PRompt Optimization in Multi-Step Tasks (PROMST): Integrating Human Feedback and Preference Alignment

Anonymous ACL submission

Abstract

Prompt optimization aims to find the best prompt to a large language model (LLM) for a given task. LLMs have been successfully used to help find and improve prompt candidates for single-step tasks. However, realistic tasks for agents are multi-step and introduce new challenges: (1) Prompt content is likely to be more extensive and complex, making it more difficult for LLMs to analyze errors, (2) the impact of an individual step is difficult to evaluate, and (3) different people may have varied preferences about task execution. While humans struggle to optimize prompts, they are good at providing feedback about LLM outputs; we therefore introduce a new LLM-driven discrete prompt optimization framework that incorporates human-designed feedback rules to automatically offer direct suggestions for improvement. We also use an extra learned heuristic model that predicts prompt performance to efficiently sample from prompt candidates. This approach significantly outperforms both human-engineered prompts and several other prompt optimization methods across 11 representative multi-step tasks (an average 10.6%-29.3% improvement to current best methods on five LLMs respectively). We further show that the score function for tasks can be modified to better align with individual preferences. We believe our work can serve as a benchmark for automatic prompt optimization for LLM-driven multi-step tasks.

1 Introduction

The performance of large language models (LLMs) on a given task is sensitive to the prompt, so prompt engineering aims to create prompts that fully leverage the capabilities of LLMs. Due to the lack of access to model parameters for black-box LLMs, techniques for automatic prompt optimization have primarily focused on searching over the vast discrete space of tokenized language inputs (Cheng et al., 2023). Recent studies have shown that LLMs, combined with evolutionary algorithms, can help with this search by reasoning over errors made using existing prompts to suggest edits or generate new candidate prompts (Pryzant et al., 2023; Wang et al., 2023; Yang et al., 2023). These approaches have been evaluated on relatively simple one-step tasks, such as mathematical calculations (Roy and Roth, 2016; Cobbe et al., 2021), instruction induction (Honovich et al., 2022), and factual analysis (Wu et al., 2023b). The associated prompts are also relatively short, usually one to three sentences.

In this work, we aim to optimize prompts for LLM-driven agents solving multi-step tasks and propose a method called PRompt Optimization in Multi-Step Tasks (PROMST). In these tasks, an LLM is used to decide a system's actions (e.g., virtual software Wu et al. 2023a; Zhou et al. 2023a or real robots Chen et al. 2023a; Firoozi et al. 2023) as it interacts with an environment over multiple steps (Abdulhai et al., 2023). Engineering good prompts is hard due to the typical prompt length (300+ tokens) and individual task constraints and rules. Further, human preferences can introduce slight changes to how the task is scored, resulting in notably different optimal prompts. The prompts needed for multi-step tasks are more complex and it is hard to judge the long-horizon correctness of a single action. This difficulty hinders LLMs from automatically reasoning over errors and producing better prompts, which in turn reduces the effectiveness of current methods for automated prompt optimization. As a result, prompt optimization in multi-step tasks is still an open challenge.

Considering that humans excel in analyzing errors and incorporating relevant domain knowledge into feedback, we formalize PROMST as a framework involving human input, as shown in Figure 1. Here, during the multi-step agent-environment interactions, the agent (indicated by 'TaskLLM' in Figure 1) sometimes makes errors and fails the task. While some work has used LLMs to evaluate errors, we instead use human-designed feedback rules con-

structed a priori that address different types of errors. Depending on the error, feedback is automatically generated and passed as additional context to an LLM that is responsible for producing a new set of candidate prompts (indicated by 'PromptLLM' in Figure 1). We also use human-designed rules as part of the evaluation for each task; a score is assigned to each prompt indicating the agent's task performance given that prompt. Since the evaluation of many candidate prompts for multi-step tasks in environments can be expensive, we fine-tune a score prediction model online using prompt-score pairs which can be used as a heuristic to select a subset of the candidate prompts to evaluate.

Our experiments in 11 tasks show that the integration of human feedback and the score model greatly improves the prompt optimization process (10.6%-29.3% relative improvements over all baseline methods across different LLMs). Prompts optimized by PROMST achieve the best performance on most tasks. We further show that the human-designed evaluation rules can be used to help align task performance with human preferences.

In summary, our contributions are threefold: (1) To our best knowledge, PROMST is the first to explore automatic prompt optimization in multi-step agent tasks. We release all codes and prompts for 11 multi-step environments, which may serve as a benchmark for future research. (2) We show that the integration of human feedback and a fine-tuned score model outperforms existing methods across various tasks and LLMs. (3) We show that human-designed rules for task evaluation can help align optimized prompts with human preferences.

2 Related Work

Prompt Optimization To improve performance of black-box API models, it is useful to engineer the discrete prompts for downstream tasks. Various 'best practices' have emerged for human-designed task prompts, such as including examples (Brown et al., 2020) or promoting reasoning chains (Kojima et al., 2022; Wei et al., 2022). However, manually designing prompts requires extensive human trial-and-error and is sub-optimal; thus, many recent works focus on automating this process. Some methods approximate the gradients (Diao et al., 2022) or emulate them via natural language (Pryzant et al., 2023). Others use edit operators to modify an initial prompt, driven either by reinforcement learning (Zhang et al., 2023) or score-guided search (Prasad et al., 2023). To help balance exploration and exploitation of prompts, several approaches have used LLM-driven evolution (Guo et al., 2023a; Fernando et al., 2023; Ma et al., 2023; Ye et al., 2023). In several works, LLMs are directly used to generate prompt candidates (Zhou et al., 2023b) often with feedback about parent prompts (Wang et al., 2023; Ma et al., 2023).

LLM Based Agents for Multi-Step Tasks There are many recent works that use LLMs for multi-step planning. LLMs are used to interact with softwares and websites (Ma et al., 2024; Wu et al., 2023a; Zhou et al., 2023a), plan robot actions (Chen et al., 2023b; Ahn et al., 2022; Huang et al., 2022a; Ma et al., 2023; Aghzal et al., 2023), and connect to external tools (Liu et al., 2023; Chen et al., 2023a; Qin et al., 2023). All these multi-step tasks require careful design of lengthy prompts to capture all the constraints and requirements.

LLM Self-reflection from Feedback In planning domains, it is useful to provide feedback about syntactic errors (Silver et al., 2023; Skreta et al., 2023), potential infinite loops (Silver et al., 2023), inadmissible actions (Huang et al., 2022a; Lin et al., 2023), failed action execution (Huang et al., 2022b), and generated trajectories (Chen et al., 2023a). Other recent work has shown that LLM-generated feedback via self-evaluation can improve performance on a variety of tasks (Yang et al., 2022; Welleck et al., 2022; Madaan et al., 2023), including prompt engineering (Wang et al., 2023) and reinforcement learning (Shinn et al., 2023; Ma et al., 2023). Our work also uses automated feedback with human-designed categories and responses.

3 Methodology

3.1 Problem Formulation

Given a base LLM $\mathcal{B}$ and a target task T, the goal of prompt optimization is to craft an optimized natural language prompt P that maximizes the performance of $\mathcal{B}$ on T. Here the prompt P consists of multiple components, such as a task description, scoring rules, and safety constraints. In multi-step tasks, the state information of the environment at each step will be transformed into a text string S and provided to the LLM $\mathcal{B}$ to make decisions. The history of state (S), action (a), and environment feedback (e) will also be reported to LLM. For the ith trial on a particular task, the probability of an

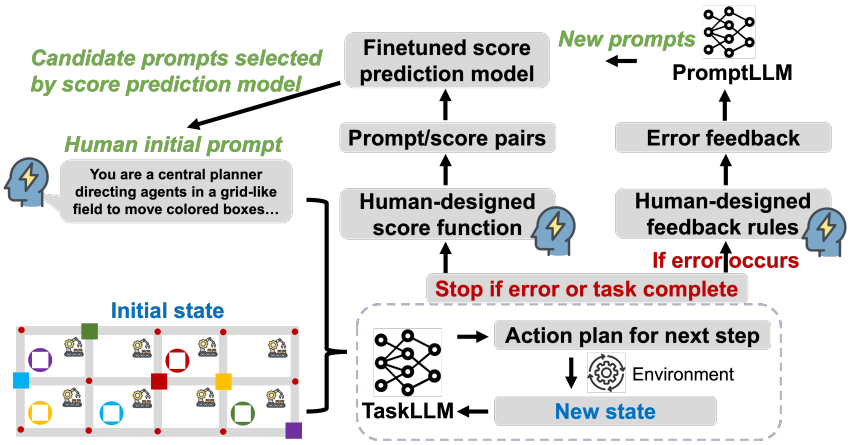

Figure 1: The PROMST framework. Given an initial human-designed prompt and the state of the environment for the current task, the TaskLLM iteratively generates an action and executes it until either an error occurs or the task is complete. Human-designed feedback rules automatically generate feedback about errors that is then provided as context to the PromptLLM when generating new prompt candidates. The task performance is scored according to a human-designed score function; this score can be used with the prompt to train a score prediction model online. Given new prompt candidates, this score prediction model is used to select a subset of candidates to evaluate for the next generation.

action sequence $[a_{i,1}, a_{i,2}, ..., a_{i,j}]$ is:

$$p_{\mathcal{B}}([a_{i,1}, a_{i,2}, \ldots, a_{i,j}]) = \prod_{k=1}^{j} p_{\mathcal{B}}(a_{i,k}|S_{i,k}, P, S_{i,k-1}, a_{i,k-1}, e_{i,k-1}, \ldots, S_{i,1}, a_{i,1}, e_{i,1}) \quad (1)$$

The sequence $[a_{i,1}, a_{i,2}, ..., a_{i,j}]$ is executed in the task environment and assigned a score based on human-designed rules or functions R. The goal of prompt optimization is to find the optimal natural language prompt P^* that maximizes a score function R:

$$P^* = \arg\max_{P \in A} \sum_{i} R(p_{\mathcal{B}}([a_{i,1}, a_{i,2}, ..., a_{i,j}])), \quad (2)$$

where A denotes the vast and complex space of all possible natural language prompts.

3.2 PROMST Framework

Figure 1 illustrates the general framework of PROMST. The goal is to more efficiently and strategically search over the vast space of possible prompts while integrating human-designed feedback of candidate prompt performance. LLMs are used in two key steps of PROMST: (1) the execution of the task via the current candidate prompt ('TaskLLM') and (2) the generation of new candidate prompts given any available feedback about the current prompt's performance on the task ('PromptLLM'). We refer to a single execution of a task as a trial. In each trial, the TaskLLM executes the task over multiple rounds of interaction with the environment; for each round, the TaskLLM is provided both the current candidate prompt and the current trial's execution history and generates the next action for the agent to take. Task execution terminates when an error is detected or the task is complete. The candidate prompt P is assigned a score for that trial via the human-designed score function. Each candidate prompt is evaluated over multiple trials in which the initial task state (e.g. number of objects, number of agents) is varied, resulting in a final average score. Once the candidate prompts have all been evaluated and assigned automatic feedback, the top performers are selected as parents for a new generation of candidate prompts. The PromptLLM uses each parent prompt and its feedback to generate new candidate prompts. This process is also described in Algorithm 1 in Appendix B. Note that the evolutionary algorithm, which has been explored by recent works (Guo et al., 2023b; Pan et al., 2023), is not the focus of this study.

Score Prediction Model In general, producing more candidate prompts per generation allows for more exploration over the space of possible prompts; however, there is a trade-off between the number of candidates per generation and the cost of evaluation, and multi-step tasks can be much more

expensive to evaluate (we query the TaskLLM for each next action). To help mitigate the evaluation cost for a generation, we learn a score prediction model online that functions as a heuristic with which to choose a subset of the generated candidate prompts for actual evaluation.

Algorithm 2 in Appendix B shows the process of implementing the score prediction model as a heuristic for filtering candidate prompts. We fine-tune a task-specific bidirectional Longformer-base (148M) (Beltagy et al., 2020) model. The prompt-score pairs on which we fine-tune are collected online during early iterations of PROMST; therefore, the score prediction model is not applied until sd^{th} generation, where sd is a hyperparameter. We continue to update the learned model at each generation with the new prompt-score pairs. To mitigate variance, we fine-tune multiple models on five rounds with the collected data following a random 4:1 train/test split. The generated prompt candidate p' will be selected for task evaluation if:

$$\begin{aligned} &\mathrm{E}[M_k(p')] + \mathrm{Var}[M_k(p')] + \mathrm{E}[\mathrm{error}_k] \\ &\geq \text{hyper_M} \times \max(D.\mathrm{score}()) \end{aligned} \tag{3}$$

where $\mathrm{E}[M_k(p')]$ and $\mathrm{Var}[M_k(p')$ are the mean and variance of predicted scores for p' from five models. The $\mathrm{E}[error_k]$ is the average testing error of five score models. The $\max(D.score())$ is the highest score of existing prompts. The hyper_M is the hyperparameter that controls the threshold.

Human-Designed Feedback Rules During task execution, the TaskLLM may encounter an error, resulting in the task being terminated. It is useful for the PromptLLM to have context about this error when generating new prompt candidates. Since automatic error analysis via LLMs is difficult for multi-step tasks (e.g. an agent stuck in an action loop), we instead use human-designed rules to automatically synthesize feedback, as shown in Figure 2. Some types of errors can be common across all tasks (e.g. syntactic errors), while others are task-specific. For types of human feedback in each task, see Appendix A.

Candidate Prompt Generator We produce new candidate prompts from a parent prompt in two steps: (1) summarizing feedback via an LLM ('SumLLM') and (2) generating new prompt candidates via an LLM provided with the summarized feedback as context ('GenLLM'). In order to encourage exploration over more diverse candidate prompts, we randomly choose 10 instances of feedback. This random selection also likely promotes more frequent errors. Given the selected feedback, SumLLM produces a summary that is included as context to GenLLM for generating new candidates. See Algorithm 3 in Appendix B for another description of this process and Appendix C for the meta-prompts used for SumLLM and GenLLM.

```
1) Syntactic error: "Here is the prompt of task
description:{prompt_task_explain}. The response {response} is in
the wrong format."

2) Stuck in the loop: "Here is the prompt and response from
previous two rounds:
Round1: {state 1 and action 1}, environment feedback after
executing the plan:{env_act_feedback_1};
Round2: {state 2 and action 2}, environment feedback after
executing the plan:{env_act_feedback_2}.
Here is the initial response generated by the task planning LLM
agent based on the concatenated prompt of task description and
current state: {response}
Based on previous rounds, here is the error feedback from humans:
It seems the LLM is stuck in the current situation, always
repeating the same answer. The task is stuck too, no box is placed
successfully in recent rounds."

3) Collision: "Your response is: {response}. The action in the
response leads to collision."

4) Move out of the grid: "Your response is: {response}. The action
in the response leads to move out of the grid."

5) Wrong picking up order: "Your response is: {response}. The
action in the response leads to wrong order of picking up goals.
Do remember pick up goal_0 first, then goal_1, goal_2, and so on."

6) Failure over query time limit: "The task is not completed over
the query time limit."

7) Invalid action: "Your response is: {response}. The action in
the response is invalid action."

8) Wrong object action: "Your response is: {response}. The action
in the response is trying to refer to nonspecific or nonexist
vehicle (truck, airplane)."
```

Figure 2: Eight examples of human-designed feedback templates. The blue-colored text represents the content specific to each instance of an error.

3.3 Baselines

We compare PROMST with six recent representative methods: Automatic Prompt Engineer (APE) (Zhou et al., 2023b), Automatic Prompt Optimization (APO) (Pryzant et al., 2023), PromptAgent (Wang et al., 2023), LLM-As-Optimizer (Yang et al., 2023), PromptBreeder (Fernando et al., 2023), and Evolutionary Prompt Optimizer (Guo et al., 2023b).

APE and LLM-as-optimizer propose to iteratively ask the PromptLLM to generate new, semantically similar prompts. Notably, errors appearing in previous prompts are not fed back into the search framework, hindering the correction of errors in new prompts. To handle this issue, APO queries an LLM to analyze errors for feedback into the prompt generator LLM. PromptAgent further augments the search efficiency by introducing a selection mechanism that considers exploration-exploitation trade-offs. PromptBreeder synthesizes meta-prompts to the prompt generator LLM by randomly concatenating thinking style prompts and mutation prompts, thus enhancing the prompt di-

versity. Evolutionary Prompt Optimizer applies a similar idea and augments prompt diversity by regularizing evolutionary algorithms such as genetic evolution and differential evolution. PROMST differs from the previous works in that the feedback is generated by human-design rules and a score prediction model is used to more efficiently sample prompt candidates.

4 Experiments

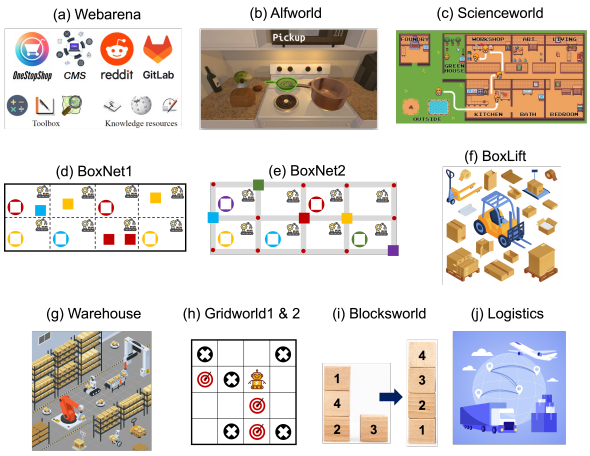

Figure 3: An illustration of the 11 environments used for multi-step task evaluation. See appendix D for more details.

We evaluate the performance of each method on 11 different environments for multi-step tasks. The tasks have been used in other work: Webarena from Zhou et al. (2023a); Alfworld from Shridhar et al. (2020); Scienceworld from Wang et al. (2022); BoxNet1, BoxNet2, BoxLift, and Warehouse from Chen et al. (2023b); Gridworld1 and Gridworld2 from Aghzal et al. (2023); Blocksworld and Logisitcs from Valmeekam et al. (2023). We implemented our own Gridworld1 and Gridworld2 since the original code is not yet available. We initialize each optimization process with human-designed seed candidate prompts; where possible, we used the provided publicly available prompts for each method. In all cases, we set the LLM sampling temperature to 0. For each method, we report the score of the best performing prompt on each task; in this case, the score is computed as:

$$S = \text{num}(\text{goal}_{success})/\text{num}(\text{goal}_{all}), \quad (4)$$

where the score S is the ratio of the number of successfully completed goals/sub-steps to the total number of goals/sub-steps. In Section 4.3, we also preliminarily test the impact of changing the scoring function S.

One interesting feature of these methods is that the LLM used to execute the task ('TaskLLM') and the LLM used to generate new candidate prompts ('PromptLLM') do not need to be the same model. We mainly test two combinations of models. The first uses GPT-3.5[1] as the TaskLLM and GPT-4[2] as the PromptLLM, and the second used GPT-4 for both the TaskLLM and PromptLLM. To verify the effectiveness of PROMST in varied models, we also evaluate it using Claude 3 Opus[3], Mixtral-8x7B[4], and Mixtral-Large[5] as both TaskLLM and PromptLLM. Mixtral-8x7B is an open model, while all the others are closed. We also explore whether the optimized prompts specialized for one type of LLM can generalize better performance to other types of LLMs.

We also include an evaluation of the learned score prediction model used in PROMST that tests how well it predicts the scores for different prompts and the relationship between the amount of training data and the error in prediction.

4.1 Hyperparameters

For a fair comparison, all methods start the optimization from initial human-designed prompts. The expansion number n is set to 20 in the first level and 8 for all additional levels. In each level, new prompt candidates are generated from the top $k = 5$ current prompts. Search terminates once the recent three levels do not have any score improvements. These settings ensure all the methods explore the same number of prompt candidates at the same level. In PROMST, the score model starts being trained and applied from level 3, i.e., $sd = 3$. We set $hyper_M = 0.8$ in Equation 3 to filter out prompts with low scores.

4.2 Environments

We test on 11 multi-step tasks requiring strong logical, geometrical, scientific, and commonsense reasoning capabilities. In each environment, the LLM agent should determine the next action in

[1] https://platform.openai.com/docs/models
[2] https://openai.com/research/gpt-4
[3] https://www.anthropic.com/news/claude-3-family
[4] https://mistral.ai/news/mixtral-of-experts/
[5] https://console.mistral.ai

Table 1: Scores for initial (human) and optimized prompts on various multi-step tasks for different methods. P.Agent, LLMOP, P.Breeder, and P.Evolution refer to PromptAgent, LLM-As-Optimizer, PromptBreeder, Evolutionary Prompt Optimizer, respectively. GPT-3.5-0613 for TaskLLM and GPT-4 for PromptLLM.

	GPT-3.5-0613-AS-TASKLLM, GPT-4-AS-PROMPTLLM							
TASK	HUMAN	APE	APO	P.AGENT	LLMOP	P.BREEDER	P.EVOLUTION	PROMST
WEBARENA	0.22	0.35	0.31	0.37	0.29	0.25	0.27	0.39
ALFWORLD	0.075	0.24	0.23	0.24	0.14	0.12	0.16	0.30
SCIENCEWORLD	0.18	0.19	0.19	0.23	0.19	0.20	0.22	0.21
BOXNET1	0.076	0.093	0.16	0.13	0.098	0.11	0.12	0.25
BOXNET2	0.044	0.075	0.16	0.17	0.086	0.090	0.075	0.22
BOXLIFT	0.31	0.69	0.70	0.74	0.55	0.58	0.62	0.90
WAREHOUSE	0.0	0.012	0.012	0.036	0.008	0.008	0.004	0.028
GRIDWORLD1	0.23	0.30	0.35	0.32	0.28	0.26	0.24	0.38
GRIDWORLD2	0.036	0.093	0.17	0.15	0.065	0.078	0.13	0.12
BLOCKSWORLD	0.19	0.25	0.42	0.48	0.29	0.22	0.27	0.60
LOGISTICS	0.083	0.083	0.12	0.12	0.083	0.083	0.12	0.18
AVERAGE	0.13	0.22	0.26	0.27	0.19	0.18	0.20	0.32

Table 2: Scores for initial (human) and optimized prompts on various multi-step tasks for different methods. GPT-4 for both TaskLLM and PromptLLM.

	GPT-4-AS-TASKLLM, GPT-4-AS-PROMPTLLM							
TASK	HUMAN	APE	APO	P.AGENT	LLMOP	P.BREEDER	P.EVOLUTION	PROMST
WEBARENA	0.57	0.59	0.64	0.60	0.58	0.58	0.59	0.62
ALFWORLD	0.45	0.49	0.50	0.53	0.50	0.47	0.49	0.57
SCIENCEWORLD	0.70	0.72	0.74	0.76	0.71	0.73	0.76	0.81
BOXNET1	0.65	0.72	0.72	0.77	0.74	0.67	0.70	0.79
BOXNET2	0.34	0.38	0.36	0.35	0.40	0.37	0.40	0.42
WAREHOUSE	0.16	0.18	0.27	0.34	0.30	0.25	0.22	0.51
GRIDWORLD1	0.73	0.78	0.82	0.89	0.83	0.76	0.80	0.86
GRIDWORLD2	0.26	0.50	0.44	0.41	0.41	0.31	0.29	0.60
BLOCKSWORLD	0.71	0.74	0.83	0.87	0.76	0.75	0.77	0.95
LOGISTICS	0.50	0.53	0.58	0.61	0.54	0.53	0.56	0.74
AVERAGE	0.51	0.56	0.59	0.61	0.58	0.54	0.56	0.69

the large discrete action space. Here we concisely introduce five representative environments. Please refer to Appendix D for a complete description of all environments.

Webarena (Figure 3a) is a real web environment containing four applications: online shopping, discussion forums, collaborative development, and business content management. **Alfworld** (Figure 3b) is the Household task that requires models to explore rooms and use commonsense reasoning. **Warehouse** (Figure 3g) requires the robots to move all boxes to the target region in fewest time steps, while making sure no collision happens in the limited space. **Gridworld1** (Figure 3h) consists of obstacles (black) and goals (red). The robot needs to visit all goals, and any attempt to move into obstacles or move out of the grid will result in failure. **Logistics** (Figure 3j) consists of packages, trucks, and airplanes. The goal is to transport packages within cities via trucks and between cities via airplanes to its goal location.

4.3 Results and Analysis

Overall better performance Table 1 and Table 2 show the main experimental results. Note that BoxLift task is not included in Table 2 since we find GPT-4 can already achieve a full score with the initial human prompt. Table 3 shows the experimental results on other three types of LLMs. Due to limited computational resources, we only select four representative tasks and two strongest baseline methods (APO, PromptAgent) when evaluating other LLMs. Table 5 and Table 6 in Appendix E test the performance of the optimized prompts trained from one LLM with other types of TaskLLMs.

The main takeaways are: 1) PROMST performs the best in most tasks. On average, PROMST outperforms strongest baseline PromptAgent with GPT-3.5-0613 (0.27 vs 0.32), GPT-4 (0.61 vs 0.69),

Table 3: Evaluation of different LLMs on a subset of the mutli-step tasks. The same LLM was used for the TaskLLM and PromptLLM in each case.

	CLAUDE-3-OPUS-20240229				OPEN-MIXTRAL-8X7B			
TASK	HUMAN	APO	P.AGENT	PROMST	HUMAN	APO	P.AGENT	PROMST
ALFWORLD	0.32	0.40	0.42	0.49	0.055	0.074	0.071	0.10
BOXNET2	0.42	0.47	0.44	0.53	0.078	0.21	0.20	0.24
WAREHOUSE	0.21	0.26	0.27	0.36	0.020	0.093	0.13	0.16
GRIDWORLD2	0.13	0.29	0.34	0.44	0.013	0.038	0.075	0.10
AVERAGE	0.27	0.36	0.37	0.46	0.041	0.10	0.12	0.15

MIXTRAL-LARGE-2402			
HUMAN	APO	P.AGENT	PROMST
0.28	0.33	0.33	0.45
0.26	0.31	0.36	0.30
0.16	0.23	0.28	0.31
0.12	0.32	0.25	0.28
0.21	0.30	0.30	0.34

Claude-3-opus (0.36 vs 0.46), Open-Mixtral-8x7B (0.12 vs 0.15), and Mixtral-large (0.30 vs 0.34). 2) When testing the best prompts trained from GPT-3.5-0613 and GPT-4 with a different TaskLLM, we find that they still outperform human prompts. 3) However, each LLM does best with the prompts optimized on it. For example, the best prompts acquired when using GPT-3.5-0613 as the TaskLLM do not further improve performance when applied to GPT-4, and vice versa. 4) PROMST performs well when the TaskLLM and PromptLLM are the same LLM, showing that it does not rely on a stronger PromptLLM to pass extra knowledge into prompts, which can be regarded as cheating.

Effects of Score Model To analyze the effects of the score model, we use BoxLift as a representative example, as shown in Figure 4. Figure 4a shows the distribution of prompt scores collected until level 4. We find most of the prompt scores are near 0.6, while many prompts achieve scores near 0. Figure 4b shows the training and testing errors of the score model versus different amounts of collected training data. The overfitting effect decreases with increasing data number. Figure 4c tests the fine-tuned score models on levels 5-8. We also evaluate the prompts that were filtered out by the score model and plot the predicted and actual scores. We find that nearly all chosen prompt candidates achieve scores much higher than 0, and the filtered prompts have reliably low scores. We also ablate PROMST without the score model, as shown in Figure 4d. The results show that both

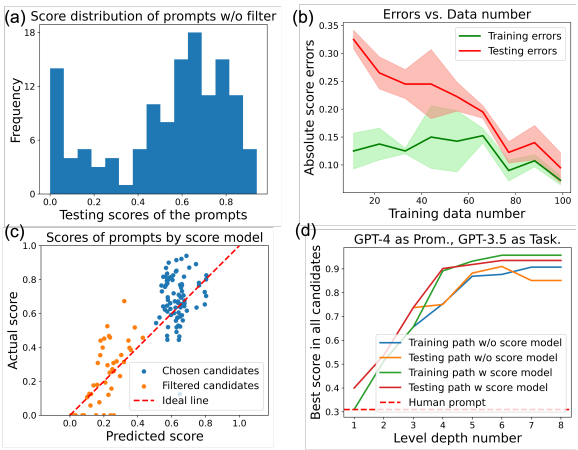

Figure 4: Several results inspecting the learned score prediction model. (a) the distribution of prompt scores when the score prediction model is not applied. (b) the prediction error of the model on the training data and heldout test data as the amount of training data increases. (c) a plot of the predicted score vs the actual score for various prompts; blue are the prompts that were chosen as parents for new candidates. (d) the trend of the best performing prompt during optimization for increasing iterations both with and without using the learned score prediction model.

the training and testing paths converge faster and achieve better scores using the score model. The ablation experiments in other environments also have the same trend (shown in Appendix F). Overall, we find the score prediction models improves the efficiency and effectiveness of prompt search.

Methods of Score Model Here we do another ablation study on the method for acquiring score prediction models. Instead of fine-tuning a pre-trained Longformer-base model, another way is few-shot learning via GPT-4. Figure 5a compares these two methods under varied training/example data number. GPT-4 is given randomly selected prompt-score pairs as examples during the study.

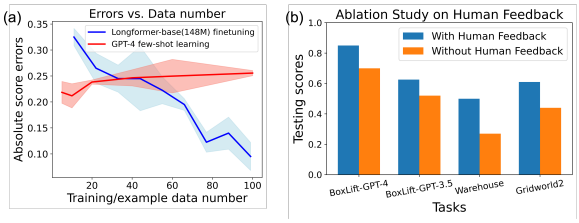

Figure 5: (a) comparison of score prediction errors for few-shot GPT-4 vs finetuning Longformer for increasing amount of few-shot examples or training data, respectively. (b) An ablation study of the impact of the human-designed feedback rules on task performance for four multi-step tasks.

We find that the performance of GPT-4 few-shot learning cannot improve with the increasing number of examples. The fine-tuning method surpasses GPT-4 few-shot learning once the data number increases over 40.

Ablation on Human Feedback We compare the method with/without human feedback, both without the learned score model. As seen in Figure 5b, human feedback contributes to much higher scores across four tasks.

Preference Alignment via Score Function The choice of score function impacts prompt optimization. The initial score functions in Equation 4 are simple and intuitive, only caring about the number of goals/sub-steps accomplished. However, humans may have different preferences for the same task. For instance, a user may also care about efficiency (the number of action steps taken) or safety (collision avoidance). We observe a general trend that the step number increases as the prompt score increases in all the three shown tasks (see Appendix H). However, in BoxNet2 (Figure 11) the collision error number gradually increases with the increasing prompt scores. These two general trends are not aligned with the user preference.

Then how to design the score function to balance user preferences remains an issue. In Appendix I, we tried the two forms of modified scores:

$$S_M = S_O - \text{ratio} * \text{factor_value} \quad (5)$$

$$S_M = S_O/(1 + \text{ratio} * \text{factor_value}), \quad (6)$$

where S_M and S_O are the modified score and the original score (defined in Equation 4), respectively. The factor_value is a factor that the user cares about, e.g., step number or collision error number. We find that the general S_M vs. S_O trend can be tuned quite disparately by adjusting the hyper-parameter ratio (see Figure 12 and Figure 13).

We choose two modified score functions that trend similarly to the original score function. Then we optimize the prompts with PROMST but using modified score function. To save computing resources, we initialize the prompt optimization with the best prompts found with the original score function. Figure 6 shows the optimization results. Compared to the original prompts acquired with the original score function (red), the newly discovered prompts (green) generally have higher modified scores, though the values of original scores slightly decrease. This suggests that we can align with human preferences by changing the form of the score functions, which can be captured and revealed by the selection framework in PROMST.

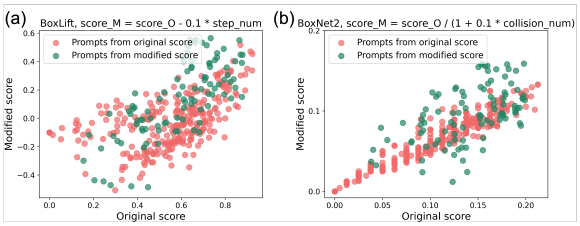

Figure 6: Human preference alignment via tuning score functions. The green dots are the new prompts further optimized over new score rules.

5 Conclusion

In this work we introduce an automatic prompt optimization framework for complex, multi-step agent tasks: PROMST. To handle the issues of task complexity, judging long-horizon correctness of individual actions, high prompt exploration cost, and human preference alignment, we propose the integration of human feedback, a learned score prediction model, and the modification of task score functions. Our approach generally outperforms six representative baselines on 11 different task environments over all the five LLMs. We show that learning a score prediction model improves the overall performance. Finally, we argue that modifying score functions can help align the optimized prompts to user preferences. All the codes and discovered prompts are released, which may serve as a benchmark for future researches in automatic prompt optimization in multi-step agent tasks.

6 Limitations

The limitations and potential societal risks of this work are as follows:

Huge resource consumption of API calls Automatic prompt optimization requires significant computing resources and LLM API queries due to its search-based nature, which is a common issue in this research track. Though the introduction of score model makes the searching more efficient, the around 100 prompt candidate exploration is still a large burden.

Score model increases computing demands of local devices The fine-tuned score prediction model trades-off the number of API queries for on-device computation by selecting good candidate prompts. Still, the training of extra score models increases the computing demands on local devices.

Fine-tuning score model requires enough data points The fine-tuning process of score models typically requires around 100 prompt-score pairs, which is suitable for black box prompt searching since over 100 data points are truly needed for satisfying performance. However, the score model may not be suitable if in the future a more efficient searching method appears so that data points are not such much.

Human-designed feedback may increase user burden Though the human-designed feedback rules are easy to acquire and very flexible, this may increase the burden of users since they need to detect and classify errors for each task. Meanwhile, the human-designed feedback may lose the usefulness if the feedback is not reasonable and proper. The user study is supposed to carry out in the future research.

Lack of explanability for better prompts Though the discovered prompts achieve much better performance, the reason why they are better are unclear. In Appendix G, we plot prompt score vs. token length and perplexity, which implies some clues that longer prompts may be better. Meanwhile, when viewing through the discoverd best prompts in Appendix J, we find some clues about better component emergence, i.e., the best prompts tend to list all the careful points one by one clearly. All these kinds of characteristics of better prompts need further extensive research.

Potential risks The potential risks of this work may be the general risks of black prompt tuning methods, where more powerful prompt learning algorithms enable more powerful prompt attack approaches. For instance, the jailbreaking task can also utilize this framework. Meanwhile, LLM based multi-step agents can be employed for harmful tasks, such as financial fraud. Then the prompt optimization methods may be used to make the harmful agent even more powerful. For these reasons, deployment of this technology will have to be carefully considered and combined with research in other areas on LLM safety to mitigate potential societal risks.

References

Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. 2023. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models. *arXiv preprint arXiv:2311.18232*.

Mohamed Aghzal, Erion Plaku, and Ziyu Yao. 2023. Can large language models be good path planners? a benchmark and investigation on spatial-temporal reasoning. *arXiv preprint arXiv:2310.03249*.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *Preprint*, arXiv:2004.05150.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2023a. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. *arXiv preprint arXiv:2306.06531*.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2023b. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? *arXiv preprint arXiv:2309.15943*.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang.

2023. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*.

Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, LIN Yong, Xiao Zhou, and Tong Zhang. 2022. Black-box prompt learning for pre-trained language models. *Transactions on Machine Learning Research*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. 2023. Foundation models in robotics: Applications, challenges, and the future. *arXiv preprint arXiv:2312.07843*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023a. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2023b. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *Preprint*, arXiv:2309.08532.

Or Honovich, Uri Shaham, Samuel R Bowman, and Omer Levy. 2022. Instruction induction: From few examples to natural language task descriptions. *arXiv preprint arXiv:2205.10782*.

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

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. Text2motion: From natural language instructions to feasible plans. *arXiv preprint arXiv:2303.12153*.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *arXiv preprint arXiv:2401.13178*.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Rui Pan, Shuo Xing, Shizhe Diao, Xiang Liu, Kashun Shum, Jipeng Zhang, and Tong Zhang. 2023. Plum: Prompt learning using metaheuristic. *Preprint*, arXiv:2311.08364.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 3827–3846.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

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*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew

Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. 2023. Generalized planning in pddl domains with pretrained large language models. *arXiv preprint arXiv:2305.11014*.

Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. 2023. Errors are useful prompts: Instruction guided task programming with verifier-assisted iterative prompting. *arXiv preprint arXiv:2303.14100*.

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*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? *arXiv preprint arXiv:2203.07540*.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2023. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

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

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2022. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023a. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2023b. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv preprint arXiv:2307.02477*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Kevin Yang, Yuandong Tian, Nanyun Peng, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4393–4479.

Qinyuan Ye, Maxamed Axmed, Reid Pryzant, and Fereshte Khani. 2023. Prompt engineering a prompt engineer. *arXiv preprint arXiv:2311.05661*.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. 2023. TEMPERA: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023a. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023b. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Types of human feedback for each task

This table displays the types of errors and corresponding human feedback for each testing task. The specific contents of each feedback is shown in Figure 2.

Webarena	Syntactic error; Stuck in the loop; Failure over query time limit; Invalid action
Alfworld	Syntactic error; Stuck in the loop; Failure over query time limit; Invalid action
Scienceworld	Syntactic error; Stuck in the loop; Failure over query time limit; Invalid action
BoxNet1	Syntactic error; Stuck in the loop; Failure over query time limit
BoxNet2	Syntactic error; Stuck in the loop; Failure over query time limit; Collision
BoxLift	Syntactic error; Stuck in the loop; Failure over query time limit
Warehouse	Syntactic error; Stuck in the loop; Failure over query time limit; Collision
Gridworld1	Syntactic error; Stuck in the loop; Failure over query time limit; Collision; Move out of the grid;
Gridworld2	Syntactic error; Stuck in the loop; Failure over query time limit; Collision; Move out of the grid; Wrong picking up order;
Blocksworld	Syntactic error; Stuck in the loop; Failure over query time limit; Invalid action
Logistics	Syntactic error; Stuck in the loop; Failure over query time limit; Invalid action; Wrong object action

Table 4: Types of human feedback for each task

B Algorithms

Algorithm 1 PRompt Optimization in Multi-Step Tasks (PROMST)

Require: p_0: initial prompt, k: beam width, d: search level depth, n: expansion number, sd: depth when score model training starts

```
B_0 ← {p_0}; D ← {}
D[p_0] ← TaskLLM(p_0)                                   ▷ [score, feed, AnP]
for i ← 1 to d − 1 do
    for all p ∈ B_i do
        P_new ← NewPrompt(p, D, i, sd, n)
        for all p_new ∈ P_new do
            D[p_new] ← TaskLLM(p_new)   ▷ Dictionary to record all prompt trials and information
        end for
    end for
    B_{i+1} ← Top_k(D)                                  ▷ Select top k prompts by scores
    Output B_{i+1}
end for
p̂ ← argmax_{p∈B_d} score(p)                             ▷ The best prompt
Output p̂
return p̂
```

Algorithm 2 NewPrompt() - line 5 of Algorithm 1

Require: p: input prompt, D: all prompts, i: current depth level, sd: depth level number when score model training starts, n: expansion number

1: $[feed, AnP] \leftarrow D[p]$ ▷ feedback and ancestor prompts (prompt trajectory leading to the current one)
2: **if** $i \geq sd$ **then**
3: $M_k \leftarrow finetune(D), k = 1, ..., 5$ ▷ Score model, input: prompt, output: score
4: $P_{new} \leftarrow \{\}$
5: $iter \leftarrow 0$
6: **while** len(P_{new}) $< n$ and $iter < 3n$ **do**
7: $iter \mathrel{+}= 1$
8: $p' = PromptLLM(p, feed, AnP, 1)$
9: **if** $E[M_k(p')] + Var[M_k(p')] + E[error_k] \geq hyper_M * \max(D.score())$ **then**
10: P_{new}.add(p')
11: **end if**
12: **end while**
13: **return** P_{new}
14: **else**
15: $\{p'_1, ..., p'_n\} = PromptLLM(p, feed, AnP, n)$
16: **return** $\{p'_1, ..., p'_m\}$
17: **end if**

Algorithm 3 PromptLLM() - line 6 and 13 of Algorithm 2

Require: p: input prompt, $feed$: list of feedback from humans, AnP: list of prompt trajectory leading to the current one, n: expansion number

1: $P_{new} \leftarrow \{\}$
2: **for** $i \leftarrow 1$ **to** n **do**
3: $feed2 \leftarrow random_select(feed, \min(10, len(feed))$
4: $\{type1, type2, ...\} \leftarrow classify_concat(feed2)$
5: $feed2LLM ='$
6: **for all** $feed_type \in \{type1, type2, ...\}$ **do**
7: $feed2LLM \mathrel{+}= SumLLM(p, feed_type)$ ▷ Summarize each type of feedback
8: **end for**
9: P_{new}.add($GenLLM(p, feed2LLM, AnP)$)
10: **end for**
11: **return** P_{new}

C Meta-prompts of $SumLLM$ and $GenLLM$

Meta-prompt of $SumLLM$
Imagine you are a prompt optimizer based on the human feedback and task execution feedback. I'm writing prompts for a language model designed for a task.

My current prompt of task specification is: {current_prompt}, but this prompt gets the following examples wrong: {feedback_type}

Based on all these errors and feedback, summarize the reasons and list all the aspects that can improve the prompt. Keep your summary concise and clear.

Meta-prompt of $GenLLM$
Imagine you are a prompt optimizer based on the feedback from the human and task execution feedback. Here is the prompt of task description: {prompt_task_explain}

However, the response generated from the initial task description prompt owns some errors. Here are the error feedback from humans: {error_feedback}

There is a list of former prompts including the current prompt, and each prompt is modified from its former prompts:{trajectory_prompts}

Based on the feedback, think about why the task planning LLM agent makes the error and try to optimize the prompt of task description to avoid this error.

The new prompts should follow these guidelines: 1. The new prompts should solve the current prompt's problems. 2. The new prompts should consider the list of prompts and evolve based on the current prompt.

Output the optimized prompt of task description without other texts:

D Description of environments for multi-step tasks

Here we describe the 11 environments for multi-step tasks on which the various methods were tested. They requires strong logical, geometrical, scientific, and commonsense reasoning capabilities.

Webarena Webarena (Figure 3a) is a real web environment containing four applications: online shopping, discussion forums, collaborative development, and business content management. It supports 11 different web browsing actions. The observation space consists of structured web content. WebArena offers multi-round and continuous web browsing interaction simulation.

Alfworld Alfworld (Figure 3b) are Household tasks that require models to explore rooms and use commonsense reasoning to perform tasks, such as "put a pencil on the desk". The execution scores are calculated by pre-defined subgoals based on necessary observations to finish a task and the success flag provided by environments.

Scienceworld Scienceworld (Figure 3c) is a complex interactive text environment that poses a significant challenge to agents' scientific commonsense. This environment requires agents to navigate through eight distinct functional rooms (e.g., workshop, kitchen) and utilize the tools to complete tasks such as "measure the melting point of the orange juice".

BoxNet1 BoxNet1 (Figure 3d) consists of robot arms, colored boxes (squares), and colored goal locations (circles). Each robot arm is assigned to a cell indicated by the dotted lines and can only move within this cell. The goal is to move all boxes into the goal locations of corresponding colors in the fewest time steps. Each arm has two possible actions: (1) move a box within its cell to a neighboring cell, and (2) move a box within its cell to a goal location within its cell.

BoxNet2 BoxNet2 (Figure 3e) is similar to BoxNet1 but has an additional constraint. In BoxNet2, boxes can only be moved between cells by being placed at the corners of cells (indicated by the small red circles), and each cell corner can only hold one box at a time. Each arm has two possible actions: (1) move a box from a corner to a different corner of its cell, and (2) move a box from a corner to a goal location within its cell.

BoxLift BoxLift (Figure 3f) consists of robots of different types and boxes of different sizes and weights. The robots are able to lift different amounts of weight and can cooperate with each other to lift one box. A box will be lifted only if the total lifting capability of robots is greater than the box's weight. The goal is to lift all boxes in fewest time steps. Further, the LLM agent can only observe the size of each box, not its actual weight. The weight of a box is roughly proportional to its size (with some randomness), so the LLM agent should benefit from incorporating prior state/action feedback when planning.

Warehouse Warehouse (Figure 3g) consists of robots that need to move all boxes to a target delivery region in the fewest time steps. The free space for the robots to move is discretized into cells, and a robot can only move to an adjacent cell in a single time step. Each cell can only contain one robot at each timestep. A robot is able to pick up a box if it is in the cell adjacent to that box. Each robot has five possible actions: (1) & (2) move left or right if the adjacent cell exists, (3) pick up an adjacent box, (4) place the box to the target delivery region, (5) move from target delivery region to any adjacent cell of free space.

Gridworld1 Gridworld1 (Figure 3h) consists of obstacles (black) and goals (red). The robot needs to visit all goals, and any attempt to move into obstacles or move out of the grid will result in failure. The robot has five possible actions: (1) move up, (2) move down, (3) move left, (4) move right, (5) visit goal.

Gridworld2 Gridworld2 is similar to Gridworld1, but the goals must be visited in a particular order. The robot action are the same as in Gridworld1, but 'visit goal' can be performed only when the

corresponding goal is in the correct order.

Blocksworld In Blocksworld (Figure 3i), the goal is to stack a set of blocks (brown) according to a specific order. A robot can pick up, unstack, or stack a block only when the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up. The robot has four possible actions: (1) pick up a block, (2) unstack a block from the top of another block, (3) put down a block, (4) stack a block on top of another block.

Logistics Logistics (Figure 3j) consists of objects, locations, and cities. The objects can be packages, trucks, or airplanes. The locations can be generic locations or airports, and each location is associated with a single city. Trucks can travel to different locations within a city but not to a different city; airplanes can travel to any airports, including those in other cities. The goal is to transport packages to their goal locations via the trucks (such as for intra-city travel) and the airplanes (such as for inter-city travel). The available actions are: (1) load a package into a truck, (2) load a package into an airplane, (3) unload a package from a truck, (4) unload a package from an airplane, (5) drive a truck from one location to another location within a city, (6) fly an airplane from one airport to another airport.

Table 5: Scores for initial and optimized prompts using different types of LLMs as TaskLLM. The optimized prompts are the best discovered prompts by PROMST for GPT-3.5-0613. The optimized prompts are further tested with GPT-3.5-0301 and GPT-4 to study whether they can keep better performances than the initial prompts.

	GPT-3.5-0613-AS-TASKLLM, GPT-4-AS-PROMPTLLM					
	GPT-3.5-0613		**GPT-3.5-0301**		**GPT-4**	
TASK	HUMAN	PROMST	HUMAN	PROMST	HUMAN	PROMST
WEBARENA	0.22	0.35	0.29	0.34	0.57	0.54
ALFWORLD	0.075	0.30	0.17	0.21	0.45	0.49
SCIENCEWORLD	0.18	0.21	0.16	0.13	0.70	0.68
BOXNET1	0.076	0.25	0.28	0.38	0.65	0.67
BOXNET2	0.044	0.22	0.088	0.28	0.34	0.31
BOXLIFT	0.31	0.90	0.69	0.91	1.0	1.0
WAREHOUSE	0.0	0.028	0.0	0.040	0.16	0.19
GRIDWORLD1	0.23	0.38	0.25	0.32	0.73	0.85
GRIDWORLD2	0.036	0.12	0.021	0.13	0.26	0.29
BLOCKSWORLD	0.19	0.60	0.33	0.24	0.71	0.62
LOGISTICS	0.083	0.18	0.12	0.083	0.50	0.63
AVERAGE	0.13	0.32	0.22	0.28	0.55	0.57

Table 6: Scores for initial and optimized prompts using different types of LLMs as TaskLLM. The optimized prompts are the best discovered prompts by PROMST for GPT-4. The optimized prompts are further tested with GPT-3.5-0301 and GPT-3.5-0613 to study whether they can keep better performances than the initial prompts.

	GPT-4-AS-TASKLLM, GPT-4-AS-PROMPTLLM					
	GPT-3.5-0613		**GPT-3.5-0301**		**GPT-4**	
TASK	HUMAN	PROMST	HUMAN	PROMST	HUMAN	PROMST
WEBARENA	0.22	0.18	0.29	0.32	0.57	0.62
ALFWORLD	0.075	0.092	0.17	0.19	0.45	0.57
SCIENCEWORLD	0.18	0.15	0.16	0.20	0.70	0.81
BOXNET1	0.076	0.12	0.28	0.32	0.65	0.79
BOXNET2	0.044	0.14	0.088	0.17	0.34	0.42
WAREHOUSE	0.0	0.019	0.0	0.025	0.16	0.51
GRIDWORLD1	0.23	0.26	0.25	0.29	0.73	0.86
GRIDWORLD2	0.036	0.057	0.021	0.042	0.26	0.60
BLOCKSWORLD	0.19	0.21	0.33	0.24	0.71	0.95
LOGISTICS	0.083	0.083	0.12	0.18	0.50	0.74
AVERAGE	0.11	0.13	0.17	0.20	0.56	0.69

E Generalization to different models for optimized prompts

F Extra ablation experiments of score models

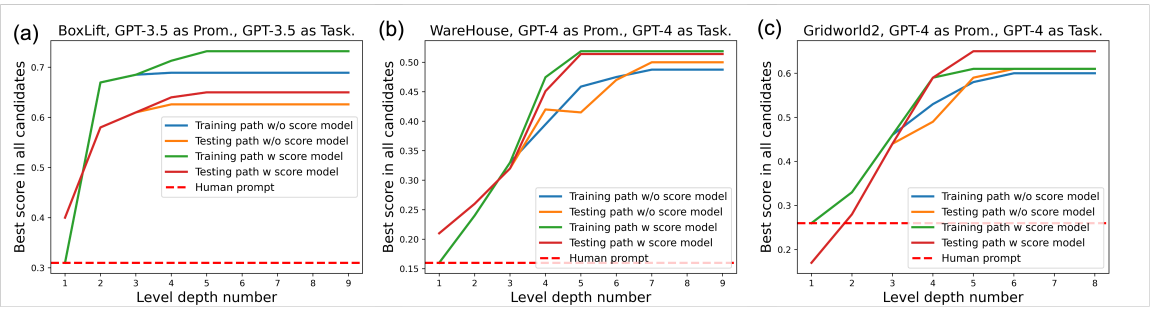

Figure 7: Ablation study of applying score models for prompt selection. The optimization trace with score model finds better prompts with less iteration steps, across all the three tasks BoxLift, WareHouse, and GridWorld2.

G Prompt score vs. token length and perplexity

We are interested in whether there are features of the prompts that correlate with high scores. We found that there is a rough trend across different tasks that longer prompts corresponded with higher scores. We also investigated prompt perplexity (using GPT-2 to get prompt token log probabilities) but found no clear correlation. All the initial and discovered best prompts are listed in Appendix J.

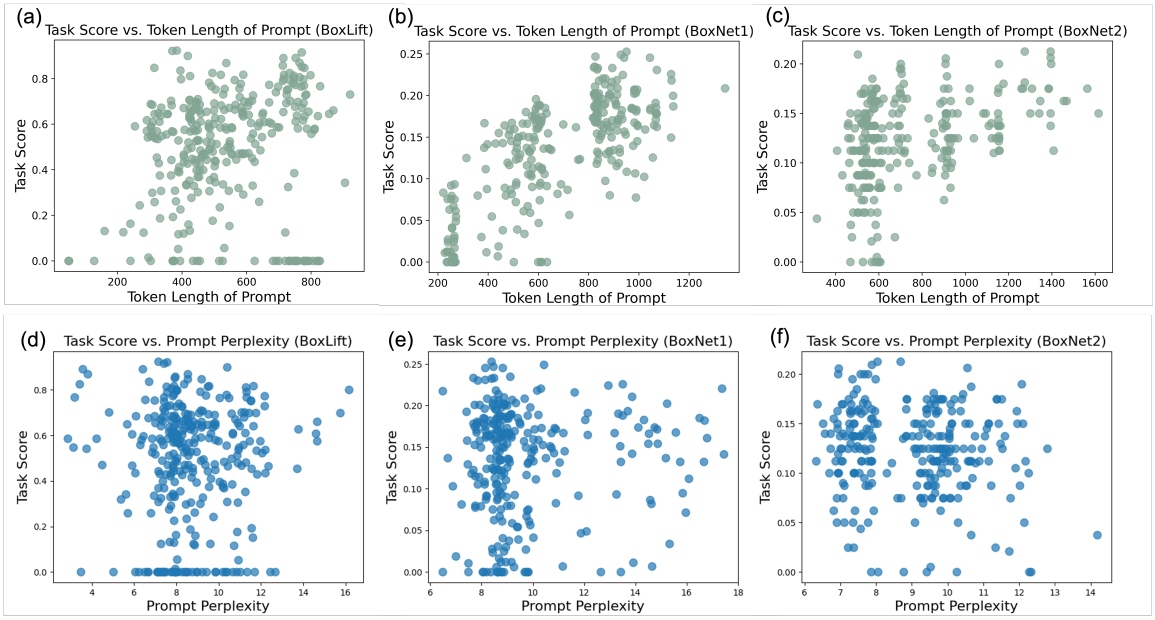

Figure 8: Score vs. prompt token length and score vs. prompt perplexity for all the explored prompts in each task.

H Component changes in each environment

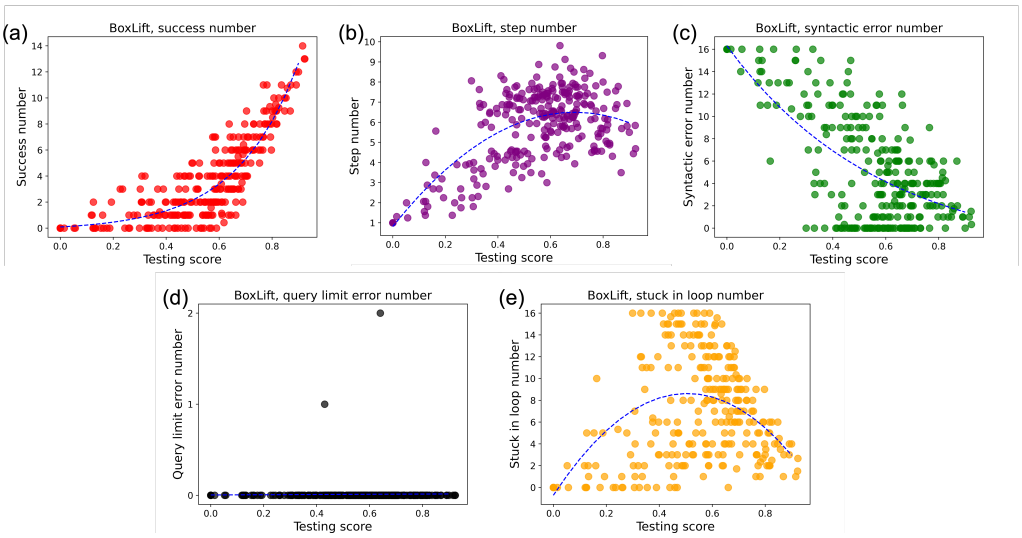

Figure 9: Component change of BoxLift.

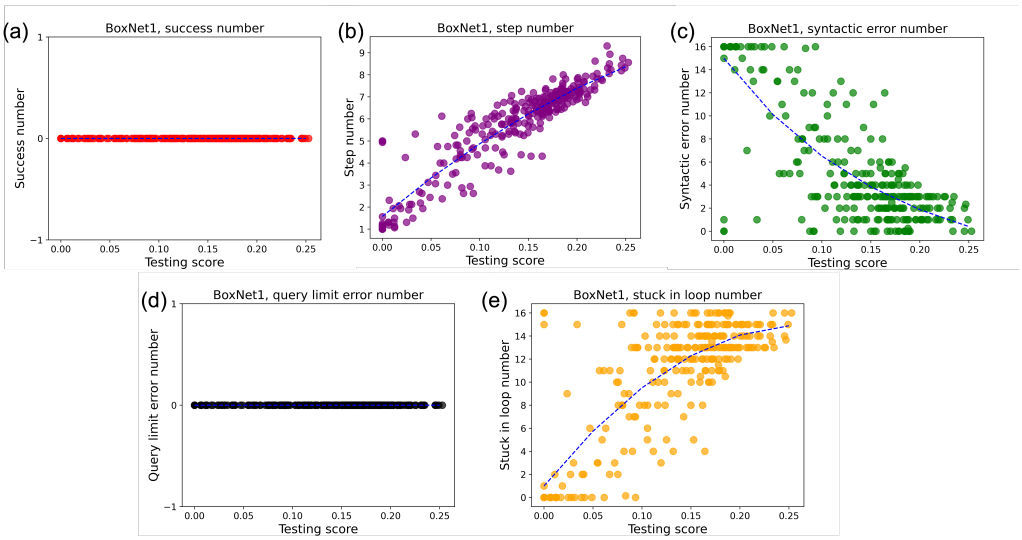

Figure 10: Component change of BoxNet1.

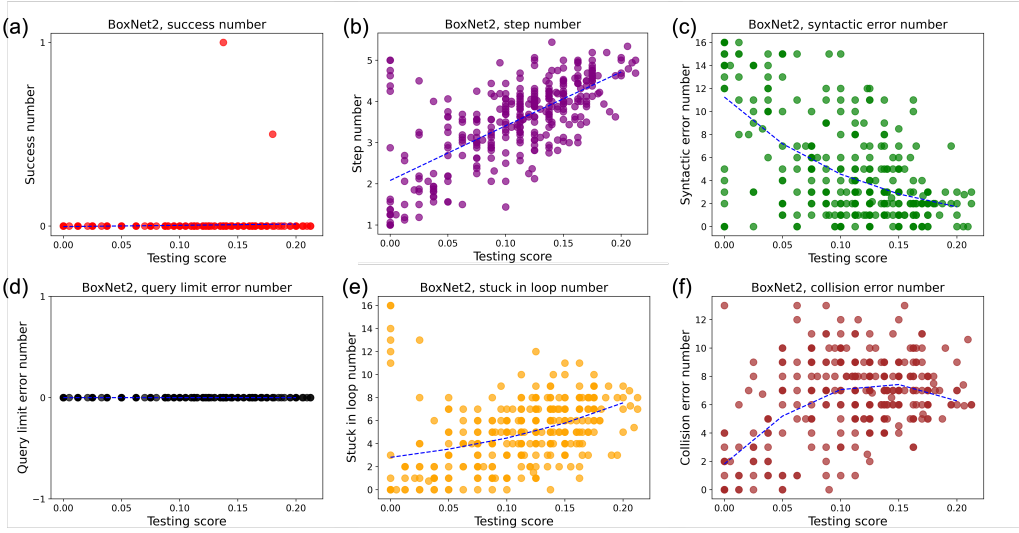

Figure 11: Component change of BoxNet2.

I The influence of score functions

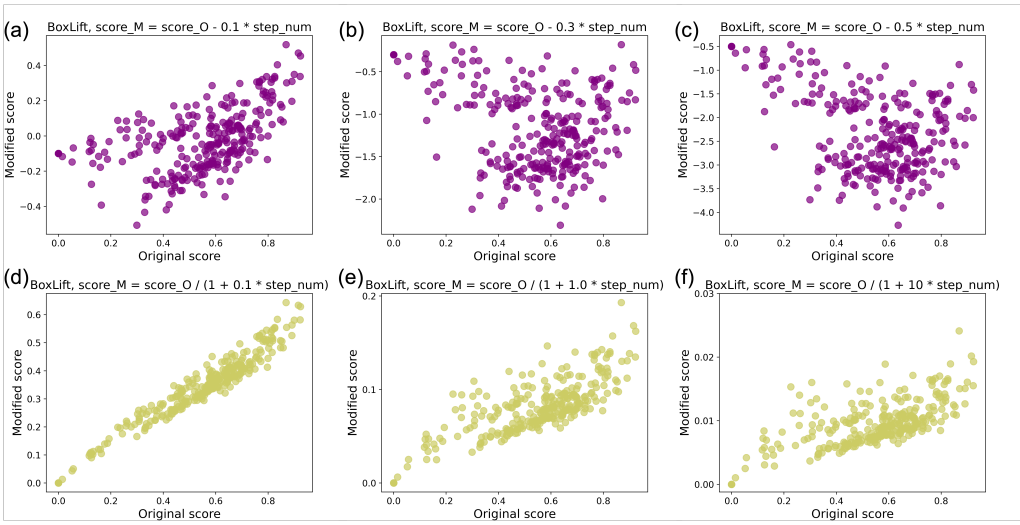

Figure 12: Modified score of BoxLift.

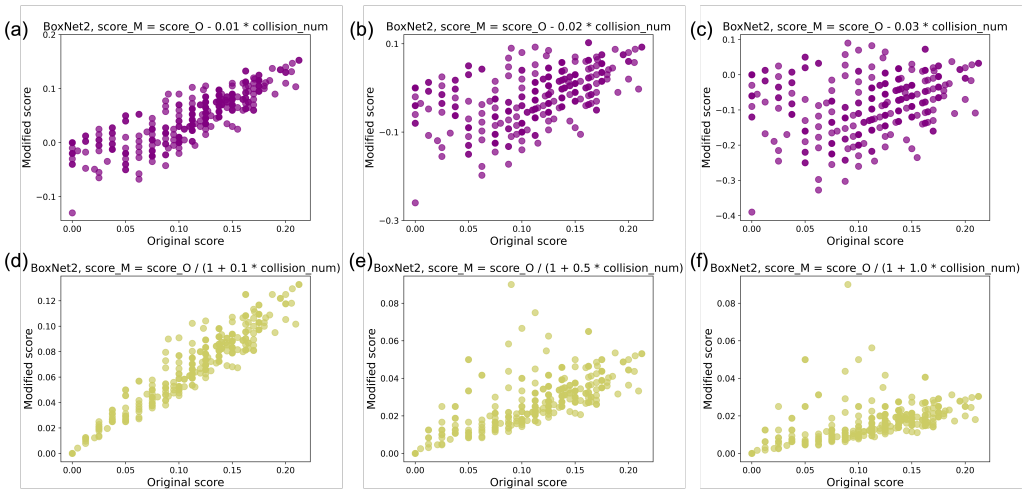

Figure 13: Modified score of BoxNet2.

J Human prompts and discovered best prompts for GPT-3.5-0613 and GPT-4 in all the 11 multi-step tasks

Our work can serve as a benchmark for prompt optimization, particularly on multi-step agent tasks. Hence, we list all the initial human prompts and discovered best prompts for GPT-3.5-0613 and GPT-4 models across the 11 tasks. Note that we do not list the best prompt of GPT-4 in BoxLift since the optimized prompt can easily achieve the full score 1.0. We also do not list the best prompt of GPT-3.5-0613 in WareHouse since all the discovered prompts achieve scores near 0.0.

Webarena Human prompt
Score = 0.22 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.57 (GPT-4 as the testing LLM)

Here's the information you'll have:
The user's objective: This is the task you're trying to complete.
The current web page's accessibility tree: This is a simplified representation of the windowed webpage, providing key information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.
The useful websites and corresponding URL you can navigate:

- `'reddit': http://reddit.com`
- `'online shop': http://onestopmarket.com`
- `'e-commerce platform': http://luma.com/admin`
- `'gitlab': http://gitlab.com`
- `'wikipedia': http://wikipedia.org`
- `'map': http://openstreetmap.org`

Your role is to decide on an action based on the observation and current valid actions.
Ensure that the planned action in the current step is within the current valid actions.
The actions you can perform fall into several categories:
Page Operation Actions:

- `'click [id]'`: This action clicks on an element with a specific id on the webpage.
- `'type [id] [content] [press_enter_after=0|1]'`: Use this to type the content into the field with id. By default, the 'Enter' key is pressed after typing unless press_enter_after is set to 0.
- `'hover [id]'`: Hover over an element with id.
- `'press [key_comb]'`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `'scroll [direction=down|up]'`: Scroll the page up or down.

Tab Management Actions:

- `'new_tab'`: Open a new, empty browser tab.
- `'tab_focus [tab_index]'`: Switch the browser's focus to a specific tab using its index.
- `'close_tab'`: Close the currently active tab.

URL Navigation Actions:

- `'goto [url]'`: Navigate to a specific URL.
- `'go_back'`: Navigate to the previously viewed page.
- `'go_forward'`: Navigate to the next page (if a previous 'go_back' action was performed).

Completion Action: `'stop [answer]'`: Apply this action when you believe the task is complete. If it is an operation-type task, use `'stop [Done]'` when finished. If the objective is to give a text-based answer, provide the answer in the bracket.
To be successful, it is very important to follow the following rules:

1. You should only issue an action that is valid given the current observation.
2. You should only issue one action at a time.
3. Generate the action in the correct format and always put the action inside a pair of @. Such as, `@click [1234]@`.
4. Complete the task by interacting with the starting page, and avoid using 'goto' actions casually.
5. Reasonable inputs will return accurate observations, so do not repeat the same action when unnecessary.

Webarena Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.39 (GPT-3.5-turbo-16k-0613 as the testing LLM)

Here's the information you'll have:

- The user's objective: This is the task you're trying to complete.
- The current web page's accessibility tree: This is a simplified representation of the windowed webpage, providing key information.

- The current web page's URL: This is the page you're currently navigating.
- The open tabs: These are the tabs you have open.

The useful websites and corresponding URLs you can navigate:

- 'reddit': 'http://reddit.com'
- 'online shop': 'http://onestopmarket.com'
- 'e-commerce platform': 'http://luma.com/admin'
- 'gitlab': 'http://gitlab.com'
- 'wikipedia': 'http://wikipedia.org'
- 'map': 'http://openstreetmap.org'

Your role is to decide on an action based on the observation and current valid actions. Ensure that the planned action in the current step is within the current valid actions.
The actions you can perform fall into several categories:
Page Operation Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the 'Enter' key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.

Tab Management Actions:

- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.

URL Navigation Actions:

- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).

Completion Action:

- `stop [answer]`: Apply this action when you believe the task is complete. If it is an operation-type task, use `stop [Done]` when finished. If the objective is to give a text-based answer, provide the answer in the bracket.

To be successful, it is very important to follow the following rules:

913

1. You should only issue an action that is valid given the current observation.
2. You should only issue one action at a time.
3. Generate the action in the correct format and always put the action inside a pair of `@`. Such as, `@click [1234]@`.
4. Complete the task by interacting with the starting page, and avoid using '`goto`' actions casually.
5. Reasonable inputs will return accurate observations, so do not repeat the same action when unnecessary.
6. If the task involves searching or filtering content, use the website's specific features designed for that purpose, such as search bars, filters, or category selectors.
7. Before issuing a `stop [Done]` action, ensure that the task's completion criteria have been met by reviewing the observations and confirming that the desired outcome is achieved.
8. If the initial action does not yield the expected result, reassess the situation and consider alternative valid actions that could lead to task completion.
9. In case of an unsuccessful outcome, explore different valid actions and utilize the website's UI elements to navigate and achieve the task objective.
10. Implement a feedback loop by reassessing and adjusting actions based on the results of previous actions and environment feedback.

Webarena Best prompt for GPT-4
Score = 0.62 (GPT-4 as the testing LLM)

Here's the information you'll have:

- **The user's objective:** This is the task you're trying to complete.
- **The current web page's accessibility tree:** This is a simplified representation of the windowed webpage, providing key information.
- **The current web page's URL:** This is the page you're currently navigating.
- **The open tabs:** These are the tabs you have open.

The useful websites and corresponding URL you can navigate:

- 'reddit': `http://reddit.com`
- 'online shop': `http://onestopmarket.com`
- 'e-commerce platform': `http://luma.com/admin`
- 'gitlab': `http://gitlab.com`
- 'wikipedia': `http://wikipedia.org`
- 'map': `http://openstreetmap.org`

Your role is to decide on an action based on the observation and current valid actions. Ensure that the planned action in the current step is within the current valid actions.
The actions you can perform fall into several categories:
Page Operation Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.

- `type [id] [content] [press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the 'Enter' key is pressed after typing unless press_enter_after is set to 0. Ensure the content syntax is correct for the context (e.g., search queries should use the proper format for the website).
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.

Tab Management Actions:

- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.

URL Navigation Actions:

- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).

Completion Action:

- `stop [answer]`: Apply this action when you believe the task is complete. If it is an operation-type task, use `stop [Done]` when finished. If the objective is to give a text-based answer, provide the answer in the bracket.

To be successful, it is very important to follow the following rules:

1. You should only issue an action that is valid given the current observation.
2. You should only issue one action at a time.
3. Generate the action in the correct format and always put the action inside a pair of `@`. Such as, `@click [1234]@`.
4. Complete the task by interacting with the starting page, and avoid using 'goto' actions casually.
5. Reasonable inputs will return accurate observations, so do not repeat the same action when unnecessary.
6. If an action does not produce the expected result, do not repeat the action. Instead, analyze the feedback and adjust the strategy accordingly.
7. Use conditional logic to adapt to feedback from the environment. If an action fails, consider alternative approaches or refine the action to achieve the desired outcome.
8. Manage time efficiently by optimizing the sequence of actions to achieve the goal quickly.
9. Include error handling to address unexpected outcomes or failures in task execution.
10. Provide clear instructions on how to refine search queries or alternative methods to locate the desired information.

Alfworld Human prompt
Score = 0.075 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.45 (GPT-4 as the testing LLM)

Your task is to interact with a virtual household simulator to accomplish a specific task. With each interaction, you will receive an observation and current valid actions. Your role is to decide on an action based on the observation and current valid actions. Please ensure that any

objects (`{obj}`) and receptacles (`{recep}`) you mention in your response are present in the observation provided. Ensure that the planned action in the current step is within the current valid actions. Example objects are like a cellphone 3, a newspaper 2, a statue 1, and a television 1. Example receptacles are like a coffeetable 1, a diningtable 1, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a dresser 1, a garbagecan 1, a sidetable 2, a sidetable 1, and a sofa 1. Example actions are like `[go to dresser 1, take statue 1 from dresser 1, heat apple 1 with microwave 1, open cabinet 2]` Do not repeat the actions all the time! Learn from the previous action/observation history.
Here are the available actions you can take:

- `take {obj} from {recep}`
- `put {obj} in/on {recep}`
- `open {recep}`
- `close {recep}`
- `toggle {obj}/{recep}`
- `clean {obj} using {recep}`
- `cool {obj} using {recep}`
- `heat {obj} using {recep}`
- `inventory`
- `examine {recep}/{obj}`
- `go to {recep}`

Alfworld Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.30 (GPT-3.5-turbo-16k-0613 as the testing LLM)

Your task is to interact with a virtual household simulator to achieve a clearly defined goal. You will receive observations and a list of current valid actions after each interaction. Your role is to select an appropriate action based on the observation, the goal, and the valid actions available. Ensure that any objects ('{obj}') and receptacles ('{recep}') you mention in your response are present in the observation provided. Your planned action must be one of the current valid actions.
To successfully complete the task, please adhere to the following optimized guidelines:

1. **Understand the Goal**: Always keep the goal at the forefront of your decision-making process. Each action you select should be a strategic step towards accomplishing this goal.
2. **Use Observations**: Analyze the observations to gain a comprehensive understanding of the environment's current state, including the location and status of objects and receptacles.
3. **Valid Action Selection**: Strictly choose your actions from the provided list of current valid actions. Do not attempt any actions that are not listed as valid for the current situation.
4. **State Tracking and Changes**: Keep a mental model of the environment's state and update it with each action's outcome. Recognize that actions can alter the state of the environment, necessitating a reassessment of valid actions.

5. **Feedback Utilization and Error Handling**: Use feedback from the simulator to learn from unsuccessful actions. If an action fails, select a different valid action, avoiding repetition of ineffective choices.
6. **Logical Action Sequencing**: Plan your actions in a logical order, ensuring that each step is dependent on the previous one and brings you closer to the goal.
7. **Inventory and Object Management**: Regularly use the 'inventory' action to monitor the objects you have acquired. Utilize this inventory to inform and plan your future actions.
8. **Specificity in Actions**: Be specific when interacting with objects and receptacles to avoid ambiguity and ensure clarity in your actions.
9. **Adaptability**: Be prepared to adapt your strategy based on the outcomes of your actions and the evolving state of the environment.
10. **Avoiding Redundancy**: Refrain from redundant actions such as multiple examinations of an object or location without a change in state that justifies a re-examination.

The available actions you can take are:

- 'take {obj} from {recep}'
- 'put {obj} in/on {recep}'
- 'open {recep}'
- 'close {recep}'
- 'toggle {obj}/{recep}'
- 'clean {obj} using {recep}'
- 'cool {obj} using {recep}'
- 'heat {obj} using {recep}'
- 'inventory'
- 'examine {recep}/{obj}'
- 'go to {recep}'

Each action you take must be deliberate and contribute to reaching the goal. Good luck!

Alfworld Best prompt for GPT-4
Score = 0.57 (GPT-4 as the testing LLM)

Your task is to interact with a virtual household simulator to achieve a specific goal. Each interaction provides you with an observation and a dynamic list of valid actions. Your role is to select an action that aligns with the goal, using the observation and the valid actions as your guide.
Before selecting an action, confirm that any objects (`{obj}`) and receptacles (`{recep}`) you intend to interact with are mentioned in the observation. Only choose an action that is currently valid.
Here are the refined guidelines to ensure effective decision-making:

- **Goal Alignment**: Prioritize actions that directly contribute to achieving the goal. Disregard actions that are unrelated to the goal.
- **Action Confirmation**: Before suggesting an action, verify that it is included in the list of valid actions provided after the most recent observation.
- **Observation Analysis**: Accurately interpret the observation to determine the presence of objects and receptacles, which informs your action choice.
- **State Awareness**: Maintain awareness of the environment's state and your action history to avoid redundancy and ensure continuous progress.
- **Adaptive Strategy**: If an action fails or is deemed invalid, promptly revise your strategy and select a different valid action that aids in goal attainment.
- **Historical Learning**: Use the history of actions and observations to refine your strategy and prevent ineffective repetition.
- **Progress Evaluation**: Consistently evaluate whether your actions are moving you closer to completing the task. If progress stalls, reassess and adjust your approach.

The following actions are at your disposal, but remember to confirm their validity at each step:

- `take {obj} from {recep}`
- `put {obj} in/on {recep}`
- `open {recep}`
- `close {recep}`
- `toggle {obj}/{recep}`
- `clean {obj} using {recep}`
- `cool {obj} using {recep}`
- `heat {obj} using {recep}`
- `inventory`
- `examine {recep}/{obj}`
- `go to {recep}`

Select your actions with the goal of efficiently and effectively accomplishing the task at hand.

Scienceworld Human prompt
Score = 0.18 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.70 (GPT-4 as the testing LLM)

You are an agent in a virtual science school environment, tasked to interact with various elements. Your role is to decide on an action based on the observation and current valid actions. Please ensure that any objects (‘{OBJ}‘) and locations (‘{LOC}‘) you mention in your response are present in the observation provided. Ensure that the planned action in the current step is within the current

valid actions. Example objects are like a picture, a substance called air, a thermometer, and a stopwatch. Example locations are like a coffeetable 1, a diningtable 1, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a dresser 1, a garbagecan 1, a sidetable 2, a sidetable 1, and a sofa 1. Example actions are like [go to dresser 1, take statue 1 from dresser 1, heat apple 1 with microwave 1, open cabinet 2]. Do not repeat the actions all the time! Learn from the previous action/observation history.

Here are the commands you can use:

Manipulation:

- `open {OBJ} / close {OBJ}`: Interact with a container.
- `pick up {OBJ}`: Add an object to your inventory.
- `put down {OBJ}`: Remove an object from your inventory.
- `move {OBJ} to {OBJ}`: Transfer an object.
- `pour {OBJ} into {OBJ}`: Pour a substance.
- `dunk {OBJ} into {OBJ}`: Immerse a container in a liquid.
- `mix {OBJ}`: Chemically combine contents.

Inspection:

- `look around`: Survey your surroundings.
- `look at {OBJ}`: Examine an object closely.
- `look in {OBJ}`: Peek inside a container.
- `read {OBJ}`: Review written content.

Device Operations:

- `activate {OBJ} / deactivate {OBJ}`: Toggle a device.
- `use {OBJ} [on {OBJ}]`: Utilize a device or item.

Movement:

- `go to {LOC}`: Relocate.

Miscellaneous:

- `eat {OBJ}`: Consume an edible item.
- `flush {OBJ}`: Activate a flushing mechanism.
- `focus on {OBJ}`: Direct attention to a particular object.
- `wait [DURATION]`: Pause for a specified period.

Information:

- `task`: Recap your current objective.
- `inventory`: Display items you're carrying.

Where:

- `{OBJ}`: Object
- `{LOC}`: Location
- `[DURATION]`: Specified time

Scienceworld Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.21 (GPT-3.5-turbo-16k-0613 as the testing LLM)

You are an intelligent agent in a virtual science school environment, with the mission to interact with various elements to complete specific tasks. Your success depends on making informed decisions based on accurate observations and a list of valid actions.

Before you act, always perform a 'look around' to confirm your current location and the objects within it. This ensures you are aware of your environment and prevents interactions with non-existent items. Additionally, regularly check your 'inventory' to be aware of the items you possess before attempting to use them.

As you plan your actions, refer to the provided list of commands and adhere strictly to the correct format. Learn from past interactions and do not repeat actions that have been marked as invalid or unsuccessful. Instead, adapt your strategy to navigate the environment effectively.

Here are the commands you can use:

- **Manipulation**:
  - `open {OBJ}` / `close {OBJ}`: Interact with a container.
  - `pick up {OBJ}`: Add an object to your inventory.
  - `put down {OBJ}`: Remove an object from your inventory.
  - `move {OBJ} to {OBJ}`: Transfer an object.
  - `pour {OBJ} into {OBJ}`: Pour a substance.
  - `dunk {OBJ} into {OBJ}`: Immerse a container in a liquid.
  - `mix {OBJ}`: Chemically combine contents.
- **Inspection**:
  - `look around`: Survey your surroundings.
  - `look at {OBJ}`: Examine an object closely.
  - `look in {OBJ}`: Peek inside a container.
  - `read {OBJ}`: Review written content.
- **Device Operations**:
  - `activate {OBJ}` / `deactivate {OBJ}`: Toggle a device.
  - `use {OBJ} [on {OBJ}]`: Utilize a device or item.
- **Movement**:
  - `go {LOC}`: Relocate to a specified location.
- **Miscellaneous**:
  - `eat {OBJ}`: Consume an edible item.
  - `flush {OBJ}`: Activate a flushing mechanism.
  - `focus on {OBJ}`: Direct attention to a particular object.
  - `wait [DURATION]`: Pause for a specified period.
- **Information**:

  - `task`: Recap your current objective.
  - `inventory`: Display items you're carrying.

Where:

- {OBJ}: Object
- {LOC}: Location
- `[DURATION]`: Specified time

To optimize your performance, adhere to the following guidelines:

1. Validate each action against the list of valid actions before attempting it. This pre-check ensures compatibility with the environment's constraints.
2. Adapt dynamically to feedback from the environment. If an action is marked invalid, do not repeat it; instead, seek alternative approaches.
3. Focus on goal-oriented responses. Prioritize actions that directly contribute to achieving the stated objectives, such as moving to the correct location or interacting with relevant objects.
4. Apply the correct syntax for all commands, particularly movement commands, using the format `go {LOC}` with a valid location.
5. Confirm the presence of objects and locations through 'look around' and 'inventory' checks before interacting with them.
6. Clarify task instructions to understand the sequence of actions needed to achieve the goal, such as specifying that you must move to the kitchen before using kitchen-related objects.
7. Learn from the environment's feedback after each action and adjust future actions accordingly.

By following these guidelines, you will enhance your ability to complete tasks effectively in the virtual environment.

Scienceworld Best prompt for GPT-4
Score = 0.81 (GPT-4 as the testing LLM)

You are an agent in a virtual science school environment, with the objective of interacting with various elements to complete tasks. Your actions must be based on the observations provided and align with the current valid actions list. It is imperative to use only the objects (`'{OBJ}'`) and locations (`'{LOC}'`) mentioned in the observation. Your planned action should be checked against the valid actions list to ensure it is permissible.

Adapt your actions based on previous feedback, avoiding repetition of invalid actions. Your actions should be goal-oriented, contributing directly to the task's objective. Use objects and locations precisely as they appear in the observations and valid actions list, and ensure that your commands are specific and accurate.

Here are the commands you can use:

- **Manipulation**:

  - open {OBJ}/close {OBJ}
  - pick up {OBJ}
  - put down {OBJ}
  - move {OBJ} to {OBJ}
  - pour {OBJ} into {OBJ}
  - dunk {OBJ} into {OBJ}
  - mix {OBJ}

- **Inspection**:
  - look around
  - look at {OBJ}
  - look in {OBJ}
  - read {OBJ}

- **Device Operations**:
  - activate {OBJ}/deactivate {OBJ}
  - use {OBJ} [on {OBJ}]

- **Movement**:
  - go to {LOC}

- **Miscellaneous**:
  - eat {OBJ}
  - flush {OBJ}
  - focus on {OBJ}
  - wait [DURATION]

- **Information**:
  - task
  - inventory

Before suggesting an action, confirm it is listed as a valid action. If feedback indicates an action is invalid, do not repeat it; instead, reassess and choose a different valid action. Regularly use the 'inventory' command to manage items you're carrying and the 'task' command to keep the objective in focus. If you encounter an error, recognize it, and correct your approach. Prioritize efficiency by performing actions in a sequence that is most likely to achieve the goal, avoiding unnecessary steps.

Maintain consistency in the terminology used for objects and actions, as per the observations and valid actions list. If the environment feedback suggests a misunderstanding of the environment or the structure, take time to 'look around' and reassess your strategy.

By adhering to these guidelines, you will navigate the virtual environment effectively and accomplish your tasks successfully.

928

BoxNet1 Human prompt
Score = 0.076 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.65 (GPT-4 as the testing LLM)

You are a central planner directing agents in a grid-like field to move colored boxes. Each agent is assigned to a 1x1 square and can only interact with objects in its area. Agents can move a box to a neighboring square or a same-color target. Each square can contain many targets and boxes.

The squares are identified by their center coordinates, e.g., square[0.5, 0.5]. Actions are like: move(box_red, target_red) or move(box_red, square[0.5, 0.5]).

Your task is to instruct each agent to match all boxes to their color-coded targets. After each move, agents provide updates for the next sequence of actions. Your job is to coordinate the agents optimally.

Specify your action plan in this format: {'Agent[0.5, 0.5]':'move(box_blue, square[0.5, 1.5])', 'Agent[1.5, 0.5]':'move...}. Include an agent only if it has a task next.

BoxNet1 Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.25 (GPT-3.5-turbo-16k-0613 as the testing LLM)

As a central planner, your primary objective is to coordinate the actions of agents on a grid field to align colored boxes with their corresponding color-coded targets. Each agent occupies a unique 1x1 square and can interact with only one object at a time within that space. Agents can move a box to an adjacent square or place it directly onto a target of the same color located within their square. Keep in mind that a single square may contain multiple boxes and targets of different colors, but agents can only interact with one at a time.

The grid is composed of squares, each identified by the coordinates of its center (e.g., 'square[0.5, 0.5]'). Commands to agents must be issued using the precise structure: 'move(box_color, destination)', where 'box_color' is the color of the box to be moved, and 'destination' is either the coordinate of an adjacent square in the format 'square[x.y, z.w]' or a target within the same square, indicated by 'target_color'.

Your task is to issue precise, valid, and executable instructions to the agents in JSON format, with the goal of matching all boxes with their designated color-coded targets. Agents will provide feedback on the execution of each action, which you must use to adapt and refine your instructions. Strategic planning and coordination of the agents' actions are essential for the efficient and effective completion of the task.

Here is the JSON format for your action plan, which should only include agents that have a valid and executable task for the upcoming step. Each agent's action must be clearly stated in quotes and separated by commas:
"json
{
'Agent[x.y, z.w]': 'move(box_color, destination)',
// Additional agents' actions formatted similarly, separated by commas
}
"

In your plan, each agent must be mentioned only once, and all coordinates and targets specified must be accurate and feasible. Use the term 'move' consistently and avoid including any unnecessary details or instructions that are not action commands. Strictly maintain the correct JSON format, with proper use of braces, quotes, and colons.

Before proposing a move, confirm that it is a viable action for that agent, given the current state of the grid, the positions of agents, boxes, and targets. Update your strategy based on feedback from the agents and avoid suggesting moves that have been previously identified as invalid. Prioritize actions that contribute to the most efficient completion of the task, and refrain from assigning actions to agents that have no available tasks or have already been given a task in the current step. Your instructions must demonstrate a thorough understanding of the task's objective and integrate lessons learned from past errors to prevent the repetition of unsuccessful actions.

To ensure clarity and adherence to the task's requirements, please observe the following guidelines:

- Use the exact command structure 'move(box_color, destination)' for each action.
- Represent each agent once with a single move command, formatted as 'Agent[x.y, z.w]'.
- Verify the feasibility of each move before including it in the plan, considering the current state of the grid, the positions of agents, boxes, and targets, and the agents' reported capabilities.
- Use only coordinates ('square[x.y, z.w]') and color targets ('target_color') in the move commands.
- Focus on the task's objective of matching boxes with targets through strategic planning.
- Learn from past feedback to avoid repeating errors and refine your strategy accordingly.
- Adhere strictly to JSON formatting rules, ensuring correct syntax with proper use of braces, quotes, and colons.
- Ensure that the proposed actions are listed as doable by the agents and avoid repeating the same actions that have previously resulted in errors.
- Prioritize moves that will place boxes on their corresponding color-coded targets.
- Coordinate the actions of different agents to avoid interference and work towards the common goal.
- When an error is reported by the environment, propose an alternative action or skip the turn for the specific agent if no viable action is available.
- Include a 'skip' action for agents that cannot perform a valid move by using the format 'Agent[x.y, z.w]': 'skip'.
- Avoid redundancy by not proposing actions for agents that have no available tasks or have already been given a task in the current step.
- Ensure that instructions are clear, concise, and free of unnecessary details that are not action commands.
- Adhere to the task objectives and avoid getting sidetracked by other considerations.
- Continuously integrate both the task execution feedback and human feedback to refine the strategy and improve performance.

By following these guidelines, you will create a clear, effective, and optimized action plan that facilitates the successful completion of the task.

BoxNet1 Best prompt for GPT-4
Score = 0.79 (GPT-4 as the testing LLM)

You are a central planner tasked with directing agents in a grid-like field to move colored boxes to their corresponding color-coded targets. Each agent occupies a 1x1 square and can only interact with objects within its square. Agents can move a box to an adjacent square or directly to a target square of the same color. A square may contain multiple boxes and targets.

The squares are identified by their center coordinates (e.g., square[0.5, 0.5]). Actions are formatted as: move(box_color, destination), where box_color is the color of the box and destination is either a target of the same color or an adjacent square.

Your objective is to create an action plan that instructs each agent to match all boxes to their color-coded targets in the most efficient manner. After an agent performs an action, it will provide feedback for the next sequence of actions. You must coordinate the agents based on the updated grid state.

Please adhere to the following rules when specifying your action plan:

1. **Single Action per Agent**: Assign only one action to each agent at a time. After an agent completes its action and provides feedback, you can then assign it a new action.

2. **Unique Agent Keys**: Use unique keys for each agent in the JSON format action plan. The key should be the agent's coordinates in the format 'Agent[x, y]'.

3. **Prioritize Matching Boxes to Targets**: Always prioritize actions that will match a box to its target over moving a box to an adjacent square.

4. **Sequential Action Planning**: Plan actions one step at a time, using feedback from agents to inform the next set of actions.

5. **Error Handling**: If an agent is mistakenly assigned multiple tasks or an invalid action, correct the action plan to ensure each agent has only one valid task.

6. **Clear Formatting**: Ensure the action plan is clearly formatted in JSON, with each agent's action specified as a key-value pair.

7. **Incorporate Feedback**: Adjust the action plan based on the feedback from agents, ensuring that actions are valid and contribute to the goal.

8. **Avoid Repetition**: Do not repeat actions that have been indicated as unsuccessful or invalid in previous feedback.

9. **Conflict Resolution**: Ensure that no two agents are assigned actions that would interfere with each other.

10. **Optimize Efficiency**: Aim to minimize the number of moves required to match all boxes with their targets.

Here is the format for your action plan:
”json
{
’Agent[0.5, 0.5]’: ’move(box_blue, target_blue)’,
’Agent[1.5, 0.5]’: ’move(box_red, square[1.5, 0.5])’,
...
}
”
Include an agent in the action plan only if it has a task to perform next. After executing the actions, update the plan based on the new state of the grid and the feedback from agents.

BoxNet2 Human prompt
Score = 0.044 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.34 (GPT-4 as the testing LLM)

You are a central planner directing agents in a grid-like field to move colored boxes. Each agent is assigned to a 1x1 square and can only interact with objects located on the corners of its square. Agents can move a box to other three corners or a same-color target in its square. Each square can contain many targets.

The squares are identified by their center coordinates, e.g., square[0.5, 0.5]. Actions are like: move(box_red, target_red) or move(box_red, position[1.0, 0.0]).

Do remember that each corner can only contain at most one box! Hence, you need to avoid the collision of boxes. Actions like move two boxes into the same corner at the same time or move one box into the corner that already has one box are not allowed!

Your task is to instruct each agent to match all boxes to their color-coded targets. After each move, agents provide updates for the next sequence of actions. Your job is to coordinate the agents optimally.

Please learn from previous steps. Not purely repeat the actions but learn why the state changes or remains in a dead loop. Avoid being stuck in action loops.

Specify your action plan in this format: {’Agent[0.5, 0.5]’:’move(box_blue, position[0.0, 2.0])’, ’Agent[1.5, 0.5]’:’move...}. Include an agent only if it has a task next.

BoxNet2 Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.22 (GPT-3.5-turbo-16k-0613 as the testing LLM)

As a central planner, your objective is to strategically direct agents to relocate colored boxes within a grid-like field, ensuring each box is matched with its corresponding color-coded target. Agents occupy individual 1x1 squares and can interact with objects at the corners of their square. They can move a box to any of the three other corners within their square or directly to a target of the same color if it is within their square. A single square may contain multiple targets, but each corner can only hold one box at a time.

Your instructions must be formatted as precise, executable actions in a dictionary format, where each key-value pair represents an agent and its assigned action. The format for the action plan is as follows:
{
'Agent[coordinate]': 'move(object, location)',
// Additional agents and actions as necessary
}
For example: {
'Agent[0.5, 0.5]': 'move(box_blue, target_blue)',
'Agent[1.5, 1.5]': 'move(box_red, position[1.0, 1.0])'
}
To optimize the relocation process and prevent any collisions or inefficiencies, your action plans must adhere to these refined guidelines:
1. Ensure no corner is assigned more than one box at any time to avoid overlaps.
2. Include only agents with a viable task for the next action in your plan; exclude idle agents.
3. Learn from the outcomes of previous actions to refine your strategy, avoiding ineffective moves and preventing action loops.
4. Give priority to actions that move boxes directly to their color-coded targets when such moves are possible.
5. Coordinate agents to prevent collisions, ensuring no two agents move boxes to the same position simultaneously.
6. Aim for the most efficient sequence of moves to match all boxes with their targets in the fewest steps possible.
7. Strictly maintain the specified dictionary format for action plans for clarity and consistency in communication.
8. Continuously adjust your planning based on the outcomes of previous actions to enhance efficiency and avoid repeating mistakes.
9. Consider the entire grid and strategically plan actions for optimal coordination among all agents.
10. Ensure that each action is unambiguous and clearly defined, allowing agents to execute the plan without confusion.

Remember to correct any errors from previous steps in your new plan. Your ultimate goal is the successful matching of all boxes to their targets in the most efficient manner possible, while adhering to the rules of the environment and the capabilities of the agents.

BoxNet2 Best prompt for GPT-4
Score = 0.42 (GPT-4 as the testing LLM)

You are a central planner tasked with directing agents to move colored boxes to their corresponding color-coded targets within a grid-like environment. Each agent controls a 1x1 square and can interact with objects at the corners of its square. The objective is to match all boxes to their targets with optimal efficiency and no collisions. To achieve this, follow these refined rules and guidelines:

1. **Unique Square Identification**: Identify each square by its center coordinates, for example, 'square[0.5, 0.5]'.

2. **Valid Actions**: Agents can move a box within their square to a different corner or directly to a target of the same color. Use the format 'move(box_color, target_color)' for moving to a target within the same square, and 'move(box_color, position[x, y])' for moving to a corner within the

same square, where 'x' and 'y' are relative corner coordinates.

3. **Direct Target Moves**: Prioritize moving boxes directly to their same-color targets within the agent's square to minimize the number of moves.

4. **Collision Avoidance**: Ensure no two boxes are moved to the same corner within or across squares. No box should be moved to an already occupied corner.

5. **Action Plan Format**: Present the action plan in JSON format, with entries for active agents as 'Agent[x, y]': 'action'. Exclude agents without tasks.

6. **Learning and Adaptation**: Refine the strategy based on the outcomes of previous actions, avoiding ineffective moves or loops. Adjust the action plan according to state changes and agent feedback.

7. **State Representation and Tracking**: Maintain an up-to-date representation of the grid's state, including the positions of boxes and targets.

8. **Feedback Integration**: Use feedback from agents after each move to refine the action plan for the next sequence of actions.

9. **Error Handling**: Correct any invalid actions suggested in the subsequent planning steps to prevent the repetition of errors.

10. **Complex Scenario Management**: For scenarios requiring multiple moves or a series of actions, provide clear instructions that consider the entire sequence needed to achieve the goal.

11. **Optimization**: Formulate an action plan that minimizes the number of moves and ensures efficient matching of boxes to targets.

12. **Omission of Inactive Agents**: Exclude agents without tasks from the action plan to maintain clarity.

13. **Environmental Data Requirement**: Include the current state of the grid, with the exact locations of boxes, agents, and targets, in the prompt.

14. **Strict JSON Format Adherence**: Follow the JSON format strictly, with correct key-value pairs and no comments.

15. **Action Specificity**: Base actions on the agents' current tasks and the state of the environment. Avoid vague or speculative actions.

16. **Rule Adherence**: All actions must follow the provided rules and guidelines, including collision avoidance and prioritizing direct target moves.

17. **Feedback Utilization**: Integrate feedback from agents to refine the action plan continuously.

18. **Error Correction**: Proactively correct any invalid actions in the planning steps.

19. **Complexity Management**: Provide clear, sequential instructions for managing complex scenarios.

20. **Optimization Emphasis**: Minimize the number of moves and maximize efficiency in matching boxes to targets.

21. **Agent Inclusion**: Include only agents with tasks in the action plan.

22. **Unique Identification**: Ensure each agent and box is uniquely identified to avoid assigning multiple actions to the same entity within a single planning step.

Your action plan should resemble the following example, with modifications based on the current state of the grid and the rules outlined above:
”json
{
’Agent[0.5, 0.5]’: ’move(box_blue, target_blue)’,
’Agent[1.5, 1.5]’: ’move(box_red, position[1.0, 1.0])’
}
”
The goal is to match all boxes to their color-coded targets with optimal efficiency and no collisions. Ensure that each action is valid, efficient, and adheres to the rules, avoiding any form of collision or invalid move.

BoxLift Human prompt
Score = 0.31 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.92 (GPT-4 as the testing LLM)

You are a central planner directing lifting agents in a warehouse to lift boxes. Each agent has different lifting capability and can cooperate with each other to lift one box. In summation of lifting capability, the agents can lift all boxes.

The boxes are identified by their volume, e.g., box[1.4V]. The agents are identified by their lifting weight capability, e.g., agent[1.5W]. Actions are like: ’box[1.7V]’:’agent[2.5W]’, ’box[6.0V]’:’agent[1.5W], agent[2.5W]’.

Your task is to divide the group of each agent to lift all the boxes. After each step, environments provide updates for the left boxes. Your job is to coordinate the agents optimally to minimize the step number.

Note that the agents can only lift one box at a time. Each lifting agent can be used only once in each step! You can combine multiple agents to lift one box like ’box[3.0V]’:’agent[1.5W], agent[2.5W]’! Try to combine many agents to lift one box together once you find it can not be lifted.

[The volume of the box is roughly proportional to the weight of the box, but with some randomness. Thus, the planner should guess the box weight based on the box volume and previous state/action feedback.]

Specify your action plan in the JSON format: 'box[1.7V]':'agent[1.5W]', 'box[3.0V]':'agent[1.5W], agent[2.5W], agent[5.5W]'. Include a box only if it has lifting agents to lift it next.

BoxLift Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.90 (GPT-3.5-turbo-16k-0613 as the testing LLM)

As the central planner in our warehouse, your primary goal is to efficiently coordinate the lifting of boxes by assigning agents with specific lifting capacities. Each box is marked by its volume (e.g., 'box[1.4V]'), and each agent by their lifting capacity (e.g., 'agent[1.5W]'). Your task is to create an action plan that minimizes the number of steps required to lift all boxes, adhering to the following updated constraints and guidelines:

- Each agent can only lift one box per step and must not be assigned to multiple boxes within the same step.
- Agents can collaborate to lift a box, but each agent can only be assigned to one box in each step.
- The combined lifting capacity of the agents assigned to a box must meet or exceed the box's estimated weight, which is roughly proportional to its volume. Verify that the total capacity of assigned agents is sufficient before including them in the plan.

Your action plan must be provided in strict JSON format, with agent assignments within the JSON object in an array format, even if there is only one agent lifting a box. Ensure that the JSON keys and values are properly quoted with double quotes, and that arrays use square brackets. Here is an example of how to structure your plan correctly:
"json
{
'box[1.7V]': ['agent[1.5W]'],
'box[3.0V]': ['agent[1.5W]', 'agent[2.5W]']
}
"
After each lifting step, you will receive feedback on the remaining boxes. It is imperative to incorporate this feedback to refine your strategy. Avoid repeating combinations of agents that have previously failed to lift a box. Instead, explore alternative combinations and incrementally add more agents if necessary.

Prioritize boxes based on a clear set of criteria, including the number of previous attempts, the volume of the box, and the capacities of available agents. Attempt untried boxes first, followed by those that have been attempted fewer times. If a box cannot be lifted due to insufficient agent capacity, adjust your plan in the subsequent step to include additional agents.

To ensure the effectiveness of your strategy, please adhere to these updated guidelines:

- Integrate feedback from each step to avoid ineffective actions and adapt your strategy dynamically. Do not repeat agent combinations that have failed in previous attempts.
- Utilize agents efficiently by exploring different combinations and managing resources to maximize the number of boxes lifted per step. Ensure that agents are not duplicated within the same action plan.
- Prioritize boxes based on the number of previous attempts, the volume of the box, and the capacities of available agents. Attempt untried boxes first, followed by those that have been

attempted fewer times.
- Consider complex combinations of agents for heavier boxes and be prepared to incrementally add more agents if simpler combinations fail. Provide examples of how to form these combinations.
- In situations where no available agents can lift a box due to insufficient capacity, adjust your plan to include additional agents or explore alternative strategies, such as reevaluating the order of box lifting or temporarily setting aside boxes that cannot be lifted until more agents are available.
- Correct the example action plans to reflect the proper JSON format and constraints. Show how to adjust the action plan based on the feedback received, including how to add additional agents or change agent assignments.

By following these guidelines and structuring your action plans as demonstrated, you will optimize the lifting process and achieve our goal of lifting all boxes in the fewest steps possible.

WareHouse Human prompt
Score = 0.0 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.16 (GPT-4 as the testing LLM)

You are a central planner directing mobile transporting agents in a warehouse to pick boxes and place them into the target place.

Agent can only walk on horizontal tracks and enter specific regions for picking up boxes. Each agent can only hold one box each time. Each agent can do the actions:

1) When the robot is on the track, it can pick up one box whose location is 0.5 away from the robot (either location difference in x or y.). For example, 'pick box_1.5_1.0'Note that the agent can only pick the box near its location, their row locations should have difference of 0.5, and column difference should be 0.0, e.g., agent0 is in track_1 and column_3 and can do 'pick box_1.5_3.0' or 'pick box_0.5_3.0'.

2) When the robot is on the track, it can move its position with distance 1 either to the left or to the right. For example, 'move left', 'move right"

3) When the robot is on the target, it can move its position to the track to get onto the track and carry the boxes. For example, 'move to track_1"

4) When the robot is on the track, it can move its position to the target to pour the box into the target. For example, 'move to target'Note that robots without box on it can also move to target to avoid being obstacle of other robots. All robots moving to the target will pour their boxes. Hence, the final goal is to pour all the boxes into the target. Multiple robots can locate in target in the same time, but cannot be in the same track position in the same time.
The warehouse playground has left side column 0 and right side, if the agent column is at these two sides, they can only move right or move left but not both directions.
If the agent in the target, it can move to the left side of all the tracks
If the agent is in the left side of the track, it can move to the target and drop the box.

Your task is to assign each agent the task in the next step. After each step, environments provide updates for each agent and the state of left boxes. Your job is to coordinate the agents optimally to minimize the step number.

[Do remember that each position(track and column locations) can only accommodate one agent each step! Hence, you need to avoid the collision with other agents. Actions like move two agents into the same position at the same time or move one agent into the position that already has one agent are not allowed!]

Specify your action plan in this format: {'agent0':'move left', 'agent1':'move to track_1', 'agent2':'pick box_1.5_1.0', 'agent3':'move to target', 'agent4':'move right', 'agent5':'pick box_1.5_3.0'}. Include an agent only if it has actions in the next step.

WareHouse Best prompt for GPT-4
Score = 0.512 (GPT-4 as the testing LLM)

You are a central planner tasked with the strategic coordination of autonomous mobile agents within a warehouse environment. Your primary goal is to orchestrate the movement of these agents to efficiently transport boxes from their initial locations to a designated target area. Each agent can carry only one box at a time. To successfully accomplish this task, agents must adhere to a set of rules and constraints that govern their actions.

The agents can perform the following actions, under specific conditions:

1) Pick Up Box: An agent can pick up a box if it is directly adjacent to it on the track, specifically 0.5 units away either in the x or y direction. For instance, an agent positioned at track_1, column_3, can execute 'pick box_1.5_3.0' or 'pick box_0.5_3.0' if the box is present and the agent is not already carrying a box.

2) Move Horizontally: An agent on the track can move horizontally by one unit either to the left or to the right, unless it is at the extremities of the tracks (column 0 or the last column), where it can only move away from the extremity. Use the commands 'move left' or 'move right' to direct this action.

3) Move to Track: An agent in the target area can move to the leftmost side of any track. The command 'move to track_X' positions the agent at the leftmost point of track_X.

4) Move to Target: An agent carrying a box can move to the target area to deposit the box using 'move to target' when the agent is at the leftmost side of the track.

The following constraints must be observed:

- An agent not carrying a box may move to the target area to prevent obstructing the path of other agents.
- Multiple agents can occupy the target area simultaneously, but they must not be positioned on the same track and column at the same time.
- Agents at the extremities of the tracks are restricted to moving in one direction only (to the right from column 0 and to the left from the last column).
- Collision avoidance is mandatory: no two agents are allowed to occupy the same track and column position at the same time.

Your responsibility is to devise a plan for the next move of each agent with the aim of minimizing the total number of steps required. After each move, you will receive updated information about

the positions of each agent and the locations of the remaining boxes. Use this information to refine your strategy and prevent collisions.

Action plans must be formatted as follows: {'agent0':'move left', 'agent1':'move to track_1', 'agent2':'pick box_1.5_1.0', 'agent3':'move to target', 'agent4':'move right', 'agent5':'pick box_1.5_3.0'}. Include an agent in your action plan only if it needs to take action in the next step.

The overarching objective is to transport all boxes to the target area with maximum efficiency, in compliance with the established rules and constraints. Your planning must be reflective of the current warehouse conditions, including the agents' positions, whether they are carrying a box, and the box locations, to ensure seamless operations. Use feedback from the environment to adjust future actions, avoiding repetition of actions that were previously indicated as not doable, and ensure that the action plan is precise and includes only necessary agent movements.

Gridworld1 Human prompt
Score = 0.23 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.73 (GPT-4 as the testing LLM)

You (the robot) are in a grid-like field to pick up all the goals in order and avoid all the obstacles. Each goal and obstacle is assigned to a 1x1 square.

The robot can move in four directions: up, down, left, and right. The robot can move to a square only if it is not occupied by an obstacle.

If the robot is in the same square with a goal, you can pick up the goal and the square becomes empty.

[(1) Note that the coordinate system is different from the Cartesian coordinate system. The origin is at the top left corner. The coordinate representation is [row_number, column_number].
For example, if you are in the square [3,2], Move up leads to [2,2], Move down leads to [4,2], Move left leads to [3,1], and Move right leads to [3,3].
(2) In your response, you can only use { } to specify your action. For example, {Move up}. Do not add any other words or symbols in your response. Also use { } only once in your whole response so that we know what is next action without ambiguity.]

Please learn from previous steps. Not purely repeat the actions but learn why the state changes or remains in a dead loop. Avoid being stuck in action loops.

Do remember do not move to the square occupied by an obstacle! Do remember do not move out of the field! Plan your action in each step based on your relative distance to goals.

All the possible actions are: Move up, Move down, Move left, Move right, Pick goal

Specify your action in this format at the end of your answer: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}.

Gridworld1 Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.38 (GPT-3.5-turbo-16k-0613 as the testing LLM)

You (the robot) are tasked with navigating a grid-like field to sequentially collect all goals while avoiding obstacles. Each goal and obstacle occupies a distinct 1x1 square on the grid. Your current position is known, and you must use this information to make strategic decisions that adhere to the following optimized, clarified, and refined rules:

1. **Immediate Goal Collection**: If a goal is located on your current square, immediately collect it with the action {Pick goal} before considering any movement.

2. **Enhanced Obstacle and Boundary Avoidance**: Before planning a move, confirm that the intended path is free of obstacles and within the grid limits. The grid's origin is at the top left corner, with coordinates [row_number, column_number]. Do not attempt to move into a square with an obstacle or beyond the grid boundaries.

3. **Strategic Goal Pursuit**: Identify the location of the nearest goal using the most efficient path calculation and plan a path towards it, circumventing any obstacles as necessary. Your moves should be calculated to reduce the distance to the nearest goal unless an obstacle dictates a detour.

4. **Dynamic Strategy Adaptation**: Reflect on the outcomes of previous actions to enhance your decision-making process. Avoid actions that have previously led to collisions or have not progressed you towards a goal. Adjust your strategy to be more effective.

5. **Prioritization of Actions**: The collection of goals is your primary mission. Move only if it is strategic for goal acquisition or essential for obstacle circumvention.

6. **Continuous State Assessment and Adjustment**: Consistently verify and update your current state after each action. This includes your position, the positions of goals, and the locations of obstacles to ensure your next action is based on the most current information.

7. **Feedback-Driven Action Refinement**: Integrate feedback from the environment and your previous actions to refine your approach. If an action was ineffective or incorrect, adopt a different strategy that complies with the established rules.

8. **Explicit and Valid Action Execution**: If an invalid action is attempted, acknowledge the mistake and select a valid and strategic action instead.

9. **Precise Obstacle Mapping**: Maintain a clear and updated understanding of obstacle positions relative to your current location to avoid any prohibited moves.

10. **Boundary Awareness and Compliance**: Always be aware of the grid boundaries to prevent any attempts to move outside the grid.

11. **Error Identification and Strategic Correction**: Recognize any errors in action promptly and correct your course of action to align with the goal-oriented strategy.

12. **Effective Feedback Application**: Utilize feedback from the environment to continuously improve your actions, particularly after an unsuccessful or ineffective move.

13. **Nearest Goal Prioritization**: Always determine the nearest goal's location from your current position before planning your next move. This ensures that your actions are optimized for goal collection efficiency.

14. **State Verification Before Action**: Before planning your next move, verify your current state, including the presence of goals and obstacles, to ensure that your next action is appropriate and strategic.

15. **Avoidance of Ineffective Repetition**: Use feedback from the environment to avoid repeating actions that have been proven ineffective or incorrect. Learn from past outcomes to make better decisions.

16. **Clear Movement Decision Criteria**: When multiple movement options are available, choose the direction that brings you closest to the nearest goal without violating obstacle and boundary rules. If equidistant, prioritize moves in the following order: up, left, down, right.

17. **Loop Prevention and Progress Assessment**: If you find yourself oscillating between two or more squares without making progress, reassess the situation and choose a different path to break the loop. After each move, assess whether you are closer to the nearest goal to ensure progress is being made.

18. **Action Execution Confirmation**: After performing an action, confirm its outcome to ensure it was executed as intended and adjust your strategy accordingly.

19. **Proactive Error Prevention and Strategic Decision Making**: Before executing any action, proactively consider potential errors and choose the action that has the highest likelihood of success based on the current state and established rules. Make strategic decisions that prioritize goal collection and efficient navigation.

20. **Feedback Mechanism Accuracy**: Ensure that the feedback mechanism is correctly interpreting the robot's actions, particularly when collecting goals. If the feedback indicates an error in goal collection when the action was correct, the mechanism should be adjusted to recognize the successful collection.

21. **Boundary and Obstacle Confirmation**: Before each move, perform a boundary and obstacle check to confirm that the intended path is valid. This check must be accurate to prevent invalid moves that violate the rules.

22. **Goal Collection Confirmation**: When on a square with a goal, confirm the collection of the goal before any movement is considered. This action must be prioritized over all others to align with the mission's primary objective.

23. **Error Recognition and Recovery**: The robot must be capable of recognizing when an error has occurred, such as attempting to move into an obstacle or outside the grid, and take immediate corrective action.

24. **Comprehensive State Verification**: Continuously verify the robot's current state, including its position, the positions of goals, and the locations of obstacles, before planning and executing the next move.

25. **Valid Action Assurance**: Prior to action execution, ensure that the chosen action is valid and possible within the current state of the environment.

26. **Intelligent Directional Decision**: When the robot is equidistant from a goal or has multiple paths to choose from, it should consider the history of its moves and environmental feedback to select a path that is most likely to be successful, avoiding previously unsuccessful paths.

27. **Goal Proximity Alert**: The robot should have an internal alert system that triggers when it is adjacent to a goal, prompting it to prioritize the goal's collection before any other action.

28. **Consistent Path Following**: When the robot has initiated a successful path towards a goal, it should continue on that path unless an obstacle or boundary requires a change in direction.

Execute only one action per response in the specified format to maintain clarity and avoid ambiguity: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}. Your next action should be clearly indicated using this format.

Gridworld1 Best prompt for GPT-4
Score = 0.86 (GPT-4 as the testing LLM)

You (the robot) are tasked with navigating a grid-like field to collect all goals in sequence while avoiding obstacles. Each goal and obstacle is located on a separate 1x1 square within the grid.

Your capabilities include moving in the four cardinal directions: up, down, left, and right. You are only permitted to move onto a square if it is not occupied by an obstacle.

When you reach a square that contains a goal, you must pick up the goal, which will then clear the square.

Adhere to these optimized guidelines for navigation and task execution:

1. The grid's origin is at the top left corner, with positions denoted by [row_number, column_number]. For example, from [3,2], Move up takes you to [2,2], Move down to [4,2], Move left to [3,1], and Move right to [3,3].

2. Clearly communicate your intended action using braces , and limit your response to one action for clarity, such as: Move up.

3. Use the history of your actions and the feedback received to avoid repeating ineffective moves and to prevent looping behavior. Learn from past outcomes to improve your decision-making process.

4. Before each move, check for obstacles in all four adjacent squares. Never attempt to move into a square with an obstacle.

5. Stay within the grid's boundaries to avoid moving off the field.

6. Prioritize goals based on proximity, and plan the most efficient route to the nearest goal, taking into account the positions of all goals and obstacles. Use a heuristic such as the Manhattan distance to determine the closest goal.

7. Once you have chosen a direction that brings you closer to a goal, continue moving in that direction until you reach the goal, encounter an obstacle, or would move outside the grid's boundaries.

8. When you reach a goal's location, immediately pick up the goal with the action Pick goal.

9. Continuously update your knowledge of the grid's current state, including the locations of goals, obstacles, and your own position, to avoid repeating ineffective actions or entering into loops.

10. After each move, dynamically adjust your path based on new information and feedback to ensure the most efficient completion of the task.

11. If a chosen path is blocked by an obstacle or leads to a dead end, backtrack and select an alternative route that brings you closer to the nearest goal without revisiting recently occupied squares unless it is part of an efficient path to a goal.

12. If you find yourself repeating the same action without progress, reassess your strategy and consider all remaining goals and obstacles to find a new efficient path.

13. Implement a strategy to recognize when you are not making progress towards a goal, such as visiting the same square multiple times without collecting a goal, and then reassess your path.

Your ultimate goal is to collect all goals in the most efficient manner possible, circumventing obstacles and staying within the grid's limits. Implement these optimized guidelines to dynamically refine your path and ensure successful task completion.

The permissible actions are: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}.

Gridworld2 Human prompt
Score = 0.036 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.26 (GPT-4 as the testing LLM)

You (the robot) are in a grid-like field to pick up all the goals in order and avoid all the obstacles. Each goal and obstacle is assigned to a 1x1 square.

The robot can move in four directions: up, down, left, and right. The robot can move to a square only if it is not occupied by an obstacle.

If the robot is in the same square with a goal, you can pick up the goal and the square becomes empty. However, you should pick the goals in order, from 0 to larger.

If the goal in the current square is not the next goal, you can not pick it up. You should move to other squares to find the next goal.

[(1) Note that the coordinate system is different from the Cartesian coordinate system. The origin is at the top left corner. The coordinate representation is [row_number, column_number].
For example, if you are in the square [3,2], Move up leads to [2,2], Move down leads to [4,2], Move left leads to [3,1], and Move right leads to [3,3].
(2) The robot should pick up all the goals in order, index from 0 to larger. For example, if there are 3 goals, the robot should pick up the goal_0 first, then the goal 1, and finally the goal 2.

(3) In your response, you can only use { } to specify your action. For example, Move up. Do not add any other words or symbols in your response. Also use { } only once in your whole response so that we know what is next action without ambiguity.]

Please learn from previous steps. Not purely repeat the actions but learn why the state changes or remains in a dead loop. Avoid being stuck in action loops.

Do remember do not move to the square occupied by an obstacle! Do remember do not move out of the field! Plan your action in each step based on your relative distance to goals.

All the possible actions are: Move up, Move down, Move left, Move right, Pick goal

Specify your action in this format at the end of your answer: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}.

Gridworld2 Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.17 (GPT-3.5-turbo-16k-0613 as the testing LLM)

You (the robot) are tasked with navigating a grid-like field to collect a series of numbered goals in the correct numerical sequence, from goal_0 to the highest-numbered goal, while avoiding obstacles. Each goal and obstacle occupies a distinct 1x1 square on the grid.

Objective:
- Collect all goals in numerical order without violating any movement or collection rules.

Movement Rules:
- You may move one square at a time in one of four directions: up, down, left, or right.
- You must not move into squares with obstacles or beyond the grid boundaries.

Goal Collection Rules:
- You must pick up a goal only if it is the next in sequence and you are on the same square as that goal.
- Once a goal is picked up, the square it occupied becomes traversable.
- If you encounter a goal that is not the next in sequence, you cannot pick it up and must navigate to find the correct goal.

Coordinate System:
- The grid's origin is at the top left corner, with coordinates given as [row_number, column_number].
- Moving up decreases the row number, moving down increases the row number, moving left decreases the column number, and moving right increases the column number.

Action Specification:
- Specify your action using only one of the following commands within curly braces: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}.
- Do not include any additional words, symbols, or multiple actions within the braces.

Adaptive Learning and Error Correction:
- Learn from the outcome of each action to avoid ineffective or rule-violating moves.
- Continuously update your strategy based on your current position, the positions of remaining

goals, and the locations of obstacles.
- Avoid repeating a sequence of moves that does not change your state or bring you closer to the next goal.
- If an action does not progress towards the goal or violates the rules, reassess and choose a different action.

Action Planning and Efficiency:
- Before each move, verify your current position and assess the most efficient path to the next goal, avoiding obstacles and grid edges.
- If you are on the same square as the next goal, the only valid action is {Pick goal}.
- If the next goal is not directly accessible, plan an alternative route that brings you closer to the goal without violating movement rules.
- Prioritize picking up the goal over moving if you are on the goal square.

State Verification:
- Before suggesting an action, confirm your current position and the location of the next goal to ensure the action is valid and efficient.

Your ultimate goal is to collect all goals in the correct sequence as efficiently as possible, adhering strictly to the movement and collection rules.

Gridworld2 Best prompt for GPT-4
Score = 0.60 (GPT-4 as the testing LLM)

You (the robot) are tasked with navigating a grid-like field to sequentially collect goals, labeled from goal_0 to the highest-numbered goal, while avoiding obstacles. Each goal and obstacle occupies a distinct 1x1 square on the grid.

Your movements are limited to four directions: up, down, left, and right. You may only move onto a square if it is not occupied by an obstacle.

Critical Rule for Goal Collection: You must collect goals in strict numerical order, starting with goal_0. Before suggesting {Pick goal}, you must perform a state verification checkpoint. This involves confirming that the goal is the next in the numerical sequence and that you are on the correct square.

Adhere to these optimized rules for successful navigation and goal collection:

1. **Sequential Goal Collection**: Before suggesting {Pick goal}, explicitly state the number of the goal you are attempting to collect and confirm it is the next in the sequence. Do not attempt to collect a goal if it is not the correct one in the order.

2. **State and Position Awareness**: Continuously update your current position on the grid and the location of the next goal. Plan your moves to efficiently reach the next goal, avoiding obstacles and grid boundaries.

3. **Action Preconditions**: Only suggest Pick goal when you have verified that you are on the correct goal square and that the goal is the next in the sequence. Provide a clear justification for your action by stating your current position and the goal's position.

4. **Learning from Errors**: If an action is ineffective, analyze the outcome, learn from the mistake, and adjust your strategy to avoid repeating the error. State the reason for the error and the adjustment you will make.

5. **Obstacle and Boundary Consideration**: Plan moves that avoid obstacles and stay within the grid's boundaries to ensure a clear path to the next goal.

6. **Strategic Path Planning**: Choose the most direct and efficient path to the next goal, avoiding obstacles and boundaries. Re-evaluate your path after each move.

7. **Single Action Response**: Provide only one action in the specified format per response: {Action}.

8. **Adaptive Strategy**: As goals are collected and the grid's layout changes, adapt your strategy to ensure continuous progress towards the next goal in sequence.

9. **Avoiding Action Loops**: Recognize and break free from loops of non-productive actions by altering your approach. Implement a mechanism to detect repeated non-productive actions and change strategy if necessary.

10. **Feedback Utilization**: Use feedback from the environment and previous errors to inform your subsequent actions and improve your navigation strategy.

11. **Explicit Change of Strategy**: If a strategy is not leading to success, explicitly state and implement a new approach to find a path to the goal.

12. **Clear Movement Rules**: Adhere to the rules of movement and goal collection without ambiguity, ensuring that each action is deliberate and aligns with the goal sequence.

Before suggesting an action, confirm your current position, the location of the next goal, and the absence of obstacles in your path. Justify your action choice by referencing the goal sequence and your current position relative to the next goal. If an error occurs, analyze why it happened and adjust your strategy accordingly.

The coordinate system for the grid has its origin at the top left corner, with coordinates represented as [row_number, column_number]. For example, from [3,2], {Move up} results in [2,2], {Move down} in [4,2], {Move left} in [3,1], and {Move right} in [3,3].

Your possible actions are: {Move up}, {Move down}, {Move left}, {Move right}, {Pick goal}. Respond with only one of these actions, formatted as shown, at the end of each turn. Before taking an action, ensure it aligns with the goal sequence and the rules provided.

Blocksworld Human prompt
Score = 0.19 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.71 (GPT-4 as the testing LLM)

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block 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.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.
I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.
I can only unstack a block from on top of another block if the block I am unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being stacked.
I can only stack a block on top of another block if the block onto which I am stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.
Once you stack a block on top of a second block, the second block is no longer clear.

Please learn from previous steps. Not purely repeat the actions but learn why the state changes or remains in a dead loop. Avoid being stuck in action loops. Specify your action in this format at the end of your answer: pick up the {}, put down the {}, stack the {} on top of the {},unstack the {} from on top of the .

Blocksworld Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.6 (GPT-3.5-turbo-16k-0613 as the testing LLM)

I am tasked with arranging a set of blocks into specific configurations through a block-stacking activity. My available actions are:
- Pick up a block that is clear and on the table.
- Unstack a clear block from the top of another block.
- Put down a block onto the table, ensuring my hand is empty afterward.
- Stack a block onto another clear block, ensuring my hand is empty afterward.

To ensure successful completion of these actions, I must follow these rules:
1. I can only manipulate one block at a time.
2. My hand must be empty before I can pick up or unstack a block.
3. A block is considered clear and eligible to be picked up if it has no blocks on top of it, is on the table, and is not being held.
4. I can unstack a block only if it is the topmost block on another and there are no blocks above it.
5. When I pick up or unstack a block, I will be holding it.
6. I can only put down or stack a block that I am currently holding.
7. A block can be stacked onto another only if the bottom block is clear.
8. My hand must be empty before and after I place or stack a block.
9. Stacking a block on top of another makes the bottom block non-clear.

To optimize task execution and avoid errors, I will adhere to the following strategies:

- Conduct a comprehensive state verification before each action to ensure all preconditions are met: my hand is empty before picking up or unstacking; the block is clear, on the table, and not being held for picking up; and I am holding a block before putting down or stacking.
- Maintain an accurate and constantly updated mental model of the block arrangement, noting the clear status of each block, the current stack configurations, and whether my hand is empty or holding a block.
- Develop a strategic action plan that is directly aligned with achieving the desired final block configuration, taking into account the current state and the steps required to reach the goal.
- Integrate feedback after each action to assess the success of the action and to update my strategy, ensuring that I do not repeat ineffective actions and that I learn from any mistakes to avoid non-progressive loops.
- Communicate my intended actions clearly and precisely, using the format: 'pick up {color} block', 'put down {color} block', 'stack {color} block on top of {color} block', 'unstack {color} block from on top of {color} block'.
- Implement an enhanced loop detection mechanism to identify and interrupt any repetitive, non-progressive action sequences, choosing a different action if necessary.
- Set and pursue intermediate goals that are necessary steps towards the final configuration, ensuring that each action is deliberate and contributes to the end goal in an incremental fashion.
- Establish a timeout or step limit to prevent exceeding the query time limit without completing the task, and reassess my strategy if progress stalls to ensure that I am always moving towards task completion.
- Explicitly state the preconditions that have been verified before proposing an action, and clearly communicate any adjustments made to the strategy based on feedback received.
- Introduce a robust error handling strategy that allows for backtracking or reassessment of the plan when an action fails, ensuring alternative actions adhere to the rules and contribute to the final goal.

By following these refined guidelines and continuously updating my approach based on the state of the blocks and the feedback received, I aim to efficiently and effectively complete the block-stacking task.

Blocksworld Best prompt for GPT-4
Score = 0.95 (GPT-4 as the testing LLM)

To effectively arrange a set of blocks into the desired stacks, adhere to the following structured approach, which has been refined based on previous feedback and identified errors:

1. **Evaluate the Goal State**: Examine the goal state configuration in detail and compare it with the current state to discern the exact actions required to achieve the goal. Maintain a clear and constant visualization of the final desired arrangement of blocks throughout the task.

2. **Action Sequence Planning**: Construct a strategic plan that delineates a sequence of actions that will methodically transition the current state towards the goal state. Prioritize actions that make definitive progress towards the goal and eliminate redundant or non-contributory steps.

3. **Preconditions Verification**: Before initiating any action, rigorously check that all preconditions are satisfied. Confirm that your hand is empty before attempting to pick up or unstack a block, and ensure that the block to be manipulated is unobstructed and either on the table or atop another block.

4. **Execute Actions**: Implement the necessary actions, strictly following the prescribed format and constraints:
- To pick up a block: 'pick up the {color} block.'
- To unstack a block: 'unstack the {color} block from on top of the {color} block.'
- To put down a block: 'put down the {color} block.'
- To stack a block: 'stack the {color} block on top of the {color} block.'

5. **Loop and Error Prevention**: Vigilantly observe your actions to identify any repetitive or non-productive patterns. Upon detecting a loop, promptly reassess and revise the action plan. Document past errors to prevent their recurrence.

6. **State Change Analysis**: After executing an action, conduct a state change analysis to verify that the system is incrementally closer to the goal state. If the action does not yield the expected progress, reevaluate and modify the plan.

7. **Continuous Learning**: Log the results of previous actions, noting both successes and failures, to refine future strategies and enhance task efficiency.

8. **Clear Goal Specification**: Keep the goal state at the forefront of your strategy, ensuring that every action is intentionally aimed at achieving that state.

9. **Feedback Integration**: After each action, incorporate feedback to improve your understanding of the current state and to guide future actions.

10. **Loop Detection and Correction**: Establish a robust mechanism to detect when you are in a loop and to prompt a strategic reassessment of the action plan.

11. **Goal State Reassessment**: Frequently reevaluate both the goal state and the current state to confirm that your actions are consistently aligned with the goal.

12. **Action Format Standardization**: Adhere to the specified action format with precision, refraining from adding prefixes or narrative explanations unless the context demands it.

13. **State Change Verification**: Post-action, ensure that the state has altered as intended and that the system is nearer to the goal state.

14. **Error Handling**: Enhance error handling protocols to avert the repetition of unsuccessful actions.

15. **Optimize Query Time**: Employ methods to expedite the planning and execution of actions, aiming for task completion with optimal efficiency.

This refined approach is designed to systematically guide you towards arranging the blocks into the goal state configuration while minimizing errors and enhancing task performance.

Logistics Human prompt
Score = 0.083 (GPT-3.5-turbo-16k-0613 as the testing LLM)
Score = 0.50 (GPT-4 as the testing LLM)

You have to plan logistics to transport packages within cities via trucks and between cities via airplanes. Locations within a city are directly connected (trucks can move between any two such locations), and so are the cities. In each city there is exactly one truck and each city has one location that serves as an airport.

Here are the actions that can be performed:
Load a package into a truck at a location.
Load a package into an airplane at a location.
Unload a package from a truck at a location.
Unload a package from an airplane at a location.
Drive a truck from one location to another location within a city.
Fly an airplane from one location in a city to another location in another city.

The following are the restrictions on the actions:
A package can be loaded into a truck only if the package and the truck are in the same location.
Once a package is loaded into a truck, the package is not at the location and is in the truck.
A package can be loaded into an airplane only if the package and the airplane are in the same location.
Once a package is loaded into an airplane, the package is not at the location and is in the airplane.
A package can be unloaded from a truck only if the package is in the truck.
Once a package is unloaded from a truck, the package is not in the truck and is at the location of the truck.
A package can be unloaded from an airplane only if the package in the airplane. Once a package is unloaded from an airplane, the package is not in the airplane and is at the location of the airplane.
A truck can be driven from one location to another if the truck is at the from-location and both from-location and to-location are locations in the same city. Once a truck is driven from one location to another, it is not at the from-location and is at the to-location.
An airplane can be flown from one city to another if the from-location and the to-location are airports and the airplane is at the from-location.
Once an airplane is flown from one city to another the airplane is not at the from-location and is at the to-location.

Please learn from previous steps. Not purely repeat the actions but learn why the state changes or remains in a dead loop. Avoid being stuck in action loops. Specify your action in this format at the end of your answer: load {} into {} at {}, unload {} from {} at {}, drive {} from {} to {} in {}, fly {} from {} to {}.

Logistics Best prompt for GPT-3.5-turbo-16k-0613
Score = 0.18 (GPT-3.5-turbo-16k-0613 as the testing LLM)

To optimize the logistics of transporting packages within cities using trucks and between cities using airplanes, follow these enhanced and precise guidelines:

1. **Loading and Unloading Preconditions:**
- Load a package into a truck only when the package and the truck are co-located.
- Load a package into an airplane only at an airport, ensuring both the package and the airplane are present.
- Unload a package from a truck only if it has been verified that the package is in that truck.
- Unload a package from an airplane only if it has been verified that the package is in that airplane.

2. **Movement Rules:**
- Trucks are restricted to travel within their respective city limits.
- Airplanes must fly between airports in different cities without exception.

3. **State Changes:**
- Reflect the package's new location as inside the vehicle upon loading and at the vehicle's location upon unloading.

4. **Action Format:**
- Actions must be articulated as follows:
- For loading/unloading: 'load {package} into {vehicle} at {location}' or 'unload {package} from {vehicle} at {location}'
- For driving: 'drive {truck} from {from-location} to {to-location} in {city}'
- For flying: 'fly {airplane} from {from-airport} to {to-airport}'

5. **Feedback and Learning:**
- Update the state of packages, trucks, and airplanes with each action taken.
- Log unsuccessful actions due to precondition failures and avoid their repetition.
- Refine plans based on feedback to ensure all actions are valid and goal-aligned.

6. **Goal-Oriented Strategy:**
- Actions must form a logical sequence that advances a package towards its destination in the most direct manner possible.

7. **Avoiding Loops:**
- Exclude any action that has been attempted unsuccessfully.
- Keep a comprehensive log of actions to identify and prevent cyclical patterns, revising the strategy as needed.

8. **Task Decomposition:**
- Segment the task into discrete sub-tasks, such as intra-city and inter-city package transfers.
- Tackle each sub-task systematically, one at a time.

9. **Time Management:**
- Streamline the planning process to ensure task completion within a set timeframe.
- Give precedence to actions that maximize time efficiency while complying with the above guidelines.

By adhering to these updated guidelines, you will devise a logistics plan that is both accurate and efficient, guaranteeing the successful delivery of packages to their designated locations.

Logistics Best prompt for GPT-4
Score = 0.74 (GPT-4 as the testing LLM)

Your task is to manage the logistics of transporting packages within and between cities using trucks and airplanes. Each city has a network of locations for truck movement and an airport for airplane transfers. There is one truck per city for local deliveries and one airport per city for intercity transfers.

To enhance logistics operations and avoid errors, follow these optimized steps:

1. **State Verification**: Prior to any action, rigorously confirm the current locations of all packages, trucks, and airplanes. This step is crucial to ensure that all subsequent actions are based on the most recent and accurate state information.

2. **Action Execution**: Execute actions strictly adhering to these preconditions:
- Load a package into a truck at a location only if the package and the truck are confirmed to be at that location.
- Load a package into an airplane at an airport only if the package and the airplane are confirmed to be at that airport.
- Unload a package from a truck at a location only if the package is confirmed to be in that truck.
- Unload a package from an airplane at an airport only if the package is confirmed to be in that airplane.
- Drive a truck from one location to another within the same city only if the truck's presence at the starting location is confirmed.
- Fly an airplane from one city's airport to another city's airport only if the airplane's presence at the starting airport is confirmed.

3. **State Update**: Immediately after each action, update the environment state to reflect the new locations of packages, trucks, and airplanes. This updated state must be used for verifying preconditions for the next actions.

4. **Efficient Planning**: Deliver all packages to their destinations using the fewest actions possible. Prioritize the shortest routes and avoid any actions that do not directly contribute to reaching the delivery goals.

5. **Adaptive Learning**: Utilize feedback from the outcomes of previous actions to continuously refine planning strategies. Avoid repeating ineffective actions and adjust plans based on the latest state information and feedback.

6. **Error Management**: If an action fails, quickly reassess the situation based on the current state and propose a new, valid action that moves towards the delivery goals.

7. **Clear Action Formatting**: Clearly express actions using the specified structure to avoid misunderstandings:

- load {package} into {truck/airplane} at {location/airport}
- unload {package} from {truck/airplane} at {location/airport}
- drive {truck} from {location} to {location} in {city}
- fly {airplane} from {airport} to {airport}

8. **Goal-Focused Actions**: Ensure every action is purposeful and directly contributes to the final destination of the packages. Eliminate any actions that are not goal-oriented.

9. **Time-Efficient Queries**: Streamline the planning process to complete tasks within the query time limit, maintaining a balance between swift operations and careful action validation.

10. **Simplified Instructions**: Provide instructions that are clear, concise, and easy to follow, ensuring they are understood and executed correctly.

By diligently following these optimized guidelines, you will significantly improve the efficiency and accuracy of the logistics operation for package delivery.