's Eye (Liu et al., 2023) employs a computational physics engine (e.g., DeepMind's MuJoCo) to simulate potential outcomes. These simulation results are then integrated into the input, enabling language models to perform more accurate and grounded reasoning.

World models should possess the ability to plan, predict, and reason effectively about physical scenarios. LLM-DM (Guan et al., 2023) constructs an explicit world (domain) model using Planning Domain Definition Language (PDDL), a formal language for representing planning problems. Language models are primarily employed to translate natural language into PDDL, while domain experts provide feedback to refine the PDDL construction.

LLM-MCTS leverages the commonsense knowledge of LLMs to reduce the search space in large-scale task planning (Zhao et al., 2023). It treats the LLM as a commonsense world model to provide prior belief, which is updated with each action and observation in the real world. During tree search, LLM-MCTS heuristically selects promising action branches by querying the LLM. MCTS samples from the belief state (probability distribution over states) to estimate the value of actions.

RAP (Hao et al., 2023) framework employs LLMs' internal world models for reasoning and planning across diverse tasks. As a gray-box approach, RAP analyzes token-level probabilities, along with state confidence and self-evaluation heuristics, to guide the planning process. However, this method requires frequent LLM queries, resulting in computational inefficiency and high costs, particularly for proprietary models.

Unlike RAP and LLM-MCTS, our approach learns an external world model from past experiences. This model predicts future states and evaluates actions to enable efficient planning. While prior work has explored using LLMs to learn and refine functions, such as learning continuous functions for symbolic regression (Merler et al., 2024), our work focuses on learning a transition function

for discrete multi-step decision-making.

3 Method

Our proposed method enhances the planning capabilities of LLMs by using the LLM as a "world model builder". This approach extends concepts from frameworks such as RAP (Hao et al., 2023), but with a key distinction: instead of utilizing the LLM's static internal world model, our method focuses on building an external model through LLM-driven generation and refining it over time with environmental interactions. Figure 1 provides an overview of our approach, illustrating the key components and workflow of the system. The figure depicts our three-stage process: first, the LLM generates an initial state transition function based on task descriptions; second, this function is iteratively refined through real-world interactions and feedback; and finally, the refined world model enables efficient planning by predicting future states without requiring additional LLM queries during execution.

3.1 World Model Architecture

The world model consists three key components.

1. **State Transition Function (f_{ST}):** This function takes the current state s_t and an action a_t as input, and predicts the next state $\hat{s}_{t+1} = f_{ST}(s_t, a_t)$. Initially, the LLM is prompted to generate this function, for instance, as a Python program given a rough description of the environment and the possible actions within the environment. This function is the main subject of the iterative refinement process detailed in Section 3.3.

2. **State Value Function (f_{SV}):** This function estimates the utility or value of a given state s with respect to a user-defined goal. It outputs a scalar value $v(s) = f_{SV}(s)$, which guides the search process towards desirable states. In the current method, f_{SV} is implemented as a hard-coded heuristic tailored to the specific task domain and goal structure.

3. **Action Suggestion Function (f_{AS}):** Given a state s, this function suggests a set of promising actions $A_p(s) = f_{AS}(s)$ that are worth exploring. This helps to prune the search space by focusing on relevant actions. Similar to f_{SV}, f_{AS} is hard-coded based on domain knowledge to identify potentially useful actions.

The user provides the initial state and the goal. The core innovation of our method lies in the LLM-driven generation and subsequent experience-based iterative refinement of f_{ST}. While f_{SV} and f_{AS} are presently fixed, their design is crucial for effective planning.

3.2 Planning with the Learned World Model

Once the components of the world model are established, and an overall goal is defined for the task, the planning process proceeds as follows:

1. From the current state s_{curr}, the Action Suggestion Function $f_{AS}(s_{curr})$ is invoked to generate a set of promising actions $A_p(s_{curr})$.

2. For each action $a \in A_p(s_{curr})$, the State Transition Function $f_{ST}(s_{curr}, a)$ is used to predict the resulting next state $\hat{s}'$.

3. This process is applied recursively to construct a search tree, where nodes represent states and edges represent actions.

4. The State Value Function $f_{SV}(\hat{s}')$ is used to evaluate the desirability of states encountered in the search tree (like $\hat{s}'$), particularly leaf nodes or states at a certain depth, in relation to the overall goal.

5. A search algorithm (e.g., Depth-First Search (DFS), Monte Carlo Tree Search (MCTS)) traverses this tree to identify an action sequence $(a_0, a_1, \ldots, a_k)$ that is expected to lead to a state with the highest value or achieve the goal.

This planning mechanism relies on the explicit world model functions, allowing for systematic exploration of future possibilities towards the given goal. Leveraging the learned world model, our method uses a search algorithm to explore multiple world branches and estimate the value of a sequence of actions.

3.3 Iterative Refinement of the State Transition Function

A key aspect of our method is the continuous improvement of the State Transition Function (f_{ST}) based on experiences gathered from interacting with the real environment. This process treats the

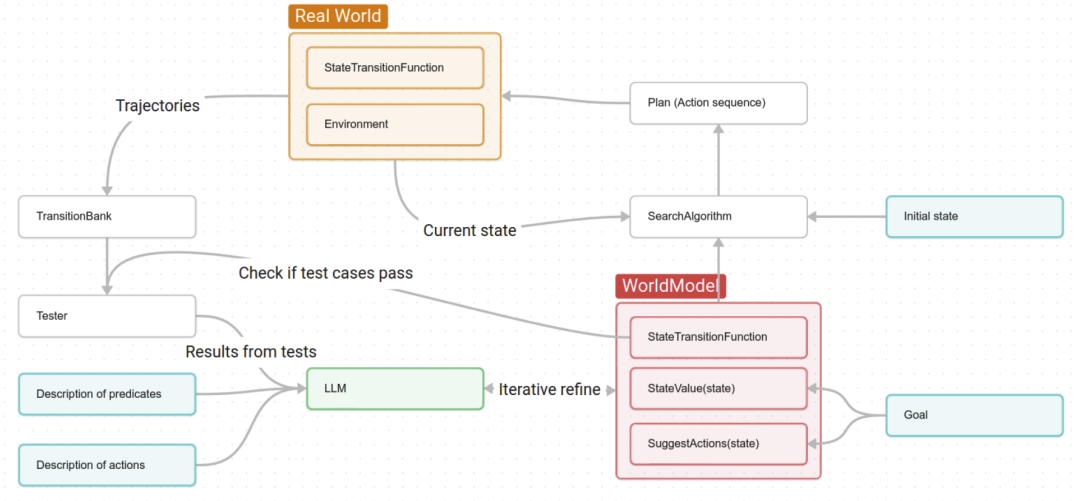

Figure 1: Overview of our proposed method for enhancing LLM planning capabilities with an external world model. The approach consists of three main components: (1) LLM-driven generation of a state transition function, (2) iterative refinement based on environmental interactions, and (3) efficient planning using the learned world model. This framework enables more accurate multi-step reasoning while reducing computational costs compared to approaches that rely solely on LLM queries for state prediction.

generation of f_{ST} as an iterative program synthesis problem.

1. **Experience Collection:** The agent executes an action (e.g., the first action a_0 from the generated plan) in the real environment. The environment then transitions from state s_t to a true subsequent state s_{t+1} according to its ground-truth dynamics. This interaction yields an experience tuple $((s_t, a_t), s_{t+1})$, which serves as a test case for f_{ST}. These test cases are accumulated in a "transition bank".

2. **Evaluation:** The current f_{ST} is evaluated against all test cases stored in the transition bank. A test case $((s_t, a_t), s_{t+1})$ is considered a failure if the predicted next state $\hat{s}_{t+1} = f_{ST}(s_t, a_t)$ does not match the observed next state s_{t+1}, or if the actual next state s_{t+1} is sufficiently different from the predicted next state $\hat{s}_{t+1}$ according to some user-defined state similarity metric.

3. **LLM-based Refinement:** The set of failing test cases (i.e., input-output pairs that f_{ST} incorrectly predicted) is provided as feedback to the LLM. The LLM is then prompted to revise or debug the f_{ST} (e.g., its Python code implementation) to correctly handle these failing instances, while ideally preserving its accuracy on previously successful cases.

4. **Iteration:** The refined f_{ST} is then re-evaluated against the transition bank. This cycle of evaluation and LLM-based refinement is repeated, progressively improving the accuracy of f_{ST}. The process can continue until all test cases pass, a predefined accuracy threshold is met, or a computational budget (e.g., number of LLM queries) is exhausted.

Through this iterative loop, the f_{ST} becomes an increasingly accurate approximation of the real world's state transition dynamics.

Our world model refinement loop is compatible with both deterministic and probabilistic transition rules. It can iteratively query the LLM, validate predicted outcomes against examples, and revise the rule as needed. This flexibility makes our approach directly applicable to planning under uncertainty—not just in fully deterministic settings.

3.4 Addressing Limitations of Existing Approaches

Our proposed methodology directly addresses several limitations observed in prior LLM-based planning approaches, such as RAP:

1. **Performance Improvement with Experience:** Unlike systems where the LLM's internal world model remains static, our approach allows the explicit f_{ST} to be continuously refined and improved as more interaction data is collected. This enables the agent's planning accuracy to increase with experience, mimicking a crucial aspect of human learning.

2. **Reduced LLM Query Cost during Planning:** In RAP, generating the search tree often requires querying the LLM at each state to predict outcomes of actions. Our method shifts the primary LLM usage to the initial generation and subsequent off-line refinement of the f_{ST}. Once f_{ST} is learned (e.g., as an executable Python function), it can be called repeatedly during the planning phase (Section 3.2) without incurring additional LLM query costs for each state transition prediction. This significantly reduces the computational expense and latency associated with LLM queries during the search process, making deeper and broader searches more feasible and aligning with the objective of maximizing performance while minimizing LLM interactions.

By externalizing and refining the world model, particularly the state transition dynamics, our method aims to achieve more robust, adaptable, and efficient planning with LLMs.

4 Experiments

We evaluate our method and the baselines on two datasets: the classic Blocksworld benchmark (Valmeekam et al., 2023), widely used for goal-conditioned symbolic planning and reasoning tasks, and our own ColorMixing dataset.

4.1 Blocksworld

Blocksworld is a classic symbolic planning domain involving a set of colored blocks stacked on a table. The agent's goal is to transform an initial block configuration into a specified goal configuration using a sequence of primitive actions such as pick up, put down, stack, and unstack. We employ subsets of this benchmark, specifically 30% from three-step problems (step 2, step 4, and step 6), to train our world model, reserving the remaining 70% for testing. In Blocksworld, the state encodes the configuration of blocks (e.g., `on`, `clear`, `ontable`), and actions include the standard operations: `pick-up`, `put-down`, `stack`, and `unstack`.

4.2 ColorMixing

The *ColorMixing Dataset* is a synthetic benchmark developed to assess the reasoning and planning capabilities of LLMs in a controlled color mixing environment. In this setting, an agent interacts with six virtual beakers, each described by color contents and volume, and is tasked with achieving a specified target color in a chosen beaker through a sequence of actions. The initial state of the environment includes five beakers prefilled with primary and neutral colors: red, green, blue, white, and black, and one empty beaker designated for mixing. We use a discrete action space composed of symbolic operations with arguments defined by integer values (e.g., beaker indices and paint amounts). Figure 2 illustrates the color mixing process. Each state is represented as a list of strings in the form: `contains <beaker_id> <R> <G> <B> <amount>`, where `<beaker_id>` is an integer from 1 to 6, `<R>`, `<G>`, and `<B>` are RGB values ranging from 0 to 255, and `<amount>` denotes the volume (0 - 200). The goal state, also in this format, specifies the desired color mixture and amount in a target beaker. The ColorMixing environment enables evaluation of both low-level world model predictions and high-level planning behavior.

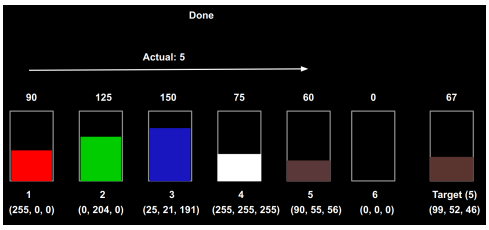

Figure 2: Visualization of the color mixing task. The initial state includes five prefilled beakers and one empty beaker. The agent aims to produce a target color in a designated beaker through sequential actions.

In the ColorMixing task, each state represents the color and volume of paint in six beakers. Actions include operations: `pour` and `done`. We generated 100 data files, each containing the initial color states of six beakers along with a corresponding target (goal) color state. We randomly selected 30% of the files as training data, and used the remaining 70% as test cases.

While the ColorMixing environment is deterministic at the transition level, it introduces uncertainty in the number of actions required to reach the goal. The agent must explore and compare action sequences of varying lengths, reflecting a form of procedural uncertainty that aligns closely with the goals of scenario planning.

4.3 Results on Blocksworld

The following subsections present the results of our method on the Blocksworld domain, focusing on both the training (refinement) phase of the world model and the task performance on the testing data.

4.3.1 Training Phase

The quality of the refined world model is evaluated using two metrics: experience accuracy, measured by the pass rate across individual state-action transitions in the transition bank, and state transition accuracy, which assesses the model's ability to simulate full state trajectories given sequences of actions from the test set. List 1 in Appendix A shows an example of a state transition function for Blocksworld refined by GPT-4.

Figure 3 illustrates the training progression of the world model in our method, refined using GPT-3.5 on the Blocksworld. The top subplot depicts the number of LLM queries made per training instance. Each dot corresponds to a specific training level, with red markers indicating instances where the algorithm failed to achieve the goal state. Notably, many levels required up to 15 GPT-3.5 queries to refine the state transition function, highlighting the limited capability of GPT-3.5 in generating accurate state transition functions. The middle subplot shows the accuracy of the world model's learned state transition function f_{ST}, measured by comparing the predicted next state to the ground-truth next state for each (s_t, a_t) in the transition bank, during training. This metric reflects the model's internal learning quality across training iterations. A sharp improvement is observed around the 16th training level, after which the accuracy plateaus, indicating that further refinement yields diminishing returns. Despite continued LLM queries, GPT-3.5 fails to consistently improve the quality of the learned transition function beyond this point.

The bottom subplot shows state transition accuracy on the held-out test set. Given an initial state and ground-truth action sequence, the learned world model predicts the resulting state after each action, and accuracy is computed as the average similarity across all predicted and ground-truth next states. Notably, goal achievement and transition accuracy may diverge, as goal completion is based on a partial specification (e.g., "`on a c`" and "`on b a`"), while state transition accuracy evaluates the entire predicted state, including predicates like "`handempty`", "`clear d`", "`ontable d`", and others. Thus, a goal may be achieved even if the overall state prediction accuracy is relatively low.

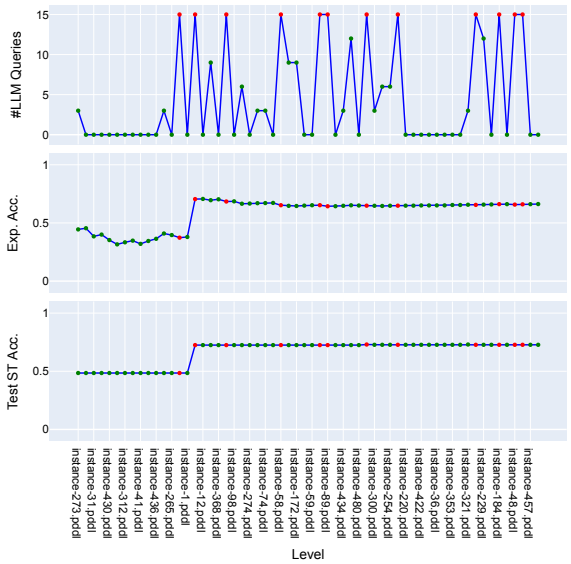

Figure 3: Training progression of the world model in our method, refined using GPT-3.5, on the Blocksworld. The x-axis denotes the training instances (levels).

We observe that GPT-3.5 demonstrates some reasoning capability in refining the world model. However, even after multiple refinement steps, the state transition accuracy remains below 0.8, indicating its limited effectiveness in learning accurate transition dynamics. Figure 5 in Appendix B presents the training progression of the world model in our method, refined using GPT-4. Compared to the case with GPT-3.5, GPT-4 demonstrates significantly stronger reasoning capabilities. Notably, the world model is successfully refined with only a single LLM query (specifically, updating the state transition function). After this refinement, the model consistently predicts accurate next states when paired with the search algorithm.

Figure 6 in Appendix B illustrates the training progression of the world model when refined using LLaMA3-70B. In this case, the model is refined based on feedback from only two training instances, requiring a total of 9 LLM queries. Compared to GPT-4, LLaMA3-70B achieves lower accuracy in modeling the state transition function, but it still outperforms GPT-3.5 in both experience and test accuracy.

4.3.2 Testing Phase

For overall task performance, we report the goal achievement accuracy, defined as the ratio of successful instances (i.e., the final state matches the goal) to the total number of test instances. Our

method was evaluated on the testing set, consisting of 70% of the instances from each step. Table 1 presents a comparison between our approach and several baselines: GPT-3.5 + CoT, GPT-4 + CoT, and RAP (Hao et al., 2023). Since RAP currently supports only LLaMA-based models, we use LLaMA3-70B as the backbone for the RAP baseline. All baseline implementations are obtained from the LLM Reasoners benchmark (Hao et al., 2024). Following the evaluation protocol used for GPT-3.5 + CoT, GPT-4 + CoT, and RAP, we use the VAL tool, a command-line validator for checking the correctness of plans in classical PDDL-based planning domains (Fox and Long, 2003), to assess each method's performance.

As shown in Table 1, all three variants of our method consistently outperform the two CoT-based GPT baselines across all steps. The world model refined with GPT-3.5 achieves lower accuracy, while those refined using GPT-4 and LLaMA3-70B reach perfect accuracy, surpassing RAP. Although the CoT-based baselines are less accurate, they require only a single LLM query. In contrast, RAP issues two LLM queries per candidate action, one for action generation and one for next-state prediction, resulting in approximately $N \times d \times 2 = 80$ LLaMA3 queries for $N = 10$ rollouts and $d = 4$ actions. Our method requires only 9 LLaMA3 queries for refinement, as shown in Figure 6 in Appendix B. Notably, RAP cannot support GPT-based models due to its reliance on token-level log probabilities, which are not accessible via the OpenAI API, making direct comparison infeasible.

Method	**Accuracy**			
	Step 2	**Step 4**	**Step 6**	**Avg.**
GPT-3.5 + CoT	20.00%	13.50%	4.69%	12.73%
GPT-4 + CoT	20.01%	14.50%	4.94%	13.15%
RAP (LLaMA3)	89.47%	85.00%	80%	84.49%
Ours (GPT-3.5)	95.24%	87.5%	78.75%	87.16%
Ours (GPT-4)	100%	100%	100%	100%
Ours (LLaMA3)	100%	100%	100%	100%

Table 1: Performance comparison across Step 2, Step 4, and Step 6 tasks for our three methods and three baselines.

4.3.3 Runtime Analysis

Table 2 compares training and inference times of our methods against several baselines. All experiments were conducted on a machine equipped with an Intel Xeon W-2255 CPU (10 cores, 20 threads, 3.70 GHz) and an NVIDIA Quadro RTX 6000 GPU with 24 GB of memory. Among our variants, the world model refined using GPT-4 achieves the lowest overall runtime, requiring only 38.57 seconds for training and 1.18 seconds for inference. In contrast, the LLaMA3-70B variant incurs the highest training time (3084.02 seconds) due to its local execution, but maintains a comparable inference time of 1.28 seconds. Despite the higher upfront cost, our method with LLaMA3-70B significantly outperforms overall efficiency of RAP. This efficiency stems from our approach refining the world model using LLaMA3-70B only during the training phase, after which a symbolic DFS planner is used at test time. In contrast, RAP repeatedly queries LLaMA3-70B during planning, resulting in substantially higher cumulative runtime.

Method	**Training Time (s)**	**Inference Time (s)**
GPT-3.5 + CoT†	–	593.8
GPT-4 + CoT†	–	586.6
RAP (LLaMA3)	–	1,518,120
Ours (GPT-3.5)†	675.97	1.17
Ours (GPT-4)†	38.57	1.18
Ours (LLaMA3)	3084.02	1.28

Table 2: Comparison of training and inference times (in seconds) for different methods. Training time refers to the refinement of the world model, while inference time measures the execution time of the model (or refined model) for task-solving. †Inference for API-based models (GPT-3.5 and GPT-4) includes network latency and remote GPU processing.

4.4 Results on ColorMixing

In the ColorMixing benchmark, we use a similarity score to measure the closeness between a predicted state and its corresponding ground-truth state. Similarity is computed by comparing corresponding beakers based on both RGB color and paint volume. Specifically, we define a weighted similarity function that combines color similarity, measured by the Euclidean distance in RGB space, and volume similarity, measured by the normalized absolute difference. A higher weight is assigned to the color component. The overall similarity between two states is computed as the average of the beaker-wise similarities. The world model is considered sufficiently accurate if the similarity between the

predicted final state and the goal state exceeds a threshold of 0.95.

4.4.1 Training Phase

We evaluate the refinement quality of the world model using three metrics: experience similarity score, state transition similarity score, and goal state similarity score. These metrics respectively, assess the model's accuracy on training state transitions, its ability to generalize to unseen action sequences, and its alignment with the desired goals. List 2 in Appendix A presents an example of a state transition function for ColorMixing, refined using GPT-4.

In the ColorMixing experiments, we use GPT-4 to refine the world model and compare our method with the GPT-4 + CoT baseline. Figure 4 presents the training progression of our approach. Similar to the Blocksworld, the top subplot shows the number of LLM queries required to refine the world model. The second subplot presents the average similarity score across all state transitions observed during training episodes, reflecting how accurately the model predicts intermediate states. A predicted color is considered a successful match if its similarity exceeds a threshold of 0.95. Matched cases are colored green, while failed instances are red.

The third subplot captures the testing performance of the learned state transition function. Unlike Blocksworld, where action sequences are fixed, we randomly sample a state–action pair and compare the predicted next state with the ground-truth. This simulates a realistic setting where the number of mixing steps is not predefined and must be inferred by the model. In both training and testing evaluations, the state transition function achieves an average similarity score above 0.97, indicating strong predictive accuracy. The fourth subplot shows the number of steps taken to reach the goal. Notably, although the average state similarity during testing is above the threshold (0.95), there are cases where the algorithm still fails to achieve the goal. This discrepancy is highlighted in the bottom subplot, which shows the final goal similarity score, computed based solely on the target beaker.

The mismatch arises because the similarity score for the state transition reflects the average similarity across all six beakers, while goal achievement is determined only by the similarity of the target beaker. Thus, if the target beaker's similarity falls below the threshold, the episode is marked as a failure, even if the average similarity across all beakers remains high.

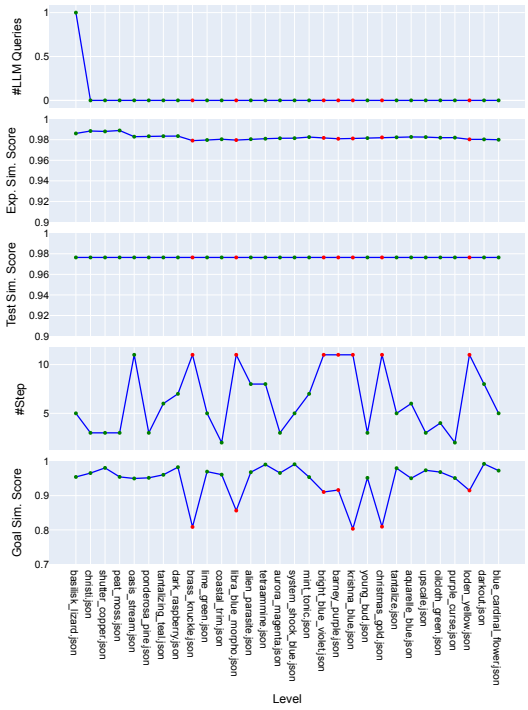

Figure 4: Training progression of the world model in our method, refined using GPT-4, on the ColorMixing Data.

4.4.2 Testing Phase

We evaluate the task performance based on two metrics: the average goal state similarity score and the pass rate, defined as the ratio of instances where the goal similarity score exceeds a predefined threshold to the total number of instances.

Table 3 presents the testing results of our method compared to the GPT-4 + CoT baseline. Due to the increased complexity of the ColorMixing task relative to Blocksworld, both GPT-3.5 and LLaMA3-70B fail to produce reliable state transition functions. Additionally, RAP cannot be used as a baseline in this setting, as it currently supports only LLaMA-based LLMs. Another potential baseline involves using the LLM's internal world model to simulate state transitions directly; however, this approach is prohibitively expensive, as it requires repeated LLM queries for each action and state transition, resulting in substantial computational overhead. For this reason, we exclude it from our evaluation. Consequently, we exclusively employ GPT-4 for refining the world model in this setting. The results in Table 3 highlight the superior performance of our approach, which achieves a perfect pass rate across all test instances.

Method	Goal Similarity Score	Pass Rate
GPT-4 + CoT	53.45%	0%
Ours (GPT-4)	98.10%	100%

Table 3: Comparison between GPT-4 + CoT and our method using GPT-4 on the ColorMixing task.

5 Conclusion

LLMs have shown promise as policies for complex decision-making tasks, but their effectiveness is limited by inaccuracies in their internal world models, leading to inefficient planning. To address this, we propose learning an external world model that dynamically improves multi-step reasoning by predicting future states at each decision point. Our experiments on Blocksworld and ColorMixing demonstrate significant improvements, achieving perfect success rates across all difficulty levels in Blocksworld while outperforming LLM-based world models.

Our work introduces both a novel approach to enhancing LLM-based planning and a new dataset for evaluating multi-step decision-making tasks. However, our experiments are currently focused on the Blocksworld and ColorMixing datasets. Further evaluation on more diverse and complex environments, such as VirtualHome or GSM8k, is necessary to fully assess the generalizability and effectiveness of our method. In future work, we plan to integrate our external world model with more efficient search algorithms to better handle tasks with large and complex action spaces.

Our method involves learning a dynamics model using the LLM and then utilizing the model to plan. It can be used for human-led scenario planning. Here, a human devises plausible but uncertain future scenarios, such as a shortage of gold flake paint or an increase in the cost of gold flake paint due to commodity fluctuations. This leads to a different set of initial conditions, which can be explored using our model-based planner. In this case, our method is used as a simulator for scenario planning under uncertainty. Since the world model is learned through interaction with the environment, the simulator can adapt to different initial conditions.

Limitations

Our approach, while delivering promising results, has several limitations that offers avenues for future work. Currently, the LLM is only responsible for generating and refining the State Transition Function (f_{ST}), while the State Value (f_{SV}) and Action Suggestion (f_{AS}) functions are hard coded. Extending the LLM's involvement to also learn these components of the world model would enhance autonomy and may improve performance. Moreover, the current system only plans on a predefined set of low-level actions; future work could explore enabling the LLM to learn higher-order actions consisting of several low-level actions allowing for hierarchical planning and potentially improving planning ability in complex environments.

Acknowledgments

This project is supported by research funding from the National Research Council of Canada's Artificial Intelligence for Design Program.

References

Maria Fox and Derek Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124.

Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Leveraging pretrained large language models to construct and utilize world models for model-based task planning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. 2024. LLM reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. In *First Conference on Language Modeling*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716*.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. 2024.

Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

Satyapriya Krishna, Jiaqi Ma, Dylan Slack, Asma Ghandeharioun, Sameer Singh, and Himabindu Lakkaraju. 2023. Post hoc explanations of language models can improve language models. *Advances in Neural Information Processing Systems*, 36:65468–65483.

Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M. Dai. 2023. Mind's eye: Grounded language model reasoning through simulation. In *The Eleventh International Conference on Learning Representations*.

Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. 2024. In-context symbolic regression: Leveraging large language models for function discovery. *arXiv preprint arXiv:2404.19094*.

Vishal Pallagani, Bharath Chandra Muppasani, Kaushik Roy, Francesco Fabiano, Andrea Loreggia, Keerthiram Murugesan, Biplav Srivastava, Francesca Rossi, Lior Horesh, and Amit Sheth. 2024. On the prospects of incorporating large language models (LLMs) in automated planning and scheduling (APS). In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 432–444.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 58202–58245. Curran Associates, Inc.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2023. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Jiannan Xiang, Tianhua Tao, Yi Gu, Tianmin Shu, Zirui Wang, Zichao Yang, and Zhiting Hu. 2023. Language models meet world models: Embodied experiences enhance language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: a benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36:31967–31987.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

A Refined State Transition

A.1 State Transition for Blocksworld

Listing 1: Refined state transition function for the Blocksworld domain refined using GPT-4.

```
def state_transition(self, state, action
    ):
    words = action.split()
    action_type = words[0]
    params = words[1:]
    next_state = set(state)

    if action_type == "pick-up":
```

```
        block = params[0]
        if f"clear {block}" in
    next_state and f"ontable {block}" in
     next_state and "handempty" in
    next_state:
            next_state.discard(f"clear {
    block}")
            next_state.discard(f"ontable
     {block}")
            next_state.discard("
    handempty")
            next_state.add(f"holding {
    block}")

    elif action_type == "put-down":
        block = params[0]
        if f"holding {block}" in
    next_state:
            next_state.discard(f"holding
     {block}")
            next_state.add(f"ontable {
    block}")
            next_state.add(f"clear {
    block}")
            next_state.add("handempty")

    elif action_type == "stack":
        block, target = params
        if f"holding {block}" in
    next_state and f"clear {target}" in
    next_state:
            next_state.discard(f"holding
     {block}")
            next_state.discard(f"clear {
    target}")
            next_state.add(f"on {block}
    {target}")
            next_state.add(f"clear {
    block}")
            next_state.add("handempty")

    elif action_type == "unstack":
        block, base = params
        if f"on {block} {base}" in
    next_state and f"clear {block}" in
    next_state and "handempty" in
    next_state:
            next_state.discard(f"on {
    block} {base}")
            next_state.discard(f"clear {
    block}")
            next_state.discard("
    handempty")
            next_state.add(f"holding {
    block}")
            next_state.add(f"clear {base
    }")

    return next_state
```

A.2 State Transition for ColorMixing

Listing 2: State transition function for the ColorMixing environment refined using GPT-4.

```
def state_transition(self, state, action
    ):
    def find_element(my_set, condition):
        for element in my_set:
            if condition(element):
                return element
        return None

    words = action.split()
    action_type = words[0]
    params = words[1:]

    new_state = set(state)

    if action_type == "pour":
        src_idx, tgt_idx, amt = [int(x)
    for x in params]
        src_contains = find_element(
    state, lambda x: x.split()[1] == str
    (src_idx))
        tgt_contains = find_element(
    state, lambda x: x.split()[1] == str
    (tgt_idx))

        src_r, src_g, src_b, src_amt = [
    int(x) for x in src_contains.split()
    [2:]]
        tgt_r, tgt_g, tgt_b, tgt_amt = [
    int(x) for x in tgt_contains.split()
    [2:]]

        # Calculate the amount of paint
    after pouring
        new_src_amt = src_amt - amt
        new_tgt_amt = tgt_amt + amt

        # Calculate the new color in the
     target beaker
        new_tgt_r = (tgt_r * tgt_amt +
    src_r * amt) // new_tgt_amt
        new_tgt_g = (tgt_g * tgt_amt +
    src_g * amt) // new_tgt_amt
        new_tgt_b = (tgt_b * tgt_amt +
    src_b * amt) // new_tgt_amt

        # Update the state with the new
    values
        new_state.discard(src_contains)
        new_state.discard(tgt_contains)

        new_state.add(f"contains {
    src_idx} {src_r} {src_g} {src_b} {
    new_src_amt}")
        new_state.add(f"contains {
    tgt_idx} {new_tgt_r} {new_tgt_g} {
    new_tgt_b} {new_tgt_amt}")

    return new_state
```

B Training Performance on Blocksworld

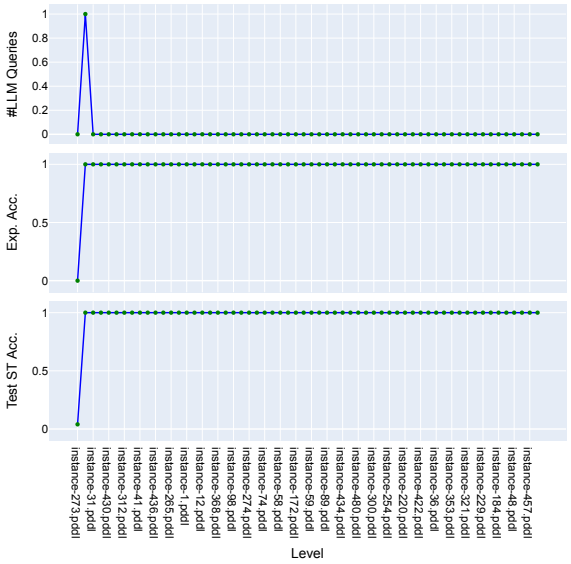

Figure 5: Training progression of the world model in our method, refined using GPT-4, on the Blocksworld domain.

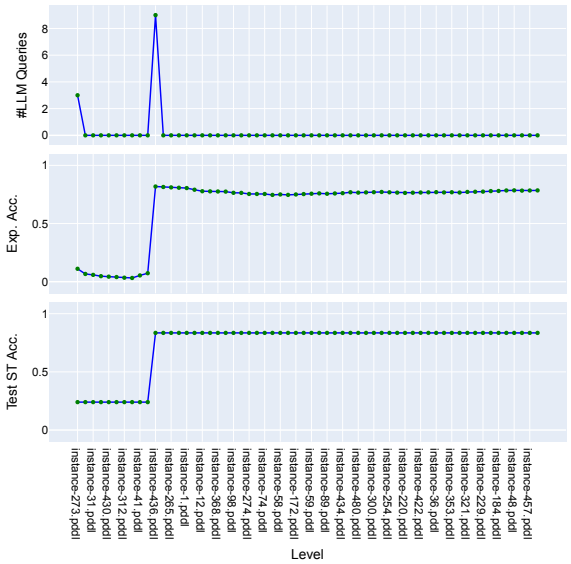

Figure 6: Training progression of the world model in the Blocksworld domain, refined using LLaMA3-70B. The plot illustrates how the model improves over training iterations.