EXPERT-INTEGRATED ACTIVE LEARNING FOR OPTIMIZING LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in Large Language Models (LLMs) have created new opportunities for their application in interactive environments. However, these agentic tasks present significant challenges due to the complexity of long and specialized interaction trajectories that are underrepresented in standard training distributions. While Reinforcement Learning (RL) post-training offers a promising approach to mitigate the need for extensive human-annotated data, it faces fundamental limitations in exploration efficiency when applied to LLMs. In this paper, we introduce a novel framework that synergistically combines RL post-training with Active Learning (AL) for LLM agents. By choosing informative tasks with reward-based filter and diversity-based selection criteria, our approach enables models to not only refine their capabilities through autonomous exploration but also strategically request expert demonstrations for challenging scenarios, thereby extending their exploration boundaries. We demonstrate the efficacy of this method on the AppWorld benchmark with DeepSeek-V3.1 as expert, showing comparable performance with full supervision while using minimal expert demonstrations. We then further look into adapting our framework for different budgets and examine the factors that affect the final performance, as well as validate our framework on WebShop benchmark. Our method highlights the potential of efficiently integrating limited human resources within RL pipelines to enhance LLM agents' capabilities in complex interactive environments.

1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have sparked significant interest in leveraging these models for a wide range of downstream tasks. Increasingly, LLMs are required to interact not only with users, but also with external tools and environments, such as APIs (Qin et al., 2023), web browsers (Thil et al., 2024), and embodied environments (Shridhar et al., 2021). Unlike traditional Natural Language Processing (NLP) tasks, these scenarios often involve multi-turn interactions, where the model must interpret feedback from the environment and adapt its strategy dynamically. However, such long and complex interaction trajectories typically reside in the long tail of the data distribution, as environmental observations are not only decided by model inputs but also the environment states. Also, environment-specific settings and action space are underrepresented in general training datasets. Consequently, zero-shot or few-shot prompting approaches often yield suboptimal performance in these agentic settings.

A common remedy to adapt pretrained models on these agentic tasks is to finetune models with annotated data. However, collecting sufficient high-quality human-environment interaction trajectories can often prohibitively expensive or infeasible, especially for scenarios with specific requirements or limited resources. As an alternative, Reinforcement Learning (RL) post-training has gained popularity. For tasks where outcomes can be evaluated by predefined rules or reward models, RL enables models to optimize their behavior by collecting trajectories and reward signals autonomously, thereby significantly reducing dependence on manual annotation.

Despite these advancements, RL approaches for LLMs face fundamental limitations in exploration efficiency. (i) From the perspective of action space, LLMs operate over vast token vocabularies, where each generated token maps to different environment-specific actions. This results in an enormous exploration space, within which only a sparse subset is truly meaningful. (ii) From the perspective of policy initialization, pretrained LLMs already possess an initial policy shaped by their pretraining data. This prior knowledge governs the generation of environment actions, but can be difficult to adapt or modify through training on a specific downstream task. To address these challenges, we propose a novel framework that combines LLM RL post training with Active Learning (AL). With this framework, the model can not only use its current policy to examine and refine its own capabilities but also actively query for additional expert data to expand the upper bound of its exploration. Our main contributions are summarized as follows:

- We examine the feasibility of introducing expert demonstrations to RL training of LLM Agents, showing performance improvements with limited expert demonstrations in AppWorld and WebShop benchmark.
- We propose a novel AL framework for RL training of LLM Agents, which actively selects informative tasks for expert annotation based on reward-based filter and diversity-based selection strategies.
- We look into the performance of our method under different budget setting and revealing the relationship between used expert demonstration and final performance.

2 RELATED WORKS

Interactive Environments for LLM Agents Recent benchmarks increasingly evaluate LLM-based agents in realistic and interactive settings. ALFWorld provided an interactive benchmark that enables transferring abstract textual instructions to a 3D environment, evaluating the reasoning and planning abilities of embodied LLM agents (Shridhar et al., 2021). WebShop introduced a scalable web-based environment with real products and instructions(Yao et al., 2022). AgentBench broadened coverage across domains such as operating systems, databases, and games (Liu et al., 2023). OSWorld benchmarked multimodal agents on real desktop applications, highlighting the challenges of multimodal interaction (Xie et al., 2024). AppWorld provided lightweight, verifiable environments for phone-API tasks (Trivedi et al., 2024). τ-bench assessed agent consistency and rule-following in tool-augmented dialogues (Yao et al., 2024). SWE-bench targeted software engineering tasks such as code generation, debugging, and refactoring (Jimenez et al., 2024). Collectively, these benchmarks underscore the growing effort to evaluate LLMs in complex, interactive tasks that require reasoning, planning, and tool use.

Reinforcement Learning for LLMs Beyond supervised instruction tuning, recent work applies RL to improve LLM reasoning, alignment, and agentic control. Group Relative Policy Optimization (GRPO) replaces the critic in Proximal Policy Optimization (PPO) with a group-wise baseline, reducing training memory cost and improving model reasoning ability (Shao et al., 2024). Decoupled Clip and Dynamic sAmpling Policy Optimization (DAPO) introduces important techniques like clip-higher and dynamic sampling, further improving models' performance (Yu et al., 2025). Dr.GRPO looks into the design of advantage estimation as well as prompting templates, achieving SOTA results on multiple math benchmarks (Liu et al., 2025).

Reinforment Learning for Downstream Agentic Tasks In LLM agent settings, several works have explored ways to transfer, stabilize and improve GRPO-like methods for multi-turn interaction tasks. Leave-One-Out Proximal Policy Optimization (LOOP) combines PPO-style clipped updates with a Leave-One-Out advantage estimator, achieving SOTA on AppWorld benchmark (Chen et al., 2025). DeepSWE introduces trajectory-level compact filtering, greatly improves open-weight model's performance on SWE-Bench (Luo et al.). MobileRL incorporates successful trajectory replay and negative rollout pruning to enhance training efficiency (Xu et al.). SimpleTIR focuses on improving trajectory quality by filtering rollouts with void turns where no valid actions are provided, preventing gradient explosions and stabilizing end-to-end RL (Xue et al., 2025).

Active Learning in NLP Tasks Active Learning focuses on improving model performance with minimal labeled data by strategically selecting the most informative samples for annotation. In previous LLM-related works, uncertainty-based methods have been commonly used to improve text classification and text summarization (Li et al., 2024; Rouzegar & Makrehchi, 2024; Bayer, 2025). Some works also explore how to apply AL to efficiently learn reward models for Reinforcement Learning from Human Feedback (RLHF) (Liu et al., 2024).

3 Problem Setting and Preliminary

LLM Agent Interaction as a Markov Decision Process Interactions involving LLM-based agents can be naturally framed as a finite-horizon Partially Observable Markov Decision Process (POMDP), represented by the tuple $(\mathcal{S}, \mathcal{A}, R, P, H)$. Here, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $R : \mathcal{S} \times \mathcal{A}$ the reward function, $P : \mathcal{S} \times \mathcal{A}$ the state transition dynamics, and H the finite horizon. In practical settings, rewards are most often provided only at the end of an episode, once the horizon H is reached.

GRPO To adapt GRPO to this framework, let G denote the rollout group size for each sample and T the number of interaction turns in a trajectory τ. We define an action a as a token generated by the LLM policy π_θ, with the response at turn t denoted as the action sequence $\mathbf{a}_t = \left[a_{p(t)+1}, a_{p(t)+2}, \ldots, a_{p(t)+l(t)}\right]$, where $l(t)$ is the number of output tokens at turn t, and $p(t) = \sum_{k=1}^{t-1} l(k)$. The full trajectory τ is written as $\tau = (\mathbf{c}, \mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_T, \mathbf{a}_T)$, with final reward R. The state s at token position $p(t) + j$ consists of the initial prompt $\mathbf{c}$ concatenated with all previously generated tokens and observations:$s_{p(t)+j} = (\mathbf{c}, \mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_t, \mathbf{a}_t[: j])$.

GRPO follows the structure of the PPO algorithm, optimizing the objective

$$\mathcal{L}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{L_i} \sum_{j=1}^{L_i} \min\Big(\text{ratio}_{i,j}(\theta) \hat{A}_{i,j},\ \text{clip}\big(\text{ratio}_{i,j}(\theta), 1-\epsilon, 1+\epsilon\big) \hat{A}_{i,j}\Big) - \beta\, D_{\text{KL}}\big(\pi_{\theta_{\text{old}}} \,\|\, \pi_\theta\big) \right],$$

where

$$L_i = p_i(T_i + 1), \quad \text{ratio}_{i,j}(\theta) = \frac{\pi_\theta(a_{i,j} \mid s_{i,j})}{\pi_{\theta_{\text{old}}}(a_{i,j} \mid s_{i,j})}, \quad \hat{A}_{i,\cdot} = \frac{R_i - \text{mean}_{k=1}^{G} R_k}{\text{std}_{k=1}^{G} R_k}.$$

In our main experiments, we adopt a modified version of GRPO that uses Leave-One-Out advantage estimation. The advantage for each token is given by $\hat{A}_{i,\cdot} = \frac{G}{G-1}\Big(R_i - \text{mean}_{k=1}^{G} R_k\Big)$.

Active Learning for LLM Agent We consider the following active learning setting. Let $\mathcal{D} = \{d_i\}_{i=1}^{N}$ denote a pool of task data, where each d_i is a unique task instance. The goal is to iteratively select informative tasks $\mathcal{D}^* \subset \mathcal{D}$ for expert demonstration $\mathcal{T}^*$, thereby improving the LLM agent's performance while minimizing annotation cost. The total annotation cost is $\mathcal{C} = \sum_{d \in \mathcal{D}^*} c \cdot m$, where c is the cost of annotating one trajectory and m is the number of expert demonstrations collected for a task.

4 Method

Overview Figure 1 demonstrates an overview of our proposed framework. With the RL training framework (e.g., GRPO), we first samples a batch of tasks from the training dataset. The training LLM Agent interacts with the environment to collect rollouts as well as reward signals and other information. Then we perform a two-step expert task selection. For the collected rewards for each task over a given step range, we perform a reward-based filter to identify the challenging task candidates. These task candidates are further filtered with a diversity-based selection strategy to ensure the selected tasks are not redundant. The final selected tasks are then sent to experts (in our experiments, a stronger model) for annotation, and the collected expert demonstrations are added to the expert replay buffer. Finally, the sampled trajectories from the LLM Agent are mixed with the expert demonstrations from the expert replay buffer for the calculation of policy gradients, which is used to update the parameters of the LLM Agent.

Reward-Based Filter A key aspect of AL is the selection of informative samples for expert annotation, which is usually measured by the model's uncertainty about a sample. In the RL setting,

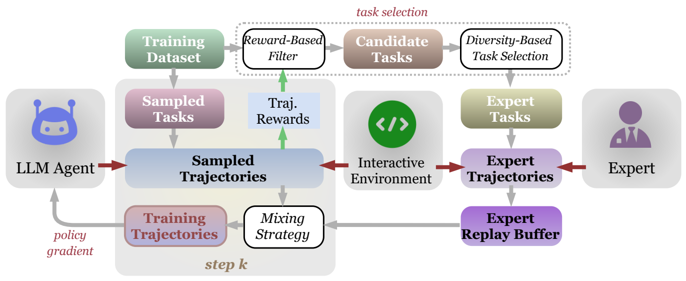

Figure 1: An overview of our proposed framework.

uncertainty can be interpreted as the model's difficulty in finding policy changes to improve performance on specific task types. Intuitively, tasks that the current policy consistently fails on, or where no noticeable improvement is observed over time, provide the clearest signal that additional supervision from experts may be beneficial.

To implement this idea, we apply the following two-step criteria for task filtering: (i) Within a sliding step window of size u, we identify tasks for which at least 80% of sampled rollouts remain unsuccessful. This criterion highlights tasks that the current policy persistently struggles with. (ii) Among these difficult tasks, we further select those where the average reward achieved in the most recent $u/2$ steps does not surpass that of the preceding $u/2$ steps by more than a small margin ϵ. This stagnation in reward suggests that the model has failed to make meaningful progress or to discover an effective learning trajectory.

By combining success rate filtering with reward progression analysis, our selection mechanism prioritizes tasks that are both persistently unsolved and resistant to incremental policy updates, thereby ensuring that expert annotation is directed towards the most informative and impactful training signals.

Diversity-Based Task Selection Another important aspect of AL is to ensure the diversity of the selected samples, which requires effective de-redundancy among the chosen tasks. To address this, we introduce a diversity-based selection strategy following the initial score-based filtering of task candidates. Specifically, after computing a similarity metric for each candidate, we employ a max-min greedy selection strategy to select a batch of diverse tasks, as detailed in Algorithm 1. This approach iteratively selects tasks that are maximally distinct from both previously selected tasks and those chosen in recent steps, thereby promoting coverage of a broader range of task types. Furthermore, we introduce a buffer step size to incorporate historical selections from the last v steps into the max-min similarity calculation. This mechanism controls the diversity of selected tasks during the buffer step while also allowing additional flexibility for the model to revisit challenging or underexplored task types if no improvements are made during the previous steps, ultimately improving the efficiency and robustness of expert demonstration collection. The accumulative cost of expert demonstrations at step n is $\mathcal{C}_{\leq n} = \sum_{k=1}^{n} \sum_{d \in \mathcal{D}_t^*} c \cdot m$.

Mixing Strategy After updating the expert replay buffer with the newly collected expert demonstrations, we construct the rollouts for policy updates by mixing trajectories generated by the LLM Agent with carefully selected expert demonstrations from the buffer. The goal of this mixing strategy is to effectively inject high-quality expert knowledge into the training process while avoiding excessive reliance on expert data, thereby preserving the diversity and exploration ability of the LLM Agent's own trajectories. To achieve this balance, we design the following rules: (i) With a mixing ratio α, we incorporate at most αG expert demonstrations for each task. To ensure that only superior knowledge is introduced, we restrict candidates to demonstrations whose rewards are strictly higher than those of all sampled trajectories from the LLM Agent. (ii) When selecting eligible expert demonstrations from the replay buffer, we prioritize demonstrations with distinct reward values to encourage reward diversity. (iii) During the replacement of sampled trajectories with expert demonstrations, we adopt a preference order: first replacing trajectories with redundant rewards

Algorithm 1 Diversity-Based Max-Min Task Selection

```
1: Input: History Expert Task List $\mathbb{D}^* = [\mathcal{D}_1^*, \mathcal{D}_2^*, \ldots, \mathcal{D}_{n-1}^*]$, Candidate Task Set at Step $n$ $\hat{\mathcal{D}}_n$,
   Similarity Metric $\mathrm{sim}(\cdot, \cdot)$, Similarity Threshold $\delta$, Buffer Step Size $v$
2: Output: Selected Task Set at Step $n$ $\mathcal{D}_n^*$
3: $\mathcal{D}_{hist}^* \leftarrow \bigcup_{i=n-v}^{n-1} \mathcal{D}_i^*$, $\mathcal{D}_n^* \leftarrow []$
4: if $\mathcal{D}_{hist}^* = \emptyset$ then
5:     select $d \sim \hat{\mathcal{D}}_n$, $\mathcal{D}_n^* \leftarrow \mathcal{D}_n^* \cup \{d\}$, $\hat{\mathcal{D}}_n \leftarrow \hat{\mathcal{D}}_n \setminus \{d\}$
6: end if
7: while $\hat{\mathcal{D}}_n \neq \emptyset$ do
8:     $d^* \leftarrow \arg\max_{d \in \hat{\mathcal{D}}_n} \min_{d' \in \mathcal{D}_n^* \cup \mathcal{D}_{hist}^*} \mathrm{sim}(d, d')$
9:     if $\min_{d' \in \mathcal{D}_n^* \cup \mathcal{D}_{hist}^*} \mathrm{sim}(d^*, d') < \delta$ then
10:        $\mathcal{D}_n^* \leftarrow \mathcal{D}_n^* \cup \{d^*\}$, $\hat{\mathcal{D}}_n \leftarrow \hat{\mathcal{D}}_n \setminus \{d^*\}$
11:    else
12:        break
13:    end if
14: end while
15: return $\mathcal{D}_n^*$
```

to maintain the diversity of original samples, and then replacing those with relatively low rewards to improve overall rollout quality. This principled mixing strategy enables the training process to benefit from superior expert knowledge while preserving sufficient variation in the rollouts, ultimately helps develop a more robust and generalizable policy. The full algorithmic details are provided in Appendix C.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Dataset We adopt AppWorld (Trivedi et al., 2024) as our main experimental environment. AppWorld provides a lightweight, verifiable, yet challenging benchmark for mobile app interaction. The training set consists of 30 scenarios, each with 3 tasks, divided into 3 difficulty levels. Following LOOP's setup (Chen et al., 2025), we use level-1 and level-2 tasks for training, covering 24 scenarios and 72 tasks. The relatively small training set enables efficient experimentation under different settings and budget constraints. For evaluation, AppWorld offers two test sets: Test Normal (168 tasks) and Test Challenge (417 tasks).

Models and Training We use Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct (Qwen et al., 2025) as base LLMs, with LoRA adapters (Hu et al., 2021) for efficient finetuning. Our framework is implemented on top of RAGEN (Wang et al., 2025). At each training step, we sample 40 tasks from the training set, with a rollout group size of 6 per task.

Experiment Settings We evaluate these three main configurations:

- */GRPO/baseline*: GRPO without external demonstrations. The policy is updated only from the model's own rollouts.
- */GRPO/full_demonstration*: Expert demonstrations for all tasks are available throughout training and are mixed into rollouts according to the mixing strategy.
- */GRPO/active_learning*: The expert replay buffer is initialized empty. During training, tasks are chosen via a reward-based filter and diversity-based selection. Expert demonstrations for these tasks are then added to the buffer and remain available for subsequent step.

We also provides results from these two baseline implementations for comparision:

Table 1: Performance of different models and settings on AppWorld benchmark. "Task Success Rate" indicates average success rate for all tasks, and "Scene Success Rate" indicates success rate for all scenes (a scene is considered a success if all three tasks under the scene are successfully executed).

Models / Settings	**Cost ($\mathcal{C}$)**	**Train**		**Test Normal**		**Test Challenge**	
		Task Success Rate (%)	Scene Success Rate (%)	Task Success Rate (%)	Scene Success Rate (%)	Task Success Rate (%)	Scene Success Rate (%)
Qwen2.5-7B-Instruct	0	1.38	0.00	0.60	0.00	1.92	0.00
Qwen2.5-14B-Instruct	0	23.61	8.33	10.71	1.79	6.00	1.44
Expert DeepSeek-V3.1	-	56.94	33.33	56.55	37.50	40.63	18.75
Qwen2.5-7B-Instruct							
/GRPO^0/baseline	0	40.27	29.17	10.71	5.36	5.27	1.44
/SFT/full_demonstration	360	44.44	25.00	27.98	10.71	8.39	2.16
/GRPO/baseline	0	41.67	33.33	11.90	3.57	3.60	0.72
/GRPO/full_demonstration	360	72.22	50.00	29.76	16.07	10.07	2.16
/GRPO/active_learning	**165**	**66.67**	**45.83**	**28.57**	**12.50**	**7.19**	**1.44**
Qwen2.5-14B-Instruct							
/GRPO^0/baseline	0	69.44	50.00	43.45	26.79	19.64	8.63
/SFT/full_demonstration	360	56.94	37.50	37.50	17.86	14.39	3.60
/GRPO/baseline	0	72.22	58.33	43.45	32.14	18.94	7.91
/GRPO/full_demonstration	360	76.39	66.67	51.19	30.36	23.50	7.91
/GRPO/active_learning	**75**	**77.78**	**66.67**	**49.40**	**32.14**	**21.82**	**9.35**

- */GRPO^0/baseline*: Vanilla GRPO without external demonstrations. Instead of using LOOP style advantage estimation, using default normalized reward.
- */SFT/full_demonstration*: Model is directly finetuned on expert demonstrations for all tasks.

In both */GRPO/full_demonstration* and */GRPO/active_learning*, we set the mixing rate to $\alpha = 0.33$. For */GRPO/active_learning*, we use a similarity threshold of $\delta = 0.65$, a reward filter window of $u = 5$, a reward filter threshold of $\epsilon = 0.05$, and a buffer update step size of $v = 10$. Expert demonstrations are simulated using DeepSeek-V3.1. For each training task, we pre-collect $m = 5$ demonstrations, which are added to the replay buffer when required. For simplicity, we assume the cost of each expert demonstration $c = 1$.

5.2 Main Results

Table 1 summarizes the main experimental results on the AppWorld benchmark. As shown, both Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct perform poorly on this benchmark. Even after standard GRPO training, the task success rate of Qwen2.5-7B-Instruct on the Test Normal split is only 11.9%, while the success rate on the training set remains below 50%. This indicates that the model struggles to fully explore the training set on its own, let alone develop a generalizable policy for unseen tasks. In contrast, incorporating expert demonstrations leads to a notable performance boost. Both */GRPO/full_demonstration* and */GRPO/active_learning* achieve significantly higher success rates, with improvements of approximately 18% over the GRPO baseline. This confirms that expert trajectories provide crucial guidance, helping the model explore previously unseen states and learn more effective task-solving strategies. The training dynamics shown in Figure 2 further support this conclusion. We observe that the gap between */GRPO/baseline* and */GRPO/full_demonstration* emerges and stabilizes after roughly 20 training steps, indicating that expert demonstrations not only accelerate learning but also enable the model to reach solutions beyond its initial capabilities. A similar pattern is observed for Qwen2.5-14B-Instruct. Although the improvements are less pronounced than for the 7B model, both */GRPO/full_demonstration* and */GRPO/active_learning* still outperform the baseline. We hypothesize that the smaller performance gap arises from the limited capability

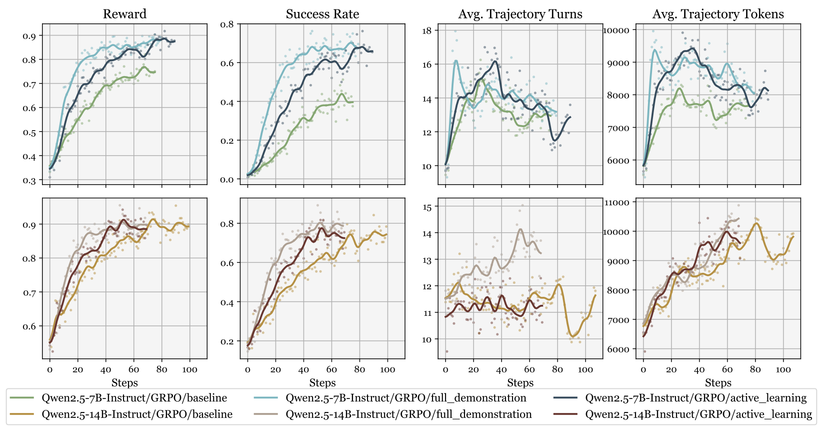

Figure 2: Training metrics of Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct under three settings. Settings with expert demonstrations reach higher rewards and success rate significantly faster than */GRPO/baseline* setting.

Table 2: Usage of expert demonstrations in */GRPO/full_demonstration* and */GRPO/active_learning* setting. "Expert tasks ($|\mathcal{D}^*|$)" denotes the number of tasks selected for expert demonstrations. "Cost ($\mathcal{C}$)" denotes number of expert demonstrations collected. "Used" denotes the number of expert demonstrations used in final rollout after mixing strategy. "Appearances" denotes the number of expert demonstration appearance in all rollouts after mixing strategy. "Efficiency" is defined as "Used" / "Appearances", the average times of appearances for a used rollout.

Models / Settings	**Expert Tasks ($|\mathcal{D}^*|$)**	**Expert Demonstrations**				**Total Rollouts**
		Cost ($\mathcal{C}$)	Used	Appearances	Efficiency	
Qwen2.5-7B-Instruct						
/GRPO/full_demonstration	72/72	360	241	629	2.61	12,000
/GRPO/active_learning	**33/72**	**165**	**119**	**457**	**3.91**	**19,200**
Qwen2.5-14B-Instruct						
/GRPO/full_demonstration	72/72	360	117	326	2.78	14,400
/GRPO/active_learning	**15/72**	**75**	**52**	**205**	**3.94**	**15,600**

of our simulated expert, which constrains the quality of the demonstrations and, consequently, their impact on a stronger model. Nevertheless, these results consistently validate the effectiveness of incorporating expert demonstrations into the training process.

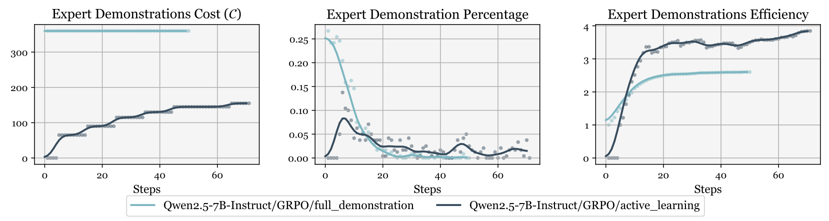

Figure 3: Usage of expert demonstrations during training of Qwen2.5-7B-Instruct under */GRPO/-full_demonstration* and */GRPO/active_learning* setting.

Table 2 shows the statistics of expert demonstration cost and usage under different settings. In */GRPO/active_learning*, Qwen2.5-7B-Instruct and Qwen2.5-14B-Instruct each acquire 33 and 15 of the total 72 training tasks as expert tasks, achieving a much lower cost $\mathcal{C}$. In our experiments, about

70% collected demonstration are actually used during training, as the mixing strategy only admits superior demonstrations. The average appearance of each demonstration used to update policy, denoted as expert demonstration efficiency, is significantly higher than */GRPO/full_demonstration*. This frequent re-use suggests that the expert tasks selected in */GRPO/active_learning* are inherently of a more challenging subset, since it takes more step for model to learn and generate rollouts with the same high rewards consistently.

Figure 3 shows changes of expert demonstration usage over the training process of the */GRPO/-full_demonstration* setting and */GRPO/active_learning* setting. We can see that in the */GRPO/active_learning* setting, the demand for additional demonstrations gradually decreases as the model improves its capabilities and stabilizes its policy. At the same time, expert demonstration efficiency remains relatively constant, suggesting that expert demonstrations added later of the contributes equally to the policy gradients as earlier ones.

5.3 Ablation Studies

Table 3: */GRPO/active_learning* with different similarity thresholds δ.

Settings (δ)	**Expert Demonstrations**		**Test Normal**	
	Cost($\mathcal{C}$)	Used	Task Success Rate (%)	Scene Success Rate (%)
0.25	60	43	17.26	3.57
0.45	110	84	27.38	10.71
0.65	165	119	28.57	12.50

Table 4: */GRPO/active_learning* with different early stopping steps for AL.

Settings (Early Stop)	**Expert Demonstrations**		**Test Normal**	
	Cost($\mathcal{C}$)	Used	Task Success Rate (%)	Scene Success Rate (%)
25 steps	90	71	25.60	12.50
45 steps	130	91	26.78	12.50
75 steps	165	119	28.57	12.50

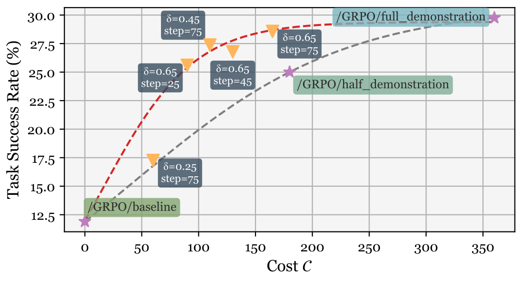

Figure 4: Demonstration cost and task success rate on Test Normal with Qwen2.5-7B-Instruct under different settings. Settings with our proposed framework is marked with yellow triangles and settings with random expert task selection marked with purple star. Logistic curve fit under these settings are respectively colored red and grey. Our proposed framework has a significant bigger Area Under Curve (AUC) than random selection method, indicating its overall superiority.

A key consideration when applying AL in production is managing annotation costs. To accommodate varying budget constraints, we test two strategies. The first is to adjust the similarity threshold in diversity-based selection, allowing control over the number of selected tasks for sustained training. The second is to apply early stopping of the AL process once the desired budget limit is reached, which is more suitable for a fixed training set. In our ablation studies, we examine both strategies and analyze how changes in budget impact model performance.

Table 3 shows the usage of expert demonstrations and performance on Test Normal with different δ. With smaller data, numbers of new expert tasks added shrinks, leading to a smaller overall demonstration cost over time. The same goes for experiments with different early stopping steps of active learning as shown in Table 4, but less drastic as more demonstrations are introduced in early steps than later steps with our strategy.

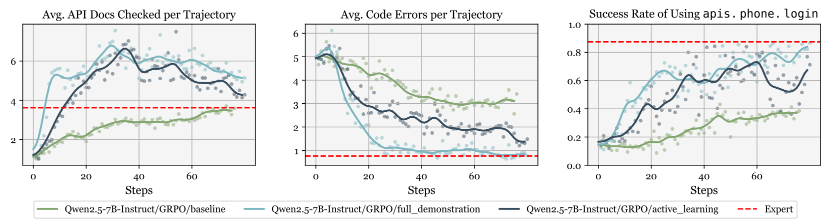

Figure 5: Average number of code documentation query and code errors in each trajectory under three settings. Settings with expert demonstrations exhibit a much faster grasp on documentation checking, as well as a lower code block error rate. Expert demonstrations also help agent to learn more fine-grained, specific policies, as the example of `apis.phone.login` shows.

Cost–Performance Relationship in Demonstration Usage To illustrate the trade-off between cost and performance with our AL framework, Figure 4 plots the cost of demonstrations against the task success rate on Test Normal. As the results show, the number of used demonstrations and performance in our framework fits a logistic function with an upper bound.

Impact of Expert Task Selection on Learning Efficiency We perform another experiment, */GRPO/half_demonstration*, in which half of the training set is randomly selected as expert tasks, and the corresponding expert demonstrations are provided from the start, similar to */GRPO/full_demonstration*. The logistic function fits for */GRPO/baseline*, */GRPO/half_demonstration*, and */GRPO/full_demonstration* reflect the trade-off with random expert task selection, which is significantly weaker than our proposed task selection process.

Influence of Demonstration Timing We notice that the results of */GRPO/active_learning* with $\delta = 0.25$ drift away from the fitted curve. In this setting, far fewer expert tasks are added each round. We attempt to continue the training and add more expert demonstrations to match the number of the other two δ settings, but the model seems already converged on a rather stable policy, and extra demonstrations do not improve the performance, indicating that for a fixed dataset, the timing when expert demonstrations are introduced also affects the development of a superior policy.

5.4 Trajectory Analysis

We analyse training trajectories across different settings to understand how expert demonstrations guide the model toward a more reliable policy. A core requirement in AppWorld is correct API usage, a behavior not directly rewarded but crucial for efficiency and final performance. While stronger models (e.g., Qwen2.5-32B-Instruct) can acquire this behavior through GRPO alone, smaller models such as Qwen2.5-7B/14B-Instruct frequently misuse APIs under the */GRPO/baseline* setting and fail to form consistent habits such as checking documentation before invoking functions. As shown in Figure 5, */GRPO/full_demonstration* and */GRPO/active_learning* encourage Qwen2.5-7B-Instruct to consult documentation roughly twice as often and reduce codeblock errors. Expert demonstrations also improve more fine-grained behaviors. For instance, the API `apis.phone.login` requires a phone number instead of an email. Without demonstrations, the model fails this call in 60% of attempts; with demonstrations, the success rate rises to around 75%. Models trained with demonstrations check the relevant documentation in around 85% of trajectories, compared to around 15% under */GRPO/baseline*. These results show that expert demonstrations help LLM agents develop environment-aligned behavioral patterns, prioritizing actual environment specifications over prior assumptions.

5.5 Additional Validation on WebShop

To further validate the generalizability of our method, we perform extra experiments on WebShop(Yao et al., 2022). We aggregate tasks by their goal products, then selecting a subset as candidate task set for possible expert demonstration. We use DeepSeek-V3.2 as expert, where we also provides oracle information of the task if the model cannot achieve success after a few trials. We

Table 5: Performance of Qwen2.5-3B-Instruct and different settings on WebShop benchmark.

Models / Settings	**Cost ($\mathcal{C}$)**	**Train**		**Test**	
		Success Rate (%)	Reward	Success Rate (%)	Reward
Qwen2.5-3B-Instruct	0	4.20	42.02	4.80	41.11
Expert DeepSeek-V3.2 + Oracle Information	-	35.38	65.71	34.80	64.34
/GRPO/baseline	0	52.29	76.42	56.20	78.04
/GRPO/full_demonstration	4,270	66.30	83.31	68.00	84.74
/GRPO/active_learning	**970**	**64.99**	**83.50**	**64.00**	**81.92**

use a mixing rate of $\alpha = 0.33$, a similarity threshold of $\delta = 0.7$, a reward filter window of $u = 5$, a reward filter threshold of $\epsilon = 0.05$, and a buffer update step size of $v = 10$. We use Qwen2.5-3B-Instruct as base model. As shown in 1, with less than 25% expert demonstration cost, */GRPO/active_learning* achieves around 2/3 performance growth of */GRPO/full_demonstration*. We provide more information about the implementation in D.

6 Discussion

In our experiment, we implement the expert $\mathcal{E}$ with an external model, denoted as $\mathcal{M}^*$. This design allows for consistent demonstration trajectories throughout different experiment setting, enabling more reliable and comparable evaluations. However, there remain some issue: (i) In experiments, about 30% of demonstrations remain unused due to the mixing strategy. (ii) Despite no issues were observed in training, distribution mismatch between expert demonstrations and sampled rollouts may affect GRPO's on-policy learning.

Expert Design In practice, the expert design can be more complex, efficient and capability-aware with joint effort of human expert $\mathcal{H}$, LLM $\mathcal{M}^*$ and task-specific information $\mathcal{I}_i$, which may include current rollouts from agent, oracle information and evaluation metrics. Here are some possible beneficial interactions between these entities:

- $\mathcal{E}(\mathcal{H}, \mathcal{M}^*)$ Expert trajectories can be formed by human expert supervising the model's interaction with the environment and making necessary adjustments at each turn. The model can be either the external model or the current training model. This interaction helps reduce human labor while maintaining consistency between expert trajectories and model rollouts.
- $\mathcal{E}(\mathcal{M}^*, \mathcal{I}_i)$ Expert trajectories can be formed via model reflection. Providing current rollouts in context increases the likelihood of sampling successful trajectories with $\mathcal{M}^*$ or building accumulative progress based on the capability of the current training agent. Oracle information and evaluation metrics can further reduce the difficulty of obtaining successful trajectories.
- $\mathcal{E}(\mathcal{H}, \mathcal{I}_i)$ Providing current rollouts and evaluation metrics helps the human expert design trajectories with specific rewards or environment feedback, informed by the mixing strategy, improving the efficiency of expert trajectory generation.

7 Conclusion

In this paper, we investigate the feasibility of incorporating expert demonstrations into the training of LLM agents. Our experiments on the AppWorld and WebShop benchmark show that including expert demonstrations helps less capable models improve performance by both broadening exploration and encouraging more consistent behavioral patterns. Furthermore, we propose an active learning framework to manage annotation budgets while maximizing the utility of expert demonstrations. Future work could explore extending this framework to other agentic tasks and developing a more generalizable approach for efficient expert annotation.

REFERENCES

Markus Bayer. ActiveLLM: Large Language Model-based Active Learning for Textual Few-Shot Scenarios. In Markus Bayer (ed.), *Deep Learning in Textual Low-Data Regimes for Cybersecurity*, pp. 89–112. Springer Fachmedien, Wiesbaden, 2025. ISBN 978-3-658-48778-2. doi: 10.1007/978-3-658-48778-2_7.

Kevin Chen, Marco F. Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement Learning for Long-Horizon Interactive LLM Agents. *CoRR*, abs/2502.01600, 2025. doi: 10.48550/ARXIV.2502.01600.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models, October 2021.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, November 2024.

Dongyuan Li, Ying Zhang, Zhen Wang, Shiyin Tan, Satoshi Kosugi, and Manabu Okumura. Active Learning for Abstractive Text Summarization via LLM-Determined Curriculum and Certainty Gain Maximization. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8959–8971, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.523.

Pangpang Liu, Chengchun Shi, and Will Wei Sun. Dual Active Learning for Reinforcement Learning from Human Feedback, December 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. doi: 10.48550/arXiv.2308.03688.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective, March 2025.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. DeepSWE: Training a Fully Open-sourced, State-of-the-Art Coding Agent by Scaling RL. https://www.together.ai/blog/deepswe.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs, October 2023.

Qwen, 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, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 Technical Report, January 2025.

Hamidreza Rouzegar and Masoud Makrehchi. Enhancing Text Classification through LLM-Driven Active Learning and Human Annotation. In Sophie Henning and Manfred Stede (eds.), *Proceedings of the 18th Linguistic Annotation Workshop (LAW-XVIII)*, pp. 98–111, St. Julians, Malta, March 2024. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, April 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. doi: 10.48550/arXiv.2010.03768.

Lucas-Andreï Thil, Mirela Popa, and Gerasimos Spanakis. Navigating WebAI: Training Agents to Complete Web Tasks with Large Language Models and Reinforcement Learning. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pp. 866–874, April 2024. doi: 10.1145/3605098.3635903.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents, July 2024.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning, May 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024. doi: 10.48550/arXiv.2404.07972.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. MOBILERL: ADVANCING MOBILE USE AGENTS WITH ADAPTIVE ONLINE REINFORCEMENT LEARNING. https://github.com/Xiao9905/AutoGLM/blob/main/static/papers/mobilerl_0820.pdf.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. SimpleTIR: End-to-End Reinforcement Learning for Multi-Turn Tool-Integrated Reasoning, September 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. doi: 10.48550/arXiv.2207.01206.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. \(\tau\)-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale, May 2025.

A Usage of LLMs in Paper Writing

During the paper writing, LLMs are used solely for polishing the writing, such as correcting spelling and grammar errors, and for no other purpose.

B GRPO Implementation Details

Multi-turn Agentic RL LLM training suffers from instablility issues. We found these following techniques helpful to stabilizes the training process:

- Response Reformatting: For each turn generation, we use regex to check if the generated response follows the expected format. If not, we first attempt to use a series of rules to reformat the response into the expected format, then if the reformatting fails, this turn is marked as invalid.
- Selective Resampling and Rollback: After the generation of each turn in one rollout, we check the validity of the generated message. If the message is invalid (e.g., no actions can be found, incomplete API calls, syntax errors, etc.), we resample up to N times. If all N resamples are invalid, we rollback to the $k + b * m$ turns before and restart the generation from there, where m is the times of rollback. If we rollback M times without getting valid trajectory, we stop this rollout and keep it at the last valid turn.
- Compact Filtering: First proposed in DAPO (Yu et al., 2025) then further adapted in Deep-SWE (Luo et al.). For rollout that ended prematurely (e.g., max token limit reached, max turn limit reached, etc.), we mask the whole rollout in loss calulation.

Our implementation is based on RAGEN (Wang et al., 2025)

C Mixing Strategy

We provide the pseudo code of our mixing strategy in Algorithm 2.

D WebShop Setting

Here is a more detailed description about the setting of our extra validation experiments on Web-Shop:

- We follow RAGEN (Wang et al., 2025) and use “items_ins_v2_1000” and “items_shuffle_1000” as our dataset, then randomly sample 5410 tasks as training set, 1000 tasks as validation set and 500 as test set.
- We first aggregates the training set by their goal products. Then for each product, we sample $\min(4, \lceil|\mathcal{T}|/2\rceil)$ tasks as a subset $\hat{\mathcal{D}}^0$ for possible expert demonstrations.
- In WebShop, we use $\mathcal{E}(\mathcal{M}^*, \mathcal{I}_i)$ expert design, where $\mathcal{M}^*$ is DeepSeek-V3.2 and $\mathcal{I}_i$ includes meta information of tasks. We first sample 3 trajectories $\mathcal{M}^*$. Then, if none of these 3 trajectories achieve success, we add $\mathcal{I}_i$ into context and continue to generate 2 more trajectories. Else, we keep the same prompt and continue to generate 2 more.
- In Reward-based Filter, we calculate the average reward difference at product level, and products further go through Diversity-based Selection. The intersection of $\hat{\mathcal{D}}^0$ and tasks corresponding to selected products are finally selected as expert tasks.
- The similarity is calculated using the query, category, attribute and option information of each product, where category are compared at phrase level, and others first broken down into words then used to calculate set similarity. Then the 4 scores are finally given weights 0.3, 0.4, 0.2, 0.1 to form the final similarity score.

E Computation and Memory Cost

In our framework, the computation and memory cost mostly comes from:

- Reward-based Filter: In this module, we calculate the average reward of tasks in two concessive step windows. We first iterate over each rollout to get the rewards, then calculate the difference in average reward for each task involved. The time complexity is $O(u \times G \times |\mathcal{D}_n|)$, where $u/2$ is the size of a window, G is the rollout number for each task and $|\mathcal{D}_n|$ is the number of tasks at step n. The memory complexity is $O(|\mathcal{D}_n|)$. The operations are simple scalar additions and multiplications and should cost minimal time or memory.s

Algorithm 2 Mixing Strategy

```
1: Input: Sampled Trajectory Set For Task $d_i$ $\mathcal{T}_i = \{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,G}\}$, Expert Demonstration
   Set For Task $d_i$ $\mathcal{T}_i^* = \{\tau_{i,1}^*, \tau_{i,2}^*, \ldots, \tau_{i,m}^*\}$, Mixing Ratio $\alpha$, Outcome Reward Function $R(\cdot)$
2: Output: Mixed Trajectory Set For Task $d_i$ $\tilde{\mathcal{T}}_i$
3: Initialize $\mathcal{T}_i^{candi} \leftarrow []$
4: for each $\tau_{i,j}^* \in \mathcal{T}_i^*$ do
5:     if $R(\tau_{i,j}^*) > \max \tau_{i,k} \in \mathcal{T}_i R(\tau_{i,k})$ then
6:         $\mathcal{T}_i^{candi} \leftarrow \mathcal{T}_i^{candi} \cup \{\tau_{i,j}^*\}$
7:     end if
8: end for
9: Initialize $\tilde{\mathcal{T}}_i \leftarrow []$, $\mathcal{T}_i^{sub} \leftarrow []$
10: while $|\tilde{\mathcal{T}}_i| < \alpha \times G$ and $\mathcal{T}_i^{candi} \neq \emptyset$ do
11:     Randomly select $\tau_{i,j}^* \in \mathcal{T}_i^{candi}$
12:     if $R(\tau_{i,j}^*) = \text{any}_{\tau_{i,k} \in \tilde{\mathcal{T}}_i} R(\tau_{i,k})$ then
13:         $\mathcal{T}_i^{sub} = \mathcal{T}_i^{sub} \cup \{\tau_{i,j}^*\}$
14:     else
15:         $\tilde{\mathcal{T}}_i \leftarrow \tilde{\mathcal{T}}_i \cup \{\tau_{i,j}^*\}$
16:     end if
17:     $\mathcal{T}_i^{candi} = \mathcal{T}_i^{candi} \setminus \{\tau_{i,j}^*\}$
18: end while
19: if $|\tilde{\mathcal{T}}_i| < \alpha \times G$ and $\mathcal{T}_i^{sub} \neq \emptyset$ then
20:     Randomly select $\alpha \times \max(G - |\tilde{\mathcal{T}}_i|, |\mathcal{T}_i^{sub}|)$ trajectories from $\mathcal{T}_i^{sub}$ and add to $\tilde{\mathcal{T}}_i$
21: end if
22: Initialize $\mathcal{T}_i^{sub} \leftarrow []$
23: while $|\tilde{\mathcal{T}}_i| < G$ do
24:     Randomly select $\tau_{i,j} \in \mathcal{T}_i$
25:     if $R(\tau_{i,j}) = \text{any}_{\tau_{i,k} \in \tilde{\mathcal{T}}_i} R(\tau_{i,k})$ then
26:         $\mathcal{T}_i^{sub} = \mathcal{T}_i^{sub} \cup \{\tau_{i,j}\}$
27:     else
28:         $\tilde{\mathcal{T}}_i \leftarrow \tilde{\mathcal{T}}_i \cup \{\tau_{i,j}\}$
29:     end if
30:     $\mathcal{T}_i = \mathcal{T}_i \setminus \{\tau_{i,j}\}$
31: end while
32: if $|\tilde{\mathcal{T}}_i| < G$ and $\mathcal{T}_i^{sub} \neq \emptyset$ then
33:     Randomly select $\max(G - |\tilde{\mathcal{T}}_i|, |\mathcal{T}_i^{sub}|)$ trajectories from $\mathcal{T}_i^{sub}$ and add to $\tilde{\mathcal{T}}_i$
34: end if
35: return $\tilde{\mathcal{T}}_i$
```

- Diversity-based Selection: In this module, we use similarity between tasks to further select appropriate expert tasks. The similarity computation cost is very dependent on the design of the similarity metric design. In our experiment, we use the similarity of the set of API functions involved in tasks, which can be pre-computed with time complexity and memory complexity $O(|\mathcal{D}|^2)$. The operations are simple set similarity calculation ($|A \cap B|/|A \cup B|$) and should cost minimal time or memory.

  The time and memory complexity of Max-Min selection is $O(|\hat{\mathcal{D}}_n|^3)$ and $O(|\hat{\mathcal{D}}_n|^2)$ in theory, where $|\hat{\mathcal{D}}_n|$ is the number of candidate expert tasks at step n, but much less in practice as we apply a threshold to end it prematurely. For a larger batch size, we can further reduce the cost by using a sampling-based Max-Min selection where instead of calculating the similarity of all the tasks left, we sample k tasks to check if there are any tasks that qualifies, which further reduce the time complexity to $O(|\hat{\mathcal{D}}_n|^2 \times k)$.

- Trajectory Mixing: In this module, we mix the agent rollouts with expert demonstrations with a few pre-defined rules. The time and memory complexity is $O(G + m)$, where G is the rollout number for each task and m is the number of expert demonstrations for each tasks, which are all rather small constants.

In conclusion, the extra operations introduced in these modules are orthogonal to the size of the models or environments and mostly only related to the rollout batch size. More importantly, these operations are almost all scalar operations is negligible compared to the rollout process itself.

F Example Trajectory

Here we provide an example trajectory from AppWorld for reference.

```
< PROMPT START >
[
  { "role": "system", "content": "You're a helpful assistant. " },
  {
    "role": "user",
    "content": "I am your supervisor and you are a super intelligent AI Assistant whose job is to achieve
   ↪  my day-to-day tasks completely autonomously.\n\nTo do this, you will need to interact with app/s
   ↪  (e.g., spotify, venmo etc) using their associated APIs on my behalf. For this you will undertake
   ↪  a *multi-step conversation* using a python REPL ... code, that I will execute, to solve the task.
   ↪  Let's start with the task\n\nMy name is: Melissa Bailey. My personal email is
   ↪  mel.bailey@gmail.com and phone number is 3383946795.\nTask: How many playlists do I have in
   ↪  Spotify?"
  },
  {
    "role": "assistant",
    "content": "Okay. Lets first find which APIs are available to use in
   ↪  Spotify.\n<answer>\nprint(apis.api_docs.show_api_descriptions(app_name='spotify'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n ...\n \"login : Login to your account.\",\n \"logout : Logout from your
   ↪  account.\",\n ...\n]"
  },
  {
    "role": "assistant",
    "content": "Okay. Looks like I can use the `login` api. Lets find its
   ↪  specifications.\n\n<answer>\nprint(apis.api_docs.show_api_doc(app_name='spotify',
   ↪  api_name='login'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"app_name\": \"spotify\",\n \"api_name\": \"login\",\n \"path\":
   ↪  \"/auth/token\",\n \"method\": \"POST\",\n \"description\": \"Login to your account.\",\n
   ↪  \"parameters\": [\n  {\n   \"name\": \"username\",\n   \"type\": \"string\",\n   \"required\":
   ↪  true,\n   \"description\": \"Your account email.\",\n ...   }\n ],\n \"response_schemas\": {\n
   ↪  \"success\": {\n   \"token_type\": \"string\",\n   \"access_token\": \"string\"\n  },\n
   ↪  \"failure\": {\n   \"message\": \"string\"\n  }\n }\n}"
  },
  {
    "role": "assistant",
    "content": "I need the supervisor's username and password. Lets see if any app can help me get
   ↪  that.\n\n<answer>\nprint(apis.api_docs.show_app_descriptions())\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{'api_docs': 'An app to search and explore API documentation.', 'supervisor':
   ↪  \"An app to access supervisor's personal information, account credentials, addresses, payment
   ↪  cards, and manage the assigned task.\", 'amazon': 'An online shopping app to buy products and
   ↪  manage orders, returns, etc.', 'phone': 'An app to find and ...  'A bill splitting app to track
   ↪  and split expenses with people.', 'simple_note': 'A note-taking app to create and manage notes',
   ↪  'todoist': 'A task management app to manage todo lists and collaborate on them with others.'}"
  },
  {
    "role": "assistant",
    "content": "Looks like the supervisor app could help me with
   ↪  that.\n\n<answer>\nprint(apis.api_docs.show_api_descriptions(app_name='supervisor'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n  ...\n  \"show_account_passwords : Show your supervisor's account
   ↪  passwords.\"\n  ...\n]"
  },
  {
    "role": "assistant",
    "content": "I can use `show_account_passwords` to get the passwords. So I will call that and save it
   ↪  in a variable for future reference.\n<answer>\npasswords=apis.supervisor.show_account_passwords(⌋
   ↪  )\nprint(passwords)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n  {\n    \"account_name\": \"spotify\",\n    \"password\":
   ↪  \"dummy_spotify_pass\"\n  },\n  {\n    \"account_name\": \"file_system\",\n    \"password\":
   ↪  \"dummy_fs_pass\"\n  }\n]"
  },
  {
    "role": "assistant",
    "content": "So the Spotify password is an entry in the `passwords` list with the
   ↪  account_name=spotify. I can use it to login now with the provided email address:
   ↪  mel.bailey@gmail.com.\n\n<answer>\nspotify_password = [account_password[\"account_name\"] ==
   ↪  \"spotify\" for account_password in passwords][0][\"password\"]\nlogin_result =
   ↪  apis.spotify.login(username='mel.bailey@gmail.com',
   ↪  password=spotify_password)\nprint(login_result)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"token_type\": \"Bearer\",\n \"access_token\": \"fake_access_token\"\n}"
  },
  {
    "role": "assistant",
    "content": "First, let's check the available APIs for the Spotify
   ↪  app.\n<answer>\nprint(apis.api_docs.show_api_descriptions(app_name='spotify'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n...\n{\n  \"name\": \"show_playlist_library\",\n  \"description\": \"Get a
   ↪  list of playlists in the user's playlist library.\"\n},\n{\n  \"name\": \"show_playlist\",\n
   ↪  \"description\": \"Get detailed information about a specific playlist. You can view your own
   ↪  playlists or others' playlists if they are public.\"\n },\n...\n]"
  },
  {
    "role": "assistant",
    "content": "The `show_playlist_library` API seems to be the one we need. Let's check its
   ↪  specifications.\n<answer>\nprint(apis.api_docs.show_api_doc(app_name='spotify',
   ↪  api_name='show_playlist_library'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"app_name\": \"spotify\",\n \"api_name\": \"show_playlist_library\",\n
   ↪  \"path\": \"/private_playlists\",\n \"method\": \"GET\",\n \"description\": \"Get a list of
   ↪  playlists in the user's playlist library.\",\n \"parameters\": [\n  {\n   \"name\":
   ↪  \"access_token\",\n   \"type\": \"string\",\n   \"required\": true, ...
   ↪  \"2019-01-01T00:00:00\",\n   \"is_public\": true,\n   \"rating\": 0.0,\n   \"like_count\": 1,\n
   ↪  \"owner_email\": \"user@example.com\",\n   \"playlist_id\": 1,\n   \"song_ids\": [\n    1\n   ]\n
   ↪  }\n ]\n}"
  },
  {
    "role": "assistant",
    "content": "I need to page through all the playlists to get the list of playlists and save it in
   ↪  `playlists`.\n<answer>\npage_index = 0\nplaylists = []\nwhile page_index < 10:\n  playlist_page =
   ↪  apis.spotify.show_playlist_library(access_token=spotify_access_token, page_index=page_index)\n
   ↪  if playlist_page:\n    playlists.extend(playlist_page)\n    page_index += 1\n  else:\n
   ↪  break\nnum_playlists = len(playlists)\nprint(num_playlists)\n\n</answer>"
  },
  { "role": "user", "content": "Output:\n23" },
  {
    "role": "assistant",
    "content": "Now that the task is completed, I can call apis.supervisor.complete_task(). Since this
   ↪  task has an answer to be returned, I will pass that as an
   ↪  argument.\n\n<answer>\napis.supervisor.complete_task(answer=num_playlists)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\nMarked the active task
   ↪  complete.\n\n\n----------------------------------------------\n\n**Key instructions**:\n(1) Make
   ↪  sure to start code blocks with <answer> and end them with </answer>.\n\n(2) Remember you can use
   ↪  the variables in your code in subsequent code blocks.\n\n(3) Remember that the email addresses,
   ↪  access tokens and variables (e.g. spotify_password) in the example above are not valid
   ↪  anymore.\n\n(4) You can use the \"supervisor\" app to get information about my accounts and use
   ↪  the \"phone\" app to get information about friends and family.\n\n(5) Always look at API
   ↪  specifications (using apis.api_docs.show_api_doc) before calling an API.\n\n(6) Write small
   ↪  chunks of code and only one chunk of code in every step. Make sure everything is working
   ↪  correctly before making any irreversible change.\n\n(7) Many APIs return items in \"pages\". Make
   ↪  sure to run through all the pages by looping over `page_index`.\n\n(8) Once you have completed
   ↪  the task, make sure to call apis.supervisor.complete_task(). If the task asked for some
   ↪  information, return it as the answer argument, i.e. call
   ↪  apis.supervisor.complete_task(answer=<answer>). Many tasks do not require an answer, so in those
   ↪  cases, just call apis.supervisor.complete_task() i.e. do not pass any argument.\n\nUsing these
   ↪  APIs, now generate code to solve the actual task:\n\nMy name is: Adrian Burton. My personal email
   ↪  is adrian_burt@gmail.com and phone number is 5025534184.\nTask: The \"~/pictures/\" directory in
   ↪  my file system has photo files organized in sub-directories for each vacation spot. Compress them
   ↪  and save them in \"~/pictures/vacations/<vacation_spot>.zip\" for each vacation spot, and then
   ↪  delete all vacation spot sub-directories. Here, <vacation_spot> is the name of the vacation spot
   ↪  as it appears in the sub-directory name."
  },
< PROMPT END >
  {
    "role": "assistant",
    "content": "First, I need to explore the file system to see what vacation spot sub-directories exist
   ↪  in ~/pictures/.\n<answer>\nprint(apis.api_docs.show_api_descriptions(app_name='file_system'))\n<┘
   ↪  /answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n {\n  \"name\": \"show_account\",\n  \"description\": \"Show your account
   ↪  information. Unlike show_profile, this includes private information.\"\n },\n {\n  \"name\":
   ↪  \"signup\",\n  \"description\": \"Sign up to create account.\"\n },\n {\n  \"name\":
   ↪  \"delete_account\",\n  \"description\": \"Delete your account.\"\n },\n {\n   ...
   ↪  \"description\": \"Compress a directory.\"\n },\n {\n  \"name\": \"decompress_file\",\n
   ↪  \"description\": \"Decompress a compressed file.\"\n }\n]\n\n"
  },
  {
    "role": "assistant",
    "content": "I need to login to the file system first, then list the vacation spot sub-directories in
   ↪  ~/pictures/.\n<answer>\nprint(apis.api_docs.show_api_doc(app_name='file_system',
   ↪  api_name='login'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"app_name\": \"file_system\",\n \"api_name\": \"login\",\n \"path\":
   ↪  \"/auth/token\",\n \"method\": \"POST\",\n \"description\": \"Login to your account.\",\n
   ↪  \"parameters\": [\n  {\n   \"name\": \"username\",\n   \"type\": \"string\",\n   \"required\":
   ↪  true,\n   \"description\": \"Your account email.\",\n    ...  }\n ],\n \"response_schemas\": {\n
   ↪  \"success\": {\n   \"access_token\": \"string\",\n   \"token_type\": \"string\"\n  },\n
   ↪  \"failure\": {\n   \"message\": \"string\"\n  }\n }\n}\n\n"
  },
  {
    "role": "assistant",
    "content": "I need to get the file system password from the supervisor app to
   ↪  login.\n<answer>\npasswords = apis.supervisor.show_account_passwords()\nfs_password =
   ↪  next((acc[\"password\"] for acc in passwords if acc[\"account_name\"] == \"file_system\"),
   ↪  None)\nlogin_result = apis.file_system.login(username='adrian_burt@gmail.com',
   ↪  password=fs_password)\naccess_token =
   ↪  login_result[\"access_token\"]\nprint(access_token)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\neyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJmaWxlX3N5c3RlbSthZHJpYW5fYnVydE⌋
   ↪  BnbWFpbC5jb20iLCJleHAiOjE2ODQ0MTI0MDR9.jwWfOmxx6OLyIf4nOpGhvG3QjZ1LQYppQ9ilxRCEucI\n\n"
  },
  {
    "role": "assistant",
    "content": "Now I'll list the contents of the ~/pictures/ directory to find vacation spot
   ↪  sub-directories.\n<answer>\nprint(apis.api_docs.show_api_doc(app_name='file_system',
   ↪  api_name='show_directory'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"app_name\": \"file_system\",\n \"api_name\": \"show_directory\",\n
   ↪  \"path\": \"/directory\",\n \"method\": \"GET\",\n \"description\": \"Show a list of files and/or
   ↪  sub-directories, optionally recursively, in a directory.\",\n \"parameters\": [\n  {\n
   ↪  \"name\": \"access_token\",\n   \"type\": \"string\",\n   \"required\": true,\n
   ↪  \"description\": \"Access token obtained from file_system app login.\",\n   \"default\":  ...
   ↪  \"response_schemas\": {\n  \"success\": [\n   \"string\"\n  ],\n  \"failure\": {\n   \"message\":
   ↪  \"string\"\n  }\n }\n}\n\n"
  },
  {
    "role": "assistant",
    "content": "I'll list only the sub-directories in ~/pictures/ to find the vacation spot
   ↪  directories.\n<answer>\nvacation_dirs =
   ↪  apis.file_system.show_directory(access_token=access_token, directory_path=\"~/pictures/\",
   ↪  entry_type=\"directories\", recursive=False)\nprint(vacation_dirs)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\"/home/adrian/pictures/vacations/\"]\n\n"
  },
  {
    "role": "assistant",
    "content": "I see there's a vacations directory, but I need to check what's inside it to find the
   ↪  vacation spot sub-directories.\n<answer>\nvacation_spot_dirs =
   ↪  apis.file_system.show_directory(access_token=access_token,
   ↪  directory_path=\"~/pictures/vacations/\", entry_type=\"directories\",
   ↪  recursive=False)\nprint(vacation_spot_dirs)\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n[\n \"/home/adrian/pictures/vacations/bali/\",\n
   ↪  \"/home/adrian/pictures/vacations/maui/\",\n \"/home/adrian/pictures/vacations/santorini/\",\n
   ↪  \"/home/adrian/pictures/vacations/seoul/\"\n]\n\n"
  },
  {
    "role": "assistant",
    "content": "Now I need to compress each vacation spot directory and save it as a zip file in the
   ↪  vacations directory, then delete the original
   ↪  directories.\n<answer>\nprint(apis.api_docs.show_api_doc(app_name='file_system',
   ↪  api_name='compress_directory'))\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\n{\n \"app_name\": \"file_system\",\n \"api_name\": \"compress_directory\",\n
   ↪  \"path\": \"/directory/compress\",\n \"method\": \"POST\",\n \"description\": \"Compress a
   ↪  directory.\",\n \"parameters\": [\n  {\n   \"name\": \"directory_path\",\n   \"type\":
   ↪  \"string\",\n   \"required\": true,\n   \"description\": \"Path of ...  \"message\":
   ↪  \"string\",\n   \"compressed_file_path\": \"string\"\n  },\n  \"failure\": {\n   \"message\":
   ↪  \"string\"\n  }\n }\n}\n\n"
  },
  {
    "role": "assistant",
    "content": "I'll compress each vacation spot directory to a zip file in the vacations directory with
    ↪  the vacation spot name, then delete the original directories.\n<answer>\nfor vacation_dir in
    ↪  vacation_spot_dirs:\n    vacation_spot = vacation_dir.split('/')[-2]  # Extract vacation spot
    ↪  name from path\n    zip_path = f\"~/pictures/vacations/{vacation_spot}.zip\"\n    result =
    ↪  apis.file_system.compress_directory(\n        access_token=access_token,\n
    ↪  directory_path=vacation_dir,\n        compressed_file_path=zip_path,\n
    ↪  delete_directory=True,\n        overwrite=True\n    )\n    print(f\"Compressed {vacation_dir} to
    ↪  {zip_path}: {result}\")\n</answer>"
  },
  {
    "role": "user",
    "content": "Output:\nCompressed /home/adrian/pictures/vacations/bali/ to
    ↪  ~/pictures/vacations/bali.zip: {'message': 'Directory compressed.', 'compressed_file_path':
    ↪  '/home/adrian/pictures/vacations/bali.zip'}\nCompressed /home/adrian/pictures/vacations/maui/ to
    ↪  ~/pictures/vacations/maui.zip: {'message': 'Directory compressed.', ... seoul.zip: {'message':
    ↪  'Directory compressed.', 'compressed_file_path':
    ↪  '/home/adrian/pictures/vacations/seoul.zip'}\n\n"
  },
  {
    "role": "assistant",
    "content": "The task has been completed successfully - all vacation spot directories have been
    ↪  compressed to zip files and the original directories
    ↪  deleted.\n<answer>\napis.supervisor.complete_task()\n</answer>"
  }
]

REWARD: 1.0
```