PGPO: Enhancing Agent Reasoning via Pseudocode-style Planning Guided Preference Optimization

Zouying Cao[1,3,4,†], **Runze Wang**[2], **Yifei Yang**[1,3,4], **Xinbei Ma**[1,3,4], **Xiaoyong Zhu**[2], **Bo Zheng**[2,*], **Hai Zhao**[1,*]
[1]School of Computer Science, Shanghai Jiao Tong University,
[2]Taobao & Tmail Group of Alibaba, [3]Key Laboratory of Shanghai Education Commission for Intelligent Interaction and Cognitive Engineering, Shanghai Jiao Tong University,
[4]Shanghai Key Laboratory of Trusted Data Circulation and Governance in Web3
zouyingcao@sjtu.edu.cn, yunze.wrz@alibaba-inc.com

Abstract

Large Language Model (LLM) agents have demonstrated impressive capabilities in handling complex interactive problems. Existing LLM agents mainly generate natural language plans to guide reasoning, which is verbose and inefficient. NL plans are also tailored to specific tasks and restrict agents' ability to generalize across similar tasks. To this end, we explore pseudocode-style plans (P-code Plan) to capture the structural logic of reasoning. We find that P-code Plan empowers LLM agents with stronger generalization ability and more efficiency. Inspired by this finding, we propose a pseudocode-style Planning Guided Preference Optimization method called PGPO for effective agent learning. With two planning-oriented rewards, PGPO further enhances LLM agents' ability to generate high-quality P-code Plans and subsequent reasoning. Experiments show that PGPO achieves superior performance on representative agent benchmarks and outperforms the current leading baselines. Analyses reveal the advantage of PGPO in reducing action errors and omissions during reasoning.[1]

1 Introduction

Recent advances in large language models have promoted the development of LLM agents (Wang et al., 2024a; Xi et al., 2025). Planning serves as a critical component of agent reasoning (Huang et al., 2024), allowing agents to break down complex problems into manageable subtasks. By prompting strategies (Wang et al., 2023; Prasad et al., 2024; Roy et al., 2024), task-specific fine-tuning (Qiao et al., 2024a,b) or external classical planners (Liu et al., 2023a; Arora et al., 2024), LLM agents are equipped with basic planning abilities. Due to the dominance of natural language (NL) in agent reasoning (Wei et al., 2022; Yao et al., 2023), existing researches mainly focus on generating NL plans. However, semantic ambiguities and undesired verbosity in NL may not be beneficial to agent planning, leading to low precision and inefficiency. Meanwhile, NL plans are too specific to help LLM agents generalize to other unseen yet similar tasks. Therefore, it still remains underexplored whether alternative plan formats could elicit more efficient and generalized LLM agents.

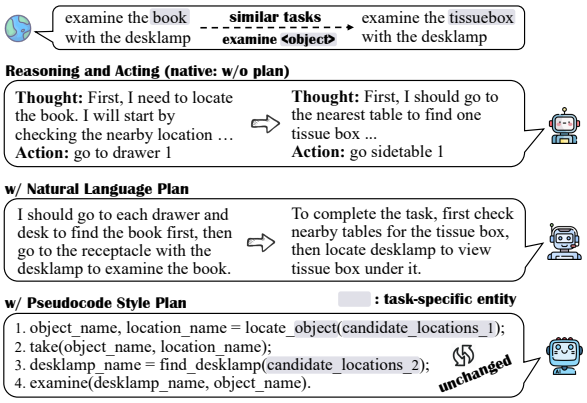


Figure 1: An example demonstrating why P-code Plan helps LLM agents generalize well. When faced with similar tasks (e.g., examine object with desklamp), the thought process can be recycled through P-code Plans.

Prior work (Wang et al., 2024c) demonstrates the advantage of executable code as agent's action over text or JSON format. Inspired by this, we explore using pseudocode to represent plan, given that plans can be considered as high-level abstraction of actions while pseudocode outlines code. Our work starts by distilling pseudocode-style plans (denoted as P-code Plan) from existing ReAct-style (Yao et al., 2023) datasets, adhering to predefined format requirements. We observe that through fine-tuning with P-code Plan incorporated, LLM agents exhibit improved out-of-distribution generalization. As shown in Figure 1, abstract plan-

[†]Work done during an internship at Alibaba Group.
[*]Corresponding authors. This research was supported by the Joint Research Project of Yangtze River Delta Science and Technology Innovation Community (No. 2022CSJGG1400), Alibaba Group through Alibaba Innovative Research Program.
[1]https://github.com/zouyingcao/PGPO.

Findings of the Association for Computational Linguistics: ACL 2025, pages 14966–14985
July 27 - August 1, 2025

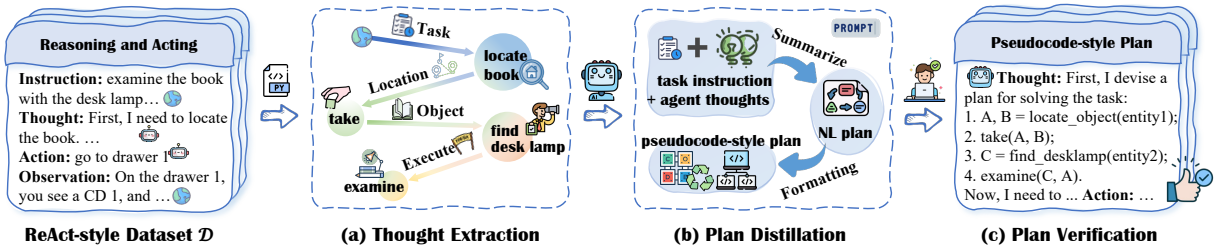


Figure 2: Overview of P-code Plan generation pipeline. We first extract the thought part from existing ReAct-style datasets. Then, we prompt GPT-4o to summarize the thought process into high-level plans. Pseudocode-style plans are finally structured with predefined formats, followed by manual verification to ensure accuracy.

ning steps in P-code Plan can capture generalizable task knowledge while NL plans focus on specific knowledge and may suffer from overfitting. Moreover, agent's planning ability learned from concise P-code Plan facilitates a more efficient reasoning process with fewer interactions.

Building upon these insights, we further propose a pseudocode-style Planning Guided Preference Optimization method named PGPO for agent capability enhancement. Specifically, we first utilize supervised fine-tuning to build a base agent. Then, PGPO contains two iterative phases: (1) the base agent performs exploration on expert trajectories to construct contrastive trajectory datasets based on two designed planning-oriented rewards; (2) direct preference optimization is employed to refine the base agent's pseudocode-style planning ability for task guidance. Through experiments with four LLMs, PGPO outperforms various strong baselines by relative 11.6% performance gain averaged across three representative agent benchmarks. In summary, our contributions are as follows:

- We investigate the effectiveness of pseudocode-style plans in agent reasoning, which are more concise and structured than NL plans. P-code Plan demonstrates its superiority in boosting the generalization ability of LLM agents.
- We further introduce PGPO, a preference optimization method that empowers LLM agents with enhanced reasoning capabilities under the guidance of pseudocode-style plans.
- Experimental results reveal that our PGPO achieves state-of-the-art performance on representative agent benchmarks, especially when dealing with more complex agent tasks.

2 Pseudocode-style Plan is Beneficial

In this section, we first define the structural representation of P-code Plan and then introduce a plan generation pipeline, followed by preliminary experiments to show the advantage of P-code Plan.

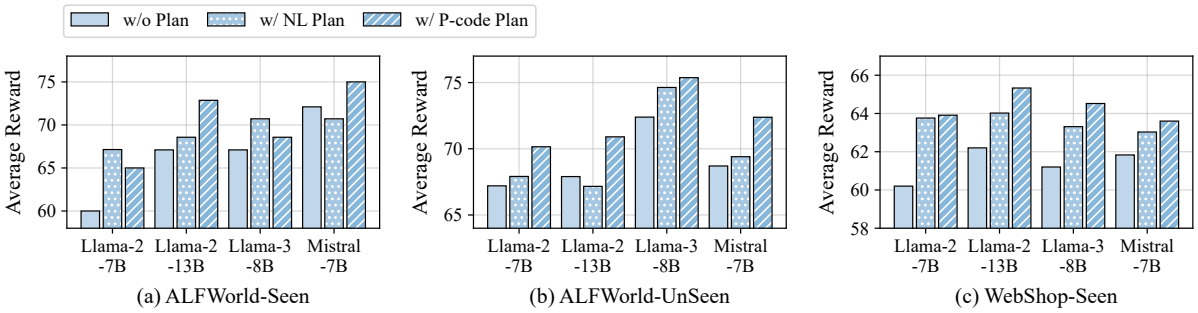
2.1 Definition of P-code Plan

In this paper, we mainly focus on LLM agents' multi-step reasoning usage: LLM agents interact with the environment to accomplish complex tasks, which can be represented as a set of Thought-Action-Observation tuples $\{(t, a, o)\}_n$. At each interaction, the LLM agent gives the inner thoughts t and takes an action a based on the observation o from the environment. n denotes the number of interaction turns. Following this task formalization, we define the format of our pseudocode-style plans.

Planning Step. $P_s = (id, name, [parameter], [return\ value], [control\ flow])$ structures each step in the plan, uniquely identified by id. Similar to function in programming languages, one planning step usually corresponds to a subset of actions oriented to one subtask. $name$ abstracts a function identifier to describe the reasoning process of this subtask, with $parameters$ enclosed in parentheses. Square brackets ($[...]$) mean optional attributes. If necessary, $return\ values$ are indicated on the left of the assignment operator, standing for the observed information. $control\ flow$ signifies standard programming structures such as *if-else* and *for*, which can be omitted when planning steps are performed sequentially.

Planning Entity. $E = \{e_1, e_2, ..., e_m\}$ refs to the prior knowledge entities used to specify some $parameters$ in the planning step. These entities serve as guidance for generating valid actions and avoiding aimless exploration.

P-code Plan. Denoted as (P_s, E), pseudocode-style plans are the combination of abstract planning steps and task-specific planning entities. Different from NL plans, P-code Plans are more structured



(a) ALFWorld-Seen (b) ALFWorld-UnSeen (c) WebShop-Seen

Figure 3: Comparison between *w/ P-code Plan* and *w/o Plan*, *w/ NL Plan* during the SFT process for LLM agents. Here *w/o Plan* symbolizes training on the original ReAct-style datasets, *w/ NL Plan* indicates incorporating natural language plans into training data and *w/ P-code Plan* represents the incorporation of pseudocode-style plans.

and concise. This format helps agent better generalize to unseen tasks[2], where unseen tasks may share similar planning steps with seen tasks but initialize different planning entities.

2.2 Plan Generation Pipeline

Initially, we have ReAct-style dataset $\mathcal{D}$ where each instance d consists of one task instruction u with its collected expert trajectory $\tau = (t_1, a_1, o_1, ..., o_{n-1}, t_n, a_n, o_n)$. Then, with LLM and human participation, as illustrated in Figure 2, our P-code Plan generation pipeline is outlined as follows:

- **Thought Extraction.** We first extract agent thoughts $\{t\}_n$ from expert trajectories, which involves task-specific knowledge for planning. The relationship between agent thoughts and task plans can be viewed as one of abstraction. The plan distills the essentials of thoughts while omitting specifics, as an abstract summary does.
- **Plan Distillation.** Subsequently, to improve the quality of distilled plans and obtain more accurate abstract knowledge, we employ a powerful model (e.g., GPT-4o) for generation. Given the task instruction u with agent thoughts $\{t\}_n$, we instruct the LLM to summarize the step-by-step plan in natural language. Next, due to the effectiveness of few-shot prompting strategy for structural generation (Valmeekam et al., 2024; Liu et al., 2023a), demonstrations of pseudocode-style plans paired with corresponding tasks are taken as input, converting natural language plans into P-code Plans. See Appendix B.2 for the prompt we used to guide plan distillation.
- **Plan Verification.** Last, we choose human to verify the LLM-generated P-code Plans whether follow the format requirements described in Section 2.1. In order to guarantee accuracy and knowledge consistency, minor manual refinement are also needed.

[2]The term "unseen tasks" refs to task scenarios that are not present in the training set, which can measure out-of-distribution generalization.

Setting	Llama-2 -7B	Llama-2 -13B	Llama-3 -8B	Mistral -7B
ALFWorld-Seen				
w/o Plan	12.04	11.38	11.49	11.45
w/ NL Plan	11.34	11.36	11.41	12.14
w/ P-code Plan	**11.33**	**11.21**	**11.29**	**10.99**
ALFWorld-Unseen				
w/o Plan	12.93	12.22	12.59	12.84
w/ NL Plan	12.13	12.27	11.88	12.33
w/ P-code Plan	**12.04**	**11.87**	**11.78**	**11.57**

Table 1: Average interaction turns required on ALFWorld. **Bold** indicates the best results of each model.

2.3 P-code Plan Improves Generalization

We adopt supervised fine-tuning (SFT) to equip base LLMs with agent abilities and then investigate whether LLM agents can benefit from our designed pseudocode-style plans. Our experiments are based on four popular LLMs: Llama-2-7B/13B (Touvron et al., 2023), Llama-3-8B (Dubey et al., 2024) and Mistral-7B-v0.3 (Jiang et al., 2023). For the agent tasks, we choose two representative datasets: ALFWorld (Shridhar et al., 2021) and WebShop (Yao et al., 2022). We employ the average reward as metric to reflect the agent performance. Note that ALFWorld comprises both seen and unseen test sets in order to evaluate the in-distribution and out-of-distribution generalization of the agents. Please ref to Appendix B.1 and B.4 for more details.

We first collect ReAct-style training expert trajectories following Xiong et al. (2024) and then use the above plan generation pipeline (Section 2.2) to

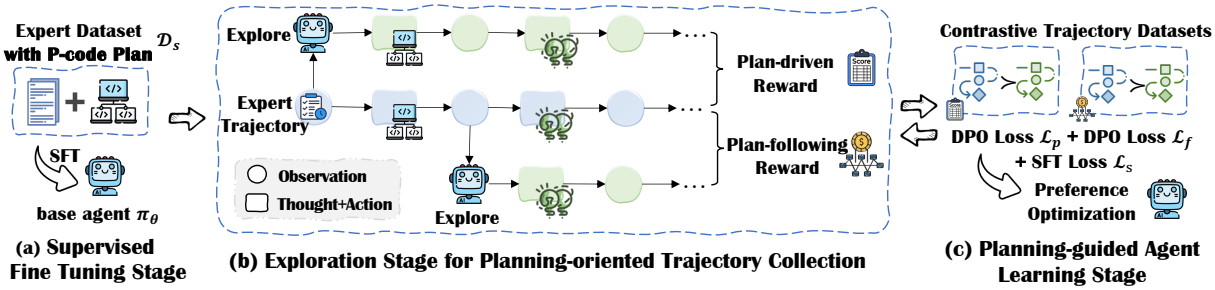


Figure 4: The overview of PGPO. Our algorithm starts by build a SFT-based agent. Then, the base agent iteratively performs exploration on expert trajectories to construct contrastive trajectory datasets based on two designed planning-oriented rewards and updates itself via preference optimization to enhance agent capabilities.

obtain P-code Plans, along with natural language plans. These two type generated plans are finally incorporated into the first step of original trajectories to construct new training datasets, respectively. In Appendix B.4, we present more details of dataset construction procedure with some examples.

As shown in Figure 3, with the help of P-code Plan, LLM agents generally have higher average reward (10 out of 12 scenarios). Compared to naive expert trajectories (*w/o Plan*), integrating plans into training data (regardless of format) empowers LLM agents with basic planning ability, which is beneficial to the following agent reasoning. Regarding planning format, although *w/ P-code Plan* brings weaker performance than *w/ NL Plan* for Llama-2-7B and Llama-3-8B on the seen ALFWorld tasks (2 out of 12 scenarios), it consistently achieves better generalization to unseen ALFWorld tasks for all four models. We consider this can be attributed that abstract planning steps in P-code Plan capture generalizable meta-knowledge guiding task solution but natural language plans are prone to overfitting. We also analyze whether executable code format could lead to higher performance (in Appendix C.2) and find verbose plans pose a greater challenge for generation, thereby negatively affect reasoning.

Moreover, we further calculate the average number of interaction turns on all evaluated instances. In Table 1, LLM agents *w/ P-code Plan* surprisingly reduce the average interaction turns compared to the other two settings. This underscores the superiority of P-code Plan in preventing blind exploration when dealing with agent tasks. Another interesting finding is handling unseen tasks requires more interactions than seen tasks, but the increase from LLM agents *w/ P-code Plan* is relatively low. This phenomenon also demonstrates our designed P-code Plan is suitable for agent generalization.

Overall, the advantage of P-code Plans over NL plans can be summarized into two aspects: (1) abstract pseudocode format helps LLM agents better generalize to unseen tasks; (2) concise and structured pseudocode facilitates a more efficient agent reasoning process with fewer interactions.

3 P-code Plan-Guided Agent Learning

Despite widespread use in open LLM agents (Chen et al., 2023; Zeng et al., 2024; Yin et al., 2024), supervised fine-tuning approach has its drawback, that is, the limited generalization ability due to over-reliance on expert trajectories (Song et al., 2024; Fu et al., 2025). Recent works focus on Direct Preference Optimization (DPO) (Rafailov et al., 2024) and its variants to develop LLM agents with contrastive trajectory dataset (Yang et al., 2024b; Song et al., 2024; Xiong et al., 2024; Shi et al., 2024). Inspired by our findings in Section 2.3, we propose a method for improving agents' pseudocode-style planning ability, called PGPO, standing for **P**lanning **G**uided **P**reference **O**ptimization. The overview of our method is depicted in Figure 4.

3.1 Planning-oriented Trajectory Collection

We start off training the base agent π_θ via SFT on the expert dataset $\mathcal{D}_s = \{(u, p, \tau)^{(i)}\}_{i=1}^{|\mathcal{D}_s|}$ with P-code Plans p generated in Section 2.2. We denote new plan-incorporated trajectory as $\tau' = (p, t_1, a_1, o_1, ..., o_{n-1}, t_n, a_n, o_n)$ and the loss function can be formulated as:

$$\mathcal{L}_{SFT}(\theta) = -\mathbb{E}_{(u,\tau')\sim\mathcal{D}_s}\{\sum_{j=1}^{n}[\log \pi_\theta(t_j|u, p, \tau_{j-1}) + \log \pi_\theta(a_j|u, p, \tau_{j-1}, t_j)] + \log \pi_\theta(p|u)\}, \quad (1)$$

where $\tau_{j-1} = (t_1, a_1, o_1, ..., a_{j-1}, o_{j-1})$ represents the interaction history of previous $j-1$ rounds.

The obtained base agent $\pi_{\theta_{base}}$ is used for exploration on the expert trajectory to collect contrastive

action pairs. Oriented towards planning, we design two rewards for contrastive trajectory construction: 1) plan-driven reward; 2) plan-following reward.

Plan-driven Reward r_d evaluates the influence of P-code Plans on the entire trajectory. Given each task instruction u, we use base agent to generate the P-code Plan $\hat{p} \sim \pi_{\theta_{base}}(\cdot|u)$ and subsequent reasoning steps $\hat{\tau} \sim \pi_{\theta_{base}}(\cdot|u, \hat{p})$. In our experiments, agent stops exploration when task completes or the maximum number of interaction rounds is exceeded. Then, the environment will give the outcome reward r_o, which is positively correlated with the quality of exploration trajectory. For simplicity, we use this outcome reward as r_d. By comparing $r_d(p, \tau)$ and $r_d(\hat{p}, \hat{\tau})$ for expert trajectory and exploration trajectory, we get our first contrastive trajectory dataset $D_p = \left\{(u, p^w, \tau^w, p^l, \tau^l)^{(i)}\right\}_{i=1}^{|D_p|}$. We use $(p^w, \tau^w) \succ (p^l, \tau^l) \mid u$ to represent the situation where (p^w, τ^w) with higher reward is preferred over (p^l, τ^l) with lower reward.

Plan-following Reward r_f assesses the extent to which agents comprehend the structured intentions behind P-code Plans and their proficiency in following these plans during task execution. Given the first expert interaction round (u, p, τ_1) as historical trajectory, we query base agent to continue exploration $\hat{\tau}_{2:m} \sim \pi_{\theta_{base}}(\cdot|u, p, \tau_1)$. m denotes the total interaction rounds. Here, we select the first two interaction rounds $\hat{\tau}_2$ as representative to analyze the alignment between plans and executed actions. Inspired by Monto Carlo Tree Search (Kocsis and Szepesvári, 2006), we quantify it as the potential to complete task successfully. It is intuitive that when the quality of P-code Plan is guaranteed, subsequent reasoning steps that closely adhere to the plan are likely to yield higher outcome rewards. Therefore, we use one scorer agent to generate new subsequent trajectory $\tau^s_{3:m'} \sim \pi_{\theta_{scorer}}(\cdot|u, p, \hat{\tau}_2)$. By sampling N new trajectories, the average outcome reward r_o from the environment estimates the plan-following reward:

$$r_f(u, p, \hat{\tau}_2) = \frac{\sum_{i=1}^{N} r_o(\tau^s_{3:m'}|u, p, \hat{\tau}_2)^{(i)}}{N} \quad (2)$$

In our approach, we use $\pi_{\theta_{base}}$ as $\pi_{\theta_{scorer}}$. Then, similar to the construction of D_p, we contrast subsequent trajectory $\tau^w_{2:n} \succ \tau^l_{2:m} \mid (u, p, \tau_1)$ based on $r_f(u, p, \tau_2)$ and $r_f(u, p, \hat{\tau}_2)$, establishing our second contrastive trajectory dataset $D_f = \left\{(u, p, \tau_1, \tau^w_{2:n}, \tau^l_{2:m})^{(i)}\right\}_{i=1}^{|D_f|}$.

3.2 Planning-guided Agent Learning

After collecting preference data, DPO method is utilized to optimize our base agent. First, based on dataset D_p, agent learns to generate high-quality P-code Plans along with subsequent reasoning by minimizing the following loss:

$$\mathcal{L}_p = -\mathbb{E}_{\substack{(u,p^w,\tau^w,p^l,\tau^l)\\ \sim \mathcal{D}_p}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(p^w, \tau^w|u)}{\pi_{ref}(p^w, \tau^w|u)} - \beta \log \frac{\pi_\theta(p^l, \tau^l|u)}{\pi_{ref}(p^l, \tau^l|u)}\right)\right] \quad (3)$$

where σ denotes the logistic function, β controls the weight of the preference for the reference model π_{ref}. Meanwhile, agent refines its parameters to develop the plan-following ability gathered from dataset D_f, which can be formulated as:

$$\mathcal{L}_f = -\mathbb{E}_{\substack{(u,p,\tau_1,\tau^w_{2:n},\tau^l_{2:m})\\ \sim \mathcal{D}_f}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(\tau^w_{2:n}|u, p, \tau_1)}{\pi_{ref}(\tau^w_{2:n}|u, p, \tau_1)} - \beta \log \frac{\pi_\theta(\tau^l_{2:m}|u, p, \tau_1)}{\pi_{ref}(\tau^l_{2:m}|u, p, \tau_1)}\right)\right] \quad (4)$$

One issue of standard DPO is that log probability of chosen trajectories may decrease over training steps, leading to sub-optimal performance. Following previous works (Pang et al., 2024; Xiong et al., 2024), we add SFT loss to mitigate this issue:

$$\mathcal{L}_s = -\mathbb{E}_{(u,p^w,\tau^w,p^l,\tau^l)\sim \mathcal{D}_p}\left[\log \pi_\theta(p^w, \tau^w|u)\right] \quad (5)$$

Finally, the optimization objective of PGPO is:

$$\min_{\pi_\theta} \left(\mathcal{L}_p + \mathcal{L}_f + \mathcal{L}_s\right) \quad (6)$$

The updated agent will be used as new base agent for exploration and iterate the above learning process until exceeding the maximum iterations. The overall procedure of PGPO is summarized in Appendix A Algorithm 1.

4 Experiments

4.1 Experimental Settings

Datasets and Metrics. Besides **ALFWorld** for embodied household tasks and **WebShop** for online shopping (as described in Section 2.3), we also include one game benchmark **TextCraft** (Prasad et al., 2024) for crafting Minecraft items. ALFWorld and TexCraft provide binary rewards to indicate task success while WebShop provides dense rewards from 0 to 1 to measure the task completion level. For all datasets, we choose the average reward as evaluation metric. Detailed statistical information of the datasets are in Appendix B.1.

Method	Llama-2-7B					Llama-2-13B				
	ALFWorld		WebShop	TextCraft	Avg.	ALFWorld		WebShop	TextCraft	Avg.
	Seen	UnSeen				Seen	UnSeen			
SFT	60.0	67.2	60.2	28.0	53.9	67.1	67.9	62.2	29.0	56.6
ETO	68.6	72.4	67.4	35.0	60.9	75.0	69.4	68.9	42.0	63.8
IPR	70.3	74.7	71.3	34.0	62.6	75.0	76.9	72.2	39.0	65.8
PGPO	**76.4**	**76.9**	**72.2**	**43.0**	**67.1**	**77.1**	**77.6**	**73.7**	**48.0**	**69.1**
Method	**Llama-3-8B**					**Mistral-7B**				
	ALFWorld		WebShop	TextCraft	Avg.	ALFWorld		WebShop	TextCraft	Avg.
	Seen	UnSeen				Seen	UnSeen			
SFT	67.1	72.4	61.2	20.0	55.2	72.1	68.7	61.8	31.0	58.4
ETO	72.1	73.1	66.2	36.0	61.9	**75.0**	72.4	66.2	38.0	62.9
IPR	72.9	73.9	72.0	38.0	64.2	73.6	73.1	**69.6**	36.0	63.1
PGPO	**75.0**	**76.9**	**72.3**	**46.0**	**67.6**	**75.0**	**77.6**	69.0	**45.0**	**66.7**

Table 2: Main results of PGPO compared to training-based baselines on ALFWorld, WebShop and TextCraft. **Bold** and underline indicate the best and the second-best results of each model. For all methods (except SFT), we report the best performance across all iterations following Xiong et al. (2024). Our PGPO is evaluated in zero-shot setting.

Method	ALFWorld		WebShop	TextCraft
	Seen	UnSeen		
ReAct+GPT-4	42.9	38.1	63.2	28.0
ReAct+GPT-3.5	7.9	10.5	62.4	20.0
ADaPT+GPT-4	75.0	69.4	64.8	**48.0**
ADaPT+GPT-3.5	70.3	71.6	62.7	26.0
PGPO+Llama-2-7B	**76.4**	**76.9**	72.2	43.0
PGPO+Llama-3-8B	75.0	**76.9**	**72.3**	46.0

Table 3: Comparative experiments on PGPO vs. prompt-based baselines. The best and second-best results are marked in **bold** and underline.

	ALFWorld		WebShop	TextCraft
	Seen	UnSeen		
PGPO	76.4	76.9	72.2	43.0
- *P-code*	75.7↓ 0.7	71.6↓ 5.3	69.6↓ 2.6	40.0↓ 3.0
- L_f	74.3↓ 2.1	75.4↓ 1.5	70.4↓ 1.8	41.0↓ 2.0
- L_s	69.3↓ 7.1	68.7↓ 8.2	64.8↓ 7.4	35.0↓ 8.0

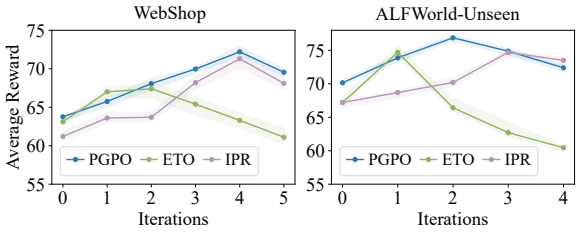
Table 4: Approach ablations of PGPO. Experiments are based on Llama-2-7B. - *P-code* represents using NL plans to replace P-code Plans. - L_f denotes leaving out the estimation of plan-following reward r_f, followed by the removal of L_f. - L_s means the removal of SFT loss.

Baselines. We compare PGPO with naive SFT and other two leading agent learning methods: (1) ETO (Song et al., 2024) applies DPO loss to improve the agent from its exploration failures; (2) IPR (Xiong et al., 2024) introduces step-wise process supervision into LLM agent training. For fair comparison, we also select Llama-2-7B/13B, Llama-3-8B and Mistral-7B-v0.1 as backbone models and use the same training data. Additionally, we include two strong closed-source LLMs: GPT-3.5-Turbo and GPT-4, utilizing two prompt-based methods ReAct (Yao et al., 2023) and ADaPT (Prasad et al., 2024) for comparison.

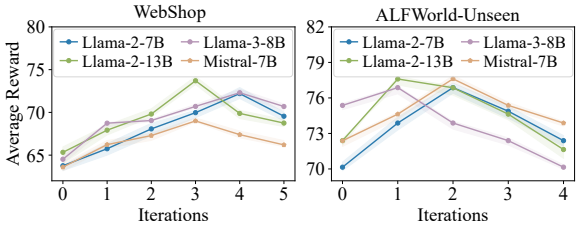
Implementation Details. During the SFT phase, we set the learning rate as 2e-5 and the batch size as 48 across 3 training epoches. The cosine scheduler is employed with a 3% warm-up ratio. In the trajectory collection stage, the base agent explores the environment using a temperature of 0. To construct contrastive pairs, we sample N=5 times with temperature=1 to calculate the plan-following reward. During the following preference optimization phase, we tune the learning rate from 5e-7 to 5e-6 and test two values for β in the DPO loss: 0.01 and 0.1. The maximum number of iterations is set to 4. All training experiments are conducted on 8 NVIDIA A100 80 GB GPUs. See Appendix B.4-B.6 for more details.

4.2 Main Results

Table 2 and 3 show the evaluation results of PGPO and baselines on three agent benchmarks. First, compared with training-based baselines, PGPO significantly increases the average reward across all the datasets. Specifically, PGPO with Llama-2-7B surpasses the state-of-the-art method IPR by an improvement of 7.2% on average reward. This indicates the incorporation of P-code Plans into training data can provide the model with enhanced reasoning abilities in accomplishing the agent tasks.



(a) Comparison among Different Methods



(b) Influence on PGPO across Different Models

Figure 5: Ablation study on optimization iterations. (a) provides a comparison of the performance of PGPO against ETO and IPR across varying iterations. (b) shows the influence of increasing iterations on PGPO across different base models. iter=0 is the SFT stage. Shaded regions indicate standard error across 5 trails.

Second, for prompt-based baselines, traditional ReAct paradigm using GPT-3.5-Turbo exposes poor performance on all agent datasets. Although ADaPT+GPT-3.5 gains a performance boost via recursive task decomposition, it still underperforms our method. In particular, PGPO+Llama-2-7B surpasses ADaPT+GPT-3.5 by relative margins of 9.5% and 17% points on WebShop and TextCraft. While prompting methods with GPT-4 improve the agent performance, PGPO+Llama-3-8B alleviates the need for few-shot context and achieves comparable or even better results. This demonstrates smaller open-source models can be effective agents through iterative training, rivaling or exceeding the agent capabilities of strong closed-source models.

Finally, we focus on the effectiveness of PGPO across different base models on various datasets. (1) Generalization on different models: Besides the LLaMA family of models, we also include Mistral-7B in Table 2 and Qwen2.5 series in Appendix C.3. Regardless of model sizes and families, our method consistently exhibit the advantage of P-code Plans guiding agent reasoning. (2) Generalization on diverse unseen tasks: To comprehensively assess its capability in out-of-distribution scenarios, we conduct additional experiments on ScienceWorld (Wang et al., 2022) in Appendix C.4 Table 13. PGPO outperforms the ETO and IPR baselines in generalizing to unseen tasks, where the performance gap is larger when dealing with more complex interactive tasks.

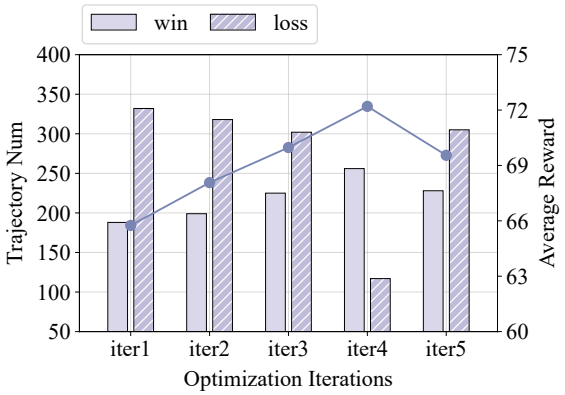


Figure 6: The correlation between collected contrastive trajectory dataset distribution and agent performance during iterative optimization. Here, "win" means agent-generated trajectory surpassing expert trajectory while "loss" represents falling short.

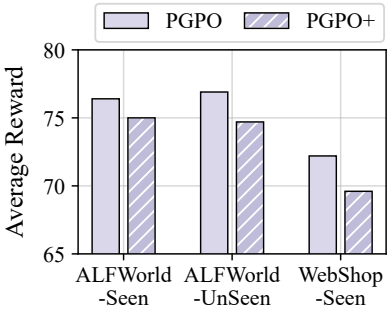


Figure 7: Comparison between PGPO+ and PGPO.

4.3 Ablation Study

Approach Ablations. Table 4 illustrates that the performance of PGPO obviously declines after removing certain key components. We observe that the most significant performance drop comes from the removal of SFT loss (- L_s), which is consistent with previous findings (Pang et al., 2024). To demonstrate the advantage of P-code Plans over NL plans, we specifically replace P-code Plans with NL plans (generation pipeline is same with Section 2.3) in training data and then employ PGPO. As expected, the results indicate even with subsequent iterative preference optimization, the performance upper bound brought by P-code Plans is still superior to that of NL plans. Additionally, the effect of using plan-following reward to include L_f in optimization loss is better than - L_f, indicating the necessity of plan-following rewards.

Ablation on Optimization Iterations. Figure 5 shows the iteration ablation results from two aspects: (a) With optimization iterations increase, all methods first exhibit performance improvements and then deteriorate due to excessive iterations. Among them, PGPO consistently achieves the high-

Task Instruction:

You are in the middle of a room. Looking quickly around you, you see a cabinet 10, a cabinet 9, a cabinet 8, a cabinet 7, a cabinet 6, a cabinet 5, a cabinet 4, a cabinet 3, a cabinet 2, a cabinet 1, a coffeemachine 1, a countertop 1, a diningtable 1, a drawer 2, a drawer 1, a fridge 1, a garbagecan 1, a microwave 1, a sinkbasin 1, a stoveburner 4, a stoveburner 3, a stoveburner 2, a stoveburner 1, and a toaster 1.
Your task is to: heat some apple and put it in diningtable.

NL Plan:

Step 1. look for the apple in the environment;
Step 2. take the apple;
Step 3. go to a microwave or other suitable appliance;
Step 4. heat the apple using microwave or other suitable appliance;
Step 5. go to the diningtable;
Step 6. put the heated apple in/on the diningtable.

Execode Plan (Python format):

```
for location in ["fridge 1", "drawer 1", "drawer 2",
                 "cabinet 1", "cabinet 2", "cabinet 3",
                 "cabinet 4", "cabinet 5", "cabinet 6",
                 "cabinet 7", "cabinet 8", "cabinet 9",
                 "cabinet 10 ", "a countertop 1"]:
    res = goto(location)
    if "apple" in res:
        apple_location = location
        break
take_from(res, apple_location)
goto("microwave 1")
heat(res, "microwave 1")
goto("diningtable 1")
put(res, "diningtable 1")
```

P-code Plan:

Abstract planning steps:
Step 1. object_name, location = locate_object(candidate_locations);
Step 2. take(object_name, location);
Step 3. go_to(microwave);
Step 4. heat(object_name, microwave);
Step 5. go_to(diningtable);
Step 6. put(object_name, diningtable).

Task-specific entities:
object = apple
candidate_locations = fridge 1, cabinets

Figure 8: Case study for our P-code Plan compared with other formats.

Method	ALFWorld	
	Seen	UnSeen
ETO	34.28%	32.83%
IPR	30.71%	28.35%
PGPO	**23.57%**	**26.86%**

Table 5: Invalid action rate on ALFWorld.

Method	Webshop
ETO	37.5
IPR	40.5
PGPO	**41.0**

Table 6: Success rate on WebShop.

est peak performance. We consider that this can be attributed to the excellent starting point (i.e., iter=0) in PGPO since the incorporation of P-code Plans into SFT data effectively enhances the agent reasoning capability. (b) Despite different base models, the peak performance of PGPO can be achieved within 4 iterations. However, the performance variation trend across them quite differs, which reflects the distinct extent of each model grasping the meta-knowledge inherent in P-code Plans. Furthermore, as depicted in Figure 6, agent achieves optimal performance during iterative optimization when the number of its exploration trajectory surpassing expert trajectory reaches a peak.

4.4 Analysis

Analysis on training time efficiency. We compare the time consumption of PGPO with two training-based baselines on WebShop. Under the same resource constraints, ETO, IPR and PGPO first undergo a 1-hour SFT phase and additionally require 1.5h, 4.5h and 3.2h per optimization iteration, respectively. Therefore, PGPO delivers a 9% performance improvement while maintains reasonable training efficiency, requiring less than twice the time cost of ETO.

P-code Plan guidance can reduce the incidence of action errors and omissions in reasoning. Taking ALFWorld as an example, we calculate the proportion of trajectories containing invalid actions for each method on Llama-2-7B in Table 5. The results demonstrate PGPO decreases action errors with the help of P-code Plans. Then we analyze the action omissions of agents via the task success rate. Since WebShop provides dense rewards, the trajectory is considered success only when final reward is 1.0, i.e., agent has selected all necessary product attributes without any omissions. From Table 6, it can be observed that PGPO achieves the highest success rate, indicating guidance from P-code Plans indeed reduces the agent's action omissions.

Step-wise reward does not necessarily elicit better LLM agents. Regarding the design of plan-following reward in PGPO, we only consider the alignment between the first step (containing the generated plans) and the second step to construct contrastive trajectory pairs. Since step-level process supervision has been effectively utilized in reasoning tasks (Lightman et al., 2024), we evaluate whether introducing step-wise reward to our PGPO could further facilitate agent reasoning. Fol-

lowing IPR (Xiong et al., 2024), we add step-level rewards into our method (denoted PGPO+). It can be observed from Figure 7 that step-wise reward negatively impacts the PGPO performance. To speculate on the reason behind, we manually check the quality of training data collected by PGPO+ and find that although step-level process supervision increases the data scale, some constructed preference pairs may be ambiguous, which poses a potential risk of reward hacking (Gao et al., 2023). Therefore, it is still challenging to accurately determine the contribution of the intermediate step, thus introducing step-wise reward instead plays a negative role in agent reasoning (Guo et al., 2025).

4.5 Case Study

In Figure 8, we show the generated P-code Plan compared with other two plan formats within the same task in ALFWorld. First, NL plans are less structured than plans in pseudocode or executable code format since elements such as articles and conjunctions may have no role in complex reasoning logic. Second, as described in Appendix C.2, Execode Plan is more verbose than P-code Plan. In this case, the Execode Plan lists almost all of locations for agent to explore, guaranteeing the solvability of this agent task. However, this may introduce unnecessary context and lead to blind trial-and-error, resulting in task failure due to exceeding the maximum interaction turns. By contrast, our P-code Plan strikes a balance between structural rigor and concision, thereby facilitating agent reasoning.

5 Related Works

LLM Agents. The remarkable capabilities of LLMs have spurred research into developing AI agents. These LLM agents are generally equipped with reasoning and acting capabilities, enabling them to handle a wide range of tasks (Richard, 2023; Nakajima, 2023; Liu et al., 2024). Prompt-based methods like ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2024) and ADaPT (Prasad et al., 2024) utilize strong closed-source LLMs to build powerful agents. However, prompting strategies are heavily dependent on those enhanced but expensive LLMs, resulting in high usage costs. Recent studies explore the fine-tuning methods based on open-source LLMs to improve agent intelligence (Chen et al., 2023; Zeng et al., 2024).

Agent Planning. Planning plays a crucial role in agent reasoning, with different planning paradigms, such as Plan&Solve (Wang et al., 2023) and Plan-Act (Liu et al., 2023b), offering diverse approaches to agent tasks. Few works have explored the potential of alternative plan formats beyond natural language. To utilize the precision of formal language, Li et al. (2024) constructs a context-free grammar to control the NL plan generation. Zhang et al. (2024) rely on translating NL tasks into Planning Domain Definition Language (PDDL) and then solve problems with PDDL planners. Silver et al. (2024) consider PDDL domains and directly use LLMs for generalized planning. However, the above studies just use such structured language to assist agent planning rather than empower LLMs to generate structured plans for enhanced reasoning. Concurrently with our work, Wen et al. (2025) introduce code-form plans to do reasoning tasks under the few-shot setting without fine-tuning.

Agent Learning. Previous works focus on learning from expert trajectory data to align agent behavior with expert (Chen et al., 2024; Yin et al., 2024). Recently, learning from preference has shown promise for developing LLM agents. NAT (Wang et al., 2024b) teaches the model to differentiate between correct and incorrect interactions during fine-tuning. ETO (Song et al., 2024) leverages iterative explored trajectories for training via DPO loss. IPR (Xiong et al., 2024) constructs step-wise contrastive action pairs using estimated step-level rewards to guide the agent optimization process. These efforts highlight iterative preference learning techniques unlock sophisticated agent capabilities.

6 Conclusion

In this paper, we propose PGPO, which empowers LLM agents with enhanced reasoning capabilities under the guidance of pseudocode-style plans. Our motivation is based on that abstract P-code Plan can capture efficient structural logic of reasoning compared with NL plans, suitable for LLM agent's generalization to analogous agent tasks. After incorporating automatically generated P-code Plans into existing ReAct-style datasets, PGPO starts by a competent base agent through SFT. Then, PGPO iteratively refines the base agent via preference optimization based on two planning-oriented rewards. Experimental results demonstrate PGPO effectively achieves new SOTA performance across three agent benchmarks. Further analysis shows that P-code Plan exhibits robust potential in mitigating action errors and critical step omissions during reasoning.

Limitations

This paper focuses on incorporating pseudocode-style plans to guide agent preference optimization. Despite its new SOTA performance, we acknowledge the following limitations of our work: 1) Our method deploys Monte Carlo sampling to estimate plan-following reward, incurring additional inference costs compared to the ETO baseline. However, sampling only needs to be conducted for one step per iteration, which is more efficient than the design of step-wise reward in IPR baseline. And ablation studies in Section 4.3 demonstrates the necessity of plan-following rewards. 2) Our method designs structured P-code Plan to enhance agent reasoning. Although powerful GPT-4o guarantees the plan quality to a certain extent, it is still necessary to research how to verify plans automatically beyond human verification.

In the future, we plan to conduct research on rule-based rewards (Guo et al., 2025) since the structured nature of our P-code Plan provides an interpretable scaffold for automating rule-based reward design. Furthermore, we explore extending our method to a multi-task training scenario, which can contribute to more generalized LLM agents.

References

Raghav Arora, Shivam Singh, Karthik Swaminathan, Ahana Datta, Snehasis Banerjee, Brojeshwar Bhowmick, Krishna Murthy Jatavallabhula, Mohan Sridharan, and Madhava Krishna. 2024. Anticipate & act: Integrating llms and classical planning for efficient task execution in household environments. In *International Conference on Robotics and Automation*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9354–9366.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025. Agentrefine: Enhancing agent generalization through refinement tuning. *arXiv preprint arXiv:2501.01702*.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Zelong Li, Wenyue Hua, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Formal-llm: Integrating formal language and natural language for controllable llm-based agents. *arXiv preprint arXiv:2402.00798*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

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

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, and 1 others. 2023b. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Liangwei Yang, Zuxin Liu, Juntao Tan, Prafulla K Choubey, Tian Lan, Jason Wu, Huan Wang, and 1 others. 2024. Agentlite: A lightweight library for building and advancing task-oriented llm agent system. *arXiv preprint arXiv:2402.15538*.

Yohei Nakajima. 2023. Babyagi: an experimental framework for a self-building autonomous agent.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. Adapt: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024a. Agent planning with world knowledge model. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. 2024b. AutoAct: Automatic agent learning from scratch for QA via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3003–3021.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Toran Bruce Richard. 2023. Significant-gravitas/autogpt: A collection of tools and experimental open-source attempts to make gpt-4 fully autonomous.

Shamik Roy, Sailik Sengupta, Daniele Bonadiman, Saab Mansour, and Arshit Gupta. 2024. Flap: Flow-adhering planning with constrained decoding in llms. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 517–539.

Wentao Shi, Mengqi Yuan, Junkang Wu, Qifan Wang, and Fuli Feng. 2024. Direct multi-turn preference optimization for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. Alfworld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.

Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B Tenenbaum, Leslie Kaelbling, and Michael Katz. 2024. Generalized planning in pddl domains with pretrained large language models. In *Proceedings of the AAAI conference on artificial intelligence*, pages 20256–20264.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024b. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024c. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

Jiaxin Wen, Jian Guan, Hongning Wang, Wei Wu, and Minlie Huang. 2025. Unlocking reasoning potential in large language models by scaling code-form planning. In *The Thirteenth International Conference on Learning Representations*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, and 1 others. 2024. Agentgym: Evolving large language model-based agents across diverse environments. *arXiv preprint arXiv:2406.04151*.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024a. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Yijun Yang, Tianyi Zhou, Kanxue Li, Dapeng Tao, Lusong Li, Li Shen, Xiaodong He, Jing Jiang, and Yuhui Shi. 2024b. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26275–26285.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2024. Agent lumos: Unified and modular training for open-source language agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12380–12403.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077.

Xiaopan Zhang, Hao Qin, Fuquan Wang, Yue Dong, and Jiachen Li. 2024. Lamma-p: Generalizable multi-agent long-horizon task allocation and planning with lm-driven pddl planner. *arXiv preprint arXiv:2409.20560*.

A PGPO Algorithm

We summarize the workflow of PGPO in Algorithm 1. Our algorithm starts by a Supervised Fine Tuning (SFT) stage. In this stage, expert dataset with P-code Plans generated in Section 2.2 is utilized to equip the base LLM with agent capabilities. Next, the algorithm proceeds with the data preparation stage. For each expert trajectory, the plan-following reward is calculated via Monte Carlo sampling method. Then, the base agent explores on expert trajectories to collect new trajectory data. Based on two planning-oriented rewards, two contrastive trajectory datasets are constructed. Finally, in the preference optimization stage, the base agent refines its parameters to improve its reasoning capability. The above data preparation and preference optimization stage will be repeated until exceeding the maximum iterations.

B Implementation Details

B.1 For Datasets

Table 7 summarizes the statistics information of the three agent datasets. For ALFWorld and WebShop, we choose the ReAct-style expert trajectories collected by Xiong et al. (2024). For TextCraft, we primarily use the train set from AgentTraj-L (Xi et al., 2024). Note that unseen tasks in ALFWorld ref to new task instances with possibly known object-receptacle pairs, but always in unseen rooms with different receptacles and scene layouts than in training data.

Dataset	#Train	#Test	#Turns
ALFWorld	2851	274 (140-Seen, 134-Unseen)	7.97
WebShop	1624	200 (200-Seen)	3.64
TexrtCraft	373	100 (100-Unseen)	7.86

Table 7: Statistics information of ALFWorld, WebShop and TextCraft. "#Turns" denotes the average number of iteraction turns for the expert trajectories.

B.2 For P-code Plan Generation

We illustrate the plan generation pipeline in Figure 2. Based on existing ReAct-style datasets (see Appendix B.1 for data source), we first extract the agent thoughts part for preparation. Then, we use

Algorithm 1: Workflow of **PGPO**

Input: Base LLM π_θ, Expert Dataset $\mathcal{D}_s = \{(u,p,\tau)^{(i)}\}_{i=1}^{|\mathcal{D}_s|}$: $\tau = (t_1, a_1, o_1, ..., t_n, a_n, o_n)$, Training Epoch T_1 for Supervised Fine Tuning, Training Epoch T_2 for Preference Optimization, Sampling Number N for Reward Estimation, Maximum Number of Iterations I

Output: The Enhanced LLM agent π_θ

// Supervised Fine Tuning Stage

for $epoch = 1$ **to** T_1 **do**

  $\pi_\theta \leftarrow \arg\min_{\pi_\theta} -\mathbb{E}_{\substack{(u,p,\tau)\\ \sim \mathcal{D}_s}} \{ \sum_{j=1}^{n} [\log \pi_\theta(t_j|u,p,\tau_{j-1}) + \log \pi_\theta(a_j|u,p,\tau_{j-1},t_j)] + \log \pi_\theta(p|u) \}$;

$\pi_{scorer} = \pi_\theta$;

for $(u,p,\tau) \in \mathcal{D}_s$ **do**

  Given the first two expert interaction rounds $\tau_2 = (t_1, a_1, o_1, t_2, a_2, o_2)$ as historical trajectory, use scorer agent to sample N subsequent trajectories: $\tau^s_{3:m'} \sim \pi_{\theta_{scorer}}(\cdot|u,p,\tau_2)$;

  Compute plan-following reward $r_f(u,p,\tau_2) = \frac{1}{N}\sum_{i=1}^{N} r_o(\tau^s_{3:m'}|u,p,\tau_2)^{(i)}$ of expert trajectory;

for $iter = 1$ **to** I **do**

  $\pi_{base} = \pi_\theta$; $\pi_{ref} = \pi_\theta$

  // Exploration Stage for Planning-oriented Trajectory Collection

  for $u \in \mathcal{D}_s$ **do**

    Get the P-code Plan and subsequent reasoning steps from base agent: $\hat{p} \sim \pi_{\theta_{base}}(\cdot|u)$, $\hat{\tau} \sim \pi_{\theta_{base}}(\cdot|u,\hat{p})$;

    Compare plan-driven rewards r_d of $(\hat{p},\hat{\tau})$ with expert trajectory (p,τ) to get $(p^w,\tau^w) \succ (p^l,\tau^l) \mid u$;

    Given the first expert interaction round (u,p,τ_1) as historical trajectory, get subsequent trajectory from base agent: $\hat{\tau}_{2:m} \sim \pi_{\theta_{base}}(\cdot|u,p,\tau_1)$;

    Similar to line 5-6, compute plan-following reward $r_f(p,u,\hat{\tau}_2)$ of agent-generated trajectory;

    Compare $r_f(p,u,\hat{\tau}_2)$ and $r_f(u,p,\tau_2)$ to get $\tau^w_{2:n} \succ \tau^l_{2:m} \mid (u,p,\tau_1)$;

  Construct two contrastive trajectory datasets: $D_p = \left\{(u,p^w,\tau^w,p^l,\tau^l)^{(i)}\right\}_{i=1}^{|D_p|}$, $D_f = \left\{(u,p,\tau_1,\tau^w_{2:n},\tau^l_{2:m})^{(i)}\right\}_{i=1}^{|D_f|}$;

  // Planning-guided Agent Learning Stage

  for $epoch = 1$ **to** T_2 **do**

    $\pi_\theta \leftarrow \arg\min_{\pi_\theta} (\mathcal{L}_p + \mathcal{L}_f + \mathcal{L}_s)$ using Eq. 3, Eq. 3 and Eq. 5;

return the enhanced LLM agent π_θ.

GPT-4o to summarize the step-by-step plan following our predefined P-code Plan format via few-shot prompting strategy. For each expert trajectory, only one P-code Plan is generated since we use greedy inference (temperature=0) in GPT-4o. One example prompt for plan distillation is shown below:

Example Prompt for Plan Distillation

Given the [Task Description], [Task] and [Solution Trajectory], you should summarize the step-by-step [Plan] in natural language for solving the task. Please note that the generated [Plan] should be global and do not contain the information from "Observation" part of [Solution Trajectory]. Then, you should format the generated [Plan] to strictly follow the pseudocode format and output in this format:
Step 1. ...
Step 2. ...
Step 3. ...
....
Here is one example.
<one-shot demonstration>

Now is your turn.
[Task Description]: **<task_description>**
[Task]: **<task>**
[Solution Trajectory]: **<agent_thoughts>**
[NL Plan]:
[P-code Plan]:

B.3 For Human Verification

To guarantee the quality of generated P-code Plans, we ask three NLP researchers to check the plan

formats and their consistency with original trajectories. According to statistic, only nearly 15% of the generated data need to be refined by human, while 85% are well structured. This demonstrates the reliability of our automatic P-code Plan generation pipeline, enabling scalable and quality-controlled data synthesis.

B.4 For SFT Stage

First, after distilling P-code Plans from agent thoughts, we need to incorporate these plans into original ReAct-style datasets. As described in Section 2.3, they are added into the first step of original trajectories. We prefix the plans with "First, I devise a plan for solving the task:" for incorporation and list one example of new constructed training data as below:

SFT Data Example

{
"from": "human",
"value": "**<task_description>**"
},{
"from": "gpt",
"value": "OK"
},{
"from": "human",
"value": "**<task>**"
},{
"from": "gpt",
"value": "**Thought**: First, I devise a plan for solving the task: **<distilled P-code Plan>** Now, I need to first check ...
Action: go to toiletpaperhanger 1"
},{
"from": "human",
"value": "**Observation**: On the toiletpaperhanger 1, you see ..."
}, ...

Next, we choose full-parameter fine-tuning for all models using FastChat framework. We detail the hyperparameters for SFT stage in Table 8.

Name	Value
num_train_epochs	3
train_batch_size	48
per_device_train_batch_size	4
per_device_eval_batch_size	4
gradient_accumulation_steps	2
learning_rate	2e-5
weight_decay	0.
warmup_ratio	0.03
lr_scheduler_type	"cosine"
model_max_length	4096

Table 8: Detailed hyperparameters used in SFT stage.

B.5 For Baselines

In this section, we provide a detailed introduction to the baselines, as well as our reproduction details.

- ETO (Song et al., 2024): This framework comprises two training phases: (1) behavior cloning stage, wherein the agent undergoes fine-tuning on expert trajectory data, followed by (2) learning from failures, which employs DPO (Rafailov et al., 2024) for subsequent policy refinement.
- IPR (Xiong et al., 2024): The iterative step-level process refinement framework enhances agent learning through step-by-step guidance. Via step-level reward estimation, IPR identifies discrepancies between agent-generated trajectories and the expert trajectories, thereby boosting the agent performance.

To reproduce experimental results, we maintain all the default hyperparameters in their public code[3] and carefully extend them to TextCraft dataset.

- ReAct (Yao et al., 2023): ReAct first integrates Chain-of-Thought (CoT) into LLM agent systems through a structured Thought-Action-Observation reasoning format. For the ReAct implementation, we adopt one-shot prompting for agent reasoning.
- ADaPT (Prasad et al., 2024): ADaPT dynamically decomposes complex tasks through recursive planning, automatically adjusting decomposition depth based on real-time feedback to align LLM competencies with evolving task demands. In our paper, we constrain the maximum interaction turns for fair comparison and directly use the open-source code for reproduction[4].

Regarding other traditional prompting methods like Plan&Solve (Wang et al., 2023) and Reflexion (Shinn et al., 2024), we do not include them for comparison in Table 3 because our chosen ADaPT baseline is enough strong and substantially outperforms them (see Table 9 for reference).

[3] ETO: https://github.com/Yifan-Song793/ETO, IPR: https://github.com/WeiminXiong/IPR

[4] https://github.com/archiki/ADaPT

Method	ALFWorld Seen	ALFWorld UnSeen	WebShop	TextCraft
Plan&Solve+GPT-3.5	46.4	43.3	61.8	22.0
Reflexison+GPT-3.5	56.4	57.5	62.4	25.0
ADaPT+GPT-3.5	<u>70.3</u>	<u>71.6</u>	<u>62.7</u>	<u>26.0</u>
PGPO+Llama-2-7B	**76.4**	**76.9**	**72.2**	**43.0**

Table 9: Comparative experiments on PGPO with more prompt-based baselines. The best and second-best results are marked in **bold** and <u>underline</u>.

Method	ALFWorld Seen	ALFWorld UnSeen	WebShop
w/o Plan	72.1	68.7	61.8
w/ NL Plan	70.7	69.4	63.0
w/ P-code Plan	**75.0**	**72.4**	**63.6**

Table 10: Comparative experiments using P-code Plans generated by the model itself.

B.6 For Benchmark Evaluation

Our evaluation framework follows previous works (Song et al., 2024; Xiong et al., 2024) and extends it to TextCraft benchmark. The maximum number of steps for ALFWorld, WebShop and TextCraft is set to 20, 10 and 20, respectively.

C Additional Experimental Results

C.1 Using Self-generated P-code Plan

As described in Section 2.2, we use one powerful closed-source LLM (i.e., GPT-4o) to generate P-code Plans from existing ReAct-style datasets for good quality control. To further demonstrate the format advantage of P-code Plan in agent reasoning, not knowledge distillation from other strong models, we conduct supplementary experiments that uses the plans generated by the model itself to construct new SFT data. Taking Mistral-7B as an example, Table 10 shows SFT *w/ P-code Plan* maintains its advantage over *w/o Plan* and *w/ NL Plan*, even when utilizing self-generated plans. This proves that the enhanced agent reasoning capabilities should be attributed to the structured nature of P-code Plan, rather than knowledge distillation from other models.

C.2 Compared with Executable Code Format

Similar to the generation of P-code Plan, we first meticulously curate few Execode Plan (standing for plans in executable code format) demonstrations and then utilize GPT-4o to synthesize the plan data via few-shot prompting. Taking ALFWorld as an example, we report the average reward of SFT *w/ P-code Plan*, *w/ Execode Plan* and *w/o Plan* across four open-source LLMs in Table 11. The results show that SFT *w/ P-code Plan* maintains better performance than SFT *w/ Execode Plan*. Sometimes SFT *w/ Execode Plan* even falls behind SFT *w/ Plan*. Through error analysis, we attribute this to two reasons: 1) executable code generation is more challenging than natural language or pseudocode generation; 2) *Execode Plan* is more verbose than *P-code Plan*, which may introduce some noise information into subsequent reasoning.

Setting	Llama-2 -7B	Llama-2 -13B	Llama-3 -8B	Mistral -7B
ALFWorld-Seen				
w/o Plan	60.0	67.1	67.1	72.1
w/ Execode Plan	62.1	60.0	67.1	61.4
w/ P-code Plan	**65.0**	**72.9**	**68.6**	**75.0**
ALFWorld-Unseen				
w/o Plan	67.2	67.9	72.4	68.7
w/ Execode Plan	68.7	63.4	64.9	64.2
w/ P-code Plan	**70.1**	**70.9**	**75.4**	**72.4**

Table 11: Comparison between w/ P-code Plan and w/o Plan, w/ Execode Plan on ALFWorld. **Bold** indicates the best results of each model.

Method	Qwen2.5-7B	Qwen2.5-14B
SFT	62.6	64.8
PGPO$_{SFT}$	65.3	65.9
ETO	67.7	68.8
PGPO	**70.7**	**72.3**

Table 12: Average reward on WebShop.

C.3 Results on Qwen Series

To further validate the generalization of our method on more models, we test PGPO on Qwen2.5-7B and Qwen2.5-14B model (Yang et al., 2024a). Taking WebShop as an example, Table 12 shows that when applying to Qwen2.5 series, our method still achieves the best performance. PGPO$_{SFT}$ represents the SFT model in our pipeline, which still has an advantage over the naive SFT baseline due to the incorporation of our designed P-code Plans. Since IPR baseline requires enormous inference costs for step-wise reward estimation, we are unable to reproduce them due to limited time constraints. We leave comprehensive comparisons as future work.

Method	ScienceWorld			
	Seen		Unseen	
	AR	SR(%)	AR	SR(%)
SFT	67.7	70.1	52.4	57.8
ETO	69.0	70.7	56.8	66.8
IPR	70.2	70.6	54.4	61.6
PGPO	**75.5**	**75.8**	**66.2**	**76.8**

Table 13: Evaluation results of PGPO and baselines on ScienceWorld. Experiments are based on Llama-2-7B. The evaluation metrics are the average reward (AR) and success rate (SR).

Dataset Size	10%	30%	50%	70%	100%
ALFWorld-Seen	55.0	70.7	71.4	75.0	76.4
ALFWorld-Unseen	45.5	61.9	63.4	69.4	76.9

Table 14: Average reward on ALFWorld when PGPO applied on smaller training data subsets.

C.4 Evaluation on ScienceWorld

ScienceWorld (Wang et al., 2022) is one agent benchmark for testing scientific reasoning abilities, which provides dense rewards from 0 to 1. For training, we use the expert trajectories collected from Song et al. (2024), comprising 1483 instances. For evaluation, we includes the development set (ScienceWorld-seen) with 194 seen scenarios and the test set (ScienceWorld-unseen) consisting of 211 new unseen task scenarios. The ScienceWorld-seen set can assess in-distribution capability while the ScienceWorld-unseen set can measure out-of-distribution generalization of the agents. Since its evaluation pipeline runs relatively slowly, we only conduct comparative experiments on the Llama2-7B model. Table 13 shows the average reward and success rate of PGPO and baselines on ScienceWorld. We can observe that PGPO maintains better performance than ETO and IPR, further indicating the advantage of P-code Plan guidance. Moreover, IPR baseline even falls short compared to the ETO baseline on ScienceWorld-Unseen, which once again confirms that step-wise reward does not necessarily elicit better LLM agents.

C.5 Using Smaller Training Data Subsets

We supplement the PGPO experiments about training Llama2-7B on smaller subsets (10%, 30%, 50%, 70%, 100%). Taking ALFWorld benchmark as an example, the evaluation results are shown in Table 14. The results indicate with 30% of training data, the in-distribution performance of our method is already strong. However, the out-of-distribution performance continues to improve as the dataset size increases, demonstrating its scalability. This aligns with expectations since OOD tasks are more challenging, but with the full dataset, our method achieves significantly better performance.

C.6 Collected Contrastive Trajectory Dataset Distribution over Iterations

Figure 6 illustrates a connection between the trajectories collected and the performance of the agent over optimization iterations. It indicates that for the agent, maintaining an increasingly number of winning trajectories leads to improved average rewards during the optimization process. Thus, we can utilize this finding to select our optimal optimization iterations.

D Case Study

We present one trajectory example for each agent benchmark compared with the current leading baseline IPR in Figure 9, 10, 11 and 12, respectively.

- In the WebShop example, we find IPR baseline tends to select the product item located in the first position of one page, which is sub-optimal. In contrast, our PGPO agent carefully browses through the whole page and successfully selects the optimal product located in the middle position of the page.
- In the ALFWorld example, our PGPO agent quickly locates the pencil with the guidance of generated P-code Plan, thus completing the task with the minimum interaction turns. Nevertheless, IPR agent blindly searches for the pencil in the shelves and drawers and fails to find the desired pencil due to exceeding the maximum interaction turns.
- In the TextCraft example, we observe IPR agent made a mistake in the thinking process, thereby ending in a collapse. On the contrary, guided by the generated P-code Plan, PGPO agent efficiently complete the crafting task.

Case Study of WebShop

IPR Agent

Task Instruction: I need a clear glass wall mounting lamp for my bath room. and I would prefer 2-light size, and price lower than 120.00 dollars
Thought: I need to find a clear glass wall mounting lamp with 2 lights for my bathroom that is under $120. I'll start by searching for these specific criteria.
Action: search[wall mounting lamp glass clear 2-light bathroom]
Observation: Page 1 (Total results: 50) [SEP] Next > [SEP] B097BFLS8Q [SEP] MOTINI 2 Light Bathroom Vanity Light Modern Wall Sconce Metal Wall Mount Lamp Fixture with 5 Inches Round Glass Canopy Black [SEP] $49.99 [SEP] B09KMCGTR1 [SEP] YAOHONG Industrial Wall Sconce 2-Lights Modern Vanity/Bathroom Lamp in Black with Clear Glass Globe Shades Wall Mount Light Fixtures for Hallway Kitchen Living Room [SEP] $59.99 [SEP] B095JTF9KC [SEP] Bathroom Vanity Light Fixtures, 2-Light Black Wall Sconce Lighting Wall Lamp with Clear Glass Shade, Vintage Wall Mounted Lights Bathroom Lights for Mirror, Living Room, Bedroom, Hallway, Porch [SEP] $39.99 …
Thought: There are several options available, but I'm interested in the MOTINI 2 Light Bathroom Vanity Light because it fits the description and is well within my budget at $49.99. I'll click on this one for more details.
Action: click[B097BFLS8Q]
Observation: <*product details*>
Thought: The product details for the MOTINI lamp show that it comes in two sizes. Since I prefer a larger size, I'll select the 22"*10" option to ensure it fits the space I have in mind.
Action: click[22"*10"]
Observation: <*product details*>
Thought: The price is still within my budget, and the lamp fits the size and light requirements I have. ..., so I'll go ahead and make the purchase by clicking the 'Buy Now' button.
Action: click[buy now]
Reward: 0.66

PGPO Agent

Task Instruction: I need a clear glass wall mounting lamp for my bath room. and I would prefer 2-light size, and price lower than 120.00 dollars
Thought: First, I devise a plan for solving the task:
Step 1. selected_item = search_item(\"wall mounting lamp with clear glass shade for bathroom 2-light size price lower than 120.00 dollars\");
Step 2. click(selected_item);
Step 3. select_size(\"2-light\");
Step 4. buy().
I think I should use the search bar to look for the wall mounting lamp with clear glass shade for the bathroom that meets my criteria.
Action: search[wall mounting lamp clear glass bathroom 2-light size price < 120.00]
Observation: Page 1 (Total results: 50) [SEP] Next > [SEP] B07786JG8X [SEP] Aspen Creative 62104, Four-Light Metal Bathroom Vanity Wall Light Fixture, 32\" Wide, Transitional Design in Satin Nickel with Clear Glass Shade [SEP] $121.73 … [SEP] B095JTF9KC [SEP] Bathroom Vanity Light Fixtures, 2-Light Black Wall Sconce Lighting Wall Lamp with Clear Glass Shade, Vintage Wall Mounted Lights Bathroom Lights for Mirror, Living Room, Bedroom, Hallway, Porch [SEP] $39.99 …
Thought: I think I should click on the option that meets my criteria, which is B095JTF9KC - Bathroom Vanity Light Fixtures, 2-Light Black Wall Sconce Lighting Wall Lamp with Clear Glass Shade.
Action: click[B095JTF9KC]
Observation: <*product details*>
Thought: I think I should click on the '2-light' option to select the size I prefer for the wall mounting lamp.
Action: click[2-light]
Observation: <*product details*>
Thought: I think I should click on the 'Buy Now' button to purchase the wall mounting lamp.
Action: click[Buy Now]
Reward: 1.0

Figure 9: Case study for WebShop based on Llama-2-7B.

Case Study of ALFWorld

IPR Agent

Task Instruction: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: put a pencil in shelf.
Thought: I need to locate a pencil and place it on a shelf. I'll start by checking the nearest shelf to see if there's already a pencil there.
Action: go to shelf 1
Observation: On the shelf 1, you see nothing
Thought: Shelf 1 is empty, so I need to check the next shelf for a pencil.
Action: go to shelf 2
Observation: On the shelf 2, you see a mug 1.
Thought: Shelf 2 has a mug, but no pencil. I'll continue searching the next shelf.
Action: go to shelf 3
Observation: On the shelf 3, you see a desklamp 1.
Thought: Shelf 3 has a desklamp, but still no pencil. I must keep looking.
Action: go to shelf 4 # Blind trail-and-error
Observation: On the shelf 4, you see a alarmclock 1.
…
Thought: Drawer 5 has a credit card and a keychain, but no pencil. I'll check drawer 6 next.
Action: go to drawer 6
Observation: The drawer 6 is closed
Thought: Drawer 6 is closed. I need to open it to see if the pencil is inside.
Action: open drawer 6
Observation: You open the drawer 6. The drawer 6 is open. In it, you see nothing
Thought: The drawer is empty. Drawer 7 is my last option to find a pencil in this room.
Action: go to drawer 7 # Invalid action
Observation: Nothing happens.
Reward: 0.0 # Task failure due to not finding the pencil

PGPO Agent

Task Instruction: You are in the middle of a room. Looking quickly around you, you see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf 3, a shelf 2, and a shelf 1.
Your task is to: put a pencil in shelf.
Thought: First, I devise a plan for solving the task:
Step 1. pencil_name, location_name = locate_a_pencil(candidate_locations);
Step 2. take_pencil(pencil_name, location_name);
Step 3. go_to_a_shelf();
Step 4. put_pencil_on_shelf().
The task is to put a pencil in a shelf. I need to locate a pencil first. The desk is a common place where pencils are kept, so I should check there.
Action: go to desk 1 # With the p-code plan guidance, the agent quickly locates the mostly likely place.
Observation: On the desk 1, you see a desklamp 1, and a pencil 1.
Thought: I see a pencil on desk 1. I need to pick it up to complete the task.
Action: take pencil 1 from desk 1
Observation: You pick up the pencil 1 from the desk 1.
Thought: Shelf 1 is the closest and most accessible shelf. I should go there to place the pencil.
Action: go to shelf 1
Observation: On the shelf 1, you see nothing."
Thought: The shelf is empty, which is perfect for placing the pencil without causing any clutter. I will put the pencil here to complete the task.
Action: put pencil 1 in/on shelf 1
Observation: You put the pencil 1 in/on the shelf 1.
Reward: 1.0 # Task success with only 4 interactions

Figure 10: Case study for ALFWorld based on Llama-2-7B.

Case Study of TextCraft

IPR Agent

Task Instruction: Crafting commands:
craft 8 cyan stained glass using 8 glass, 1 cyan dye\ncraft 8 black stained glass pane using 8 glass pane, 1 black dye\ncraft 8 gray stained glass using 8 glass, 1 gray dye\ncraft 2 gray dye using 1 black dye, 1 white dye\ncraft 1 black dye using 1 ink sac\ncraft 1 black dye using 1 wither rose\ncraft 8 yellow stained glass using 8 glass, 1 yellow dye\ncraft 8 pink stained glass using 8 glass, 1 pink dye\ncraft 8 black concrete powder using 1 black dye, 4 sand, 4 gravel\ncraft 3 light gray dye using 1 black dye, 2 white dye\ncraft 8 black stained glass using 8 glass, 1 black dye\ncraft 8 black terracotta using 8 terracotta, 1 black dye\ncraft 8 orange stained glass using 8 glass, 1 orange dye
Goal: craft black stained glass
Thought: I should check if I can get black stained glass directly from the environment or the inventory.
Action: inventory
Observation: Inventory: You are not carrying anything
Thought: I need to get black stained glass from the environment.
Action: get black stained glass
Observation: Could not execute get black stained glass
Thought: I cannot get black stained glass directly, I need to craft it. From the crafting commands, I can use: craft 8 black stained glass using 8 glass, 1 black dye. Ingredients needed: 8 glass, 1 black dye. Input assumption: I have the ingredients needed in my inventory. Let me verify this first.
Action: inventory
Observation: Inventory: You are not carrying anything.
Thought: I need to get the ingredients: 8 glass and 1 black dye.
Action: get 8 glass
Observation: Got 8 glass
Thought: I have obtained the ingredients: 8 glass. Now, I need to get 1 black dye.
Action: get 1 black dye
Observation: Could not find black dye
Thought: I cannot get black dye directly, I need to craft it. From the crafting commands, I can use: craft 1 black dye using 1 ink sac, craft 1 black dye using 1 wither rose, craft 1 black dye using 1 black dye, craft 1 black dye using 1 white dye, craft 1 black dye using 1 black concrete powder. Input assumption: I have the ingredients needed in my inventory. Let me verify this first.
Action: inventory
Observation: Inventory: [glass] (8)
Thought: I don't have any ingredients in my inventory. I need to get the ingredients: 1 ink sac, 1 wither rose, 1 black dye, 1 white dye, 1 black concrete powder.
Action: get 1 ink sac
Observation: Got 1 ink sac

PGPO Agent

Task Instruction: Crafting commands:
craft 8 black stained glass pane using 8 glass pane, 1 black dye\ncraft 2 gray dye using 1 black dye, 1 white dye\ncraft 8 lime stained glass using 8 glass, 1 lime dye\ncraft 16 glass pane using 6 glass\ncraft 1 black dye using 1 wither rose\ncraft 8 black stained glass using 8 glass, 1 black dye\ncraft 8 orange stained glass using 8 glass, 1 orange dye\ncraft 1 beacon using 1 nether star, 5 glass, 3 obsidian\ncraft 8 cyan stained glass using 8 glass, 1 cyan dye\ncraft 1 black bed using 1 white bed, 1 black dye\ncraft 1 black dye using 1 ink sac\ncraft 1 writable book using 1 book, 1 ink sac, 1 feather\ncraft 8 gray stained glass using 8 glass, 1 gray dye
Goal: craft black stained glass.
Thought: First, I devise a plan for solving the task:
Step 1. had_objects = inventory();
Step 2. if \"black stained glass\" not in had_objects: find_flag = get(\"black stained glass\");
Step 3. if find_flag == False: \"craft 8 black stained glass using 8 glass, 1 black dye\" = choose_crafting_commands(\"black stained glass\"), needed_ingredients = [\"8 glass\", \"1 black dye\"], get(\"8 glass\");
Step 4. find_dye_flag = get(\"1 black dye\");
Step 5. if find_dye_flag == False: \"craft 1 black dye using 1 wither rose\" = choose_crafting_commands(\"1 black dye\"), needed_ingredients = \"1 wither rose\", get(\"1 wither rose\");
Step 6. craft 1 black dye using 1 wither rose;
Step 7. craft 8 black stained glass using 8 glass, 1 black dye
I should check if I can get black stained glass directly from the environment or the inventory.
Action: inventory
Observation: Inventory: You are not carrying anything.
Thought: There is no black stained glass in my inventory. I should try to get it from the environment.
Action: get black stained glass
Observation: Could not execute get black stained glass
Thought: I cannot get black stained glass directly, I need to craft it. From the crafting commands, I can use: craft 8 black stained glass using 8 glass, 1 black dye. Ingredients needed: 8 glass, 1 black dye. I should check if I have these ingredients in my inventory.
Action: inventory
Observation: Inventory: You are not carrying anything
Thought: My inventory is empty. I need to get the ingredients: 8 glass and 1 black dye first.
Action: get 8 glass
Observation: Got 8 glass
Thought: I have got the glass. Now, I need to get 1 black dye.
Action: get 1 black dye

Figure 11: Case study for TextCraft based on Llama-2-7B.

Case Study of TextCraft

IPR Agent

Thought: I have obtained the ingredient: 1 ink sac. Now, I need to get 1 wither rose.
Action: get 1 wither rose
Observation: Got 1 wither rose
Thought: I have obtained the ingredient: 1 wither rose. Now, I need to get 1 black dye.
Action: get 1 black dye # Wrong action
Observation: Could not find black dye
Thought: I cannot get black dye directly, I need to craft it. From the crafting commands, I can use: craft 1 black dye using 1 ink sac, craft 1 black dye using 1 wither rose, craft 1 black dye using 1 black dye, craft 1 black dye using 1 white dye, craft 1 black dye using 1 black concrete powder. Input assumption: I have the ingredients needed in my inventory. Let me verify this first.
Action: inventory
Observation: Inventory: [glass] (8) [ink sac] (1) [wither rose] (1)
Thought: I don‘t have any ingredients in my inventory. I need to get the ingredients: 1 ink sac, 1 wither rose, 1 black dye, 1 white dye, 1 black concrete powder. # Wrong thoughts leads to repeated wrong actions
Action: get 1 ink sac
…
Thought: I don't have any ingredients in my inventory. I need to get the ingredients: 1 ink sac, 1 wither rose, 1 black dye, 1 white dye, 1 black concrete powder.
Action: get 1 ink sac
Observation: Got 1 ink sac
Thought: I have obtained the ingredient: 1 ink sac. Now, I need to get 1 wither rose.
Action: get 1 wither rose
Observation: Got 1 wither rose
Reward: 0.0 # Task failure due to exceeding the maximum interaction turns

PGPO Agent

Observation: Could not find black dye
Thought: I cannot get black dye directly, I need to craft it. From the crafting commands, I can craft 1 black dye using 1 wither rose or 1 ink sac. I should check if I have these ingredients in my inventory.
Action: inventory
Observation: Inventory: [glass] (8)
Thought: My inventory only has glass. I need to get either a wither rose or an ink sac to craft black dye.
Action: get 1 wither rose
Observation: Got 1 wither rose
Thought: I have got the wither rose. Now, I can craft the black dye.
Action: craft 1 black dye using 1 wither rose
Observation: Crafted 1 minecraft:black_dye
Thought: I have crafted the black dye. Now, I have all the ingredients needed to craft the black stained glass: 8 glass, 1 black dye.
Action: craft 8 black stained glass using 8 glass, 1 black dye
Observation: Crafted 8 minecraft:black_stained_glass
Reward: 1.0

Figure 12: Case study for TextCraft based on Llama-2-7B (Continuations in Figure 11).