AgentGym: Evaluating and Training Large Language Model-based Agents across Diverse Environments

Zhiheng Xi[1*], **Yiwen Ding**[1*], **Wenxiang Chen**[1*], **Boyang Hong**[1], **Honglin Guo**[1], **Junzhe Wang**[1], **Xin Guo**[1], **Dingwen Yang**[1], **Chenyang Liao**[1], **Wei He**[1], **Songyang Gao**[1], **Lu Chen**[1], **Rui Zheng**[1], **Yicheng Zou**[1], **Tao Gui**[2,3,4†], **Qi Zhang**[1,2], **Xipeng Qiu**[1], **Xuanjing Huang**[1,2], **Zuxuan Wu**[1,2], **Yu-Gang Jiang**[1,2]

[1]School of Computer Science, Fudan University,
[2]Institute of Trustworthy Embodied Artificial Intelligence, Fudan University,
[3]Institute of Modern Languages and Linguistics, Fudan University, [4]Pengcheng Laboratory
zhxi22@m.fudan.edu.cn, tgui@fudan.edu.cn

Abstract

Large language models (LLMs) have emerged as a promising foundation to build generally-capable agents (LLM-based agents) that can handle multi-turn decision-making tasks across various environments. However, the community lacks a unified interactive framework that covers diverse environments for comprehensive evaluation of agents, and enables exploration and learning for their self-improvement. To address this, we propose AgentGym, a framework featuring 7 real-world scenarios, 14 environments, and 89 tasks for unified, real-time, and concurrent agent interaction. We construct expanded instruction set, high-quality trajectories, and comprehensive benchmarking suite for developing LLM-based agents. Moreover, AgentGym supports interactive exploration and learning for agents through multi-turn interactions and real-time feedback. Based on AgentGym, we take the initial step to develop LLM-based agents that can handle diverse tasks via methods like self-improvement or reinforcement learning. Experimental results show that the trained agents can achieve results comparable to commercial models. We hope our work can help the community develop more advanced LLM-based agents. We release the code, dataset, benchmark, and checkpoints at https://agentgym.github.io/.

1 Introduction

Developing agents capable of performing a wide spectrum of tasks across various environments at human-level has been a long-standing goal for AI community (Wooldridge and Jennings, 1995; Silver et al., 2017, 2018; Reed et al., 2022; Xi et al., 2025). Recently, large language models (LLMs) are considered a promising foundation for constructing such generalist agents due to their generalized abilities (OpenAI, 2023; Anthropic, 2024; Anil et al., 2023), and many efforts have been made in this realm to develop generally-capable LLM-based agents (Xi et al., 2025; Wang et al., 2024).

Frameworks	Env.	Inter. Fra.	Traj.	Exploration
AgentBench (Liu et al., 2023a)	8	Eval	No	No
AgentBoard (Ma et al., 2024)	12	Eval	No	No
AgentOhana (Zhang et al., 2024)	10	No	Yes	No
Pangu-Agent (Christianos et al., 2023)	6	No	Yes	Single-Env
AgentGym (Ours)	14	Eval & Train	Yes	Multi-Env

Table 1: Comparison of AgentGym with other frameworks in several aspects: the number of environments, presence of an interactive framework and its usage, availability of trajectory sets, support for interactive exploration and learning, and the corresponding mode.

Developing advanced generally-capable agents requires diverse environments that provide real-time feedback through a unified interface, facilitating comprehensive evaluation, extensive exploration, and continuous self-improvement (Standish, 2003; Langdon, 2005; Taylor et al., 2016; Fan et al., 2022). Additionally, high-quality trajectory sets and a broad range of task instructions are essential for training these agents (Fan et al., 2022; Song et al., 2024). The former equips agents with prior knowledge and initial interaction capabilities, while the latter expands their exploration space. However, the community currently lacks a unified interactive framework that fulfills these requirements (Table 1). Existing frameworks mainly focus on constructing benchmarks (Liu et al., 2023a; Ma et al., 2024), while others collect expert trajectories for supervised fine-tuning (SFT) (Zhang et al., 2024; Christianos et al., 2023). They are unable to support continuous, real-time interactive evaluation and training, and thus are insufficient to meet the community's needs (Xi et al., 2025).

To this end, we propose a new interactive framework, AgentGym, to support the comprehensive evaluation and training of LLM-based agents

[*] Equal contribution.
[†] Corresponding authors.

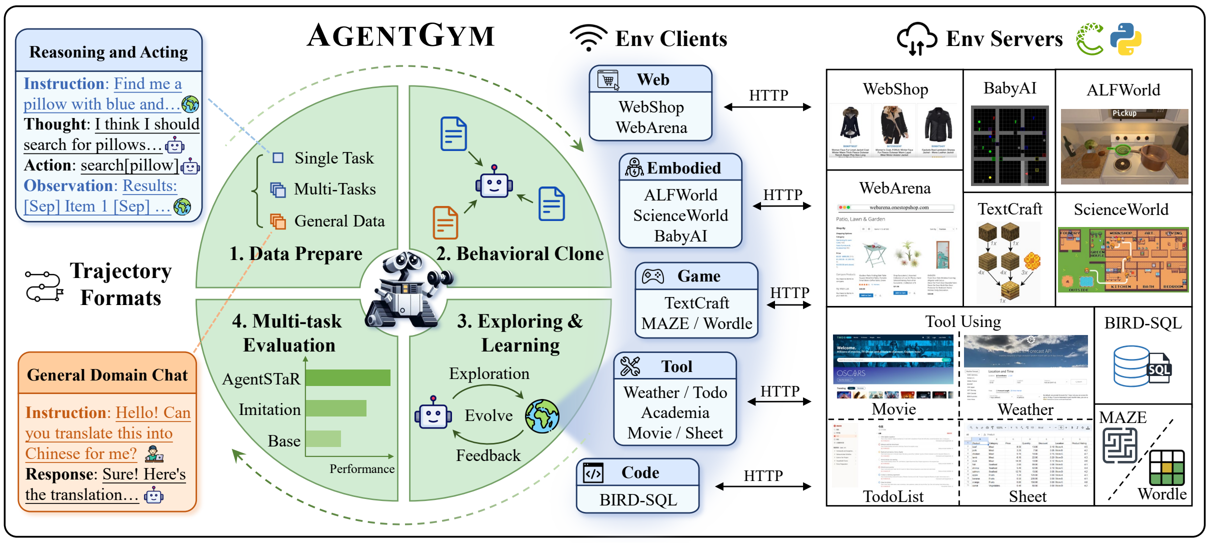


Figure 1: Overview of the AGENTGYM framework. It covers fourteen environments spanning diverse scenarios. It adopts a decoupled client-server architecture for unified and concurrent agent-environment interaction. AGENTGYM also includes expanded instructions, a comprehensive benchmark suite AGENTEVAL, and the high-quality trajectory set AGENTTRAJ (-L). We also investigate exploration-learning paradigms to explore the agent's self-improvement across various environments.

across diverse environments (See Figure 1). In this framework, we include 7 real-world scenarios: web navigation, text games, house-holding tasks, digital games, embodied tasks, tool-using, and programming. A total of 14 related environments and 89 relevant tasks are incorporated. To achieve unified, real-time, and concurrent agent interaction, we standardize task specifications, environment settings, and the observation/action spaces. Through our architectural design, agents can interact with different environments via a unified interface, receive feedback, and thus facilitate evaluation, data collection, and agent exploration and learning.

Based on this framework, we construct instructions from various environments and tasks, expanding them through rule-based strategies and AI-based techniques such as self-instruct (Wang et al., 2023) and instruction evolution (Xu et al., 2023). Subsequently, we leverage several principles to construct a benchmark suite called AGENTEVAL to comprehensively evaluate LLM-based agents. Next, we use a gather-and-filter pipeline to obtain a trajectory set called AGENTTRAJ, containing 6,130 high-quality trajectories. This set is used to train a base agent with basic capabilities and prior knowledge, which then bootstraps further agent exploration and learning. We also collect a larger trajectory set, AGENTTRAJ-L, containing 14,485 trajectories, with the same pipeline for stronger SFT performance.

To develop stronger agents that perform well across multiple tasks, we further investigate their exploration and learning across diverse environments. We investigate self-improvement methods based on the RL as Inference framework (Dayan and Hinton, 1997) and demonstrate their effectiveness through experiments, showing that agents trained using open-source models can tackle a wide range of tasks and achieve performance comparable to commercial models. Additionally, we experiment with online reinforcement learning (RL) methods and show their limitations in developing satisfactory generally-capable LLM-based agents.

In summary, our main contributions are:

1. We propose AgentGym, a new interactive framework that includes diverse scenarios and environments to comprehensively evaluate LLM-based agents and develop agents that perform well across multiple environments.

2. We construct a large dataset of task instructions and trajectories through various methods, aiding the community in developing agents. We also construct a comprehensive benchmark suite to evaluate LLM-based agents.

3. Based on AgentGym, we take the initial step to investigate LLM-based agents' exploration and learning across multiple environments for self-improvement. We carry out detailed experiments to validate the effectiveness of our

framework, dataset, and methods. We hope our work can provide support and insights to the LLM-based agents community.

2 Related Work

Frameworks for evaluating LLM-based agents. With the development of LLMs (OpenAI, 2023; Anil et al., 2023), developing agents based on them has become an important research direction (Xi et al., 2025; Wang et al., 2024). These agents are typically designed to perform multi-turn decision-making tasks (Yao et al., 2023; Aksitov et al., 2023; Chen et al., 2023). To evaluate these agents, researchers have proposed various benchmarks (Yao et al., 2022; Ma et al., 2024) and frameworks (Liu et al., 2023a; Zhou et al., 2023b; Ma et al., 2024). In this work, we include AGENTEVAL that covers more diverse scenarios and environments for providing a comprehensive evaluation.

Prompt-engineering and fine-tuning LLM-based agents. Previous work leverage prompt-engineering-based methods like ReAct (Yao et al., 2023) and PlanAct (Liu et al., 2023b) to develop agents. They prove effective on commercial models like GPT-4, but perform pooly on open-source models (Liu et al., 2023a; Christianos et al., 2023). To address this challenge, a series of work collects expert trajectories to train LLM-based agents through SFT (Zeng et al., 2023; Chen et al., 2023, 2024; Zhang et al., 2024). However, this is often costly and hard to scale, and these methods lack sufficient exploration of the agent in the environment (Yang et al., 2024; Aksitov et al., 2023).

Exploration and learning for LLM-based agents. Another line of work trains LLM-based agents based on exploration and learning with environmental feedback (Zhou et al., 2024; Christianos et al., 2023; Song et al., 2024; Abdulhai et al., 2023a). As a representative method, RL has succeeded in LLM alignment and reasoning (Askell et al., 2021; Bai et al., 2022a; Ouyang et al., 2022; Luong et al., 2024; Zhou et al., 2024), and has been introduced to agent tasks (Zhou et al., 2024). However, due to issues with reward consistency and training stability, they are typically trained in a single environment, making it difficult to train models that can handle multiple tasks (Song et al., 2024; Cao et al., 2024).

Another line of work uses self-improvement, where the model explores the environment to obtain high-reward trajectories and fine-tunes itself based on these trajectories, achieving promising performance in reasoning, coding, and web tasks (Singh et al., 2023; Zelikman et al., 2022; Aksitov et al., 2023; Song et al., 2024; Tao et al., 2024; Tian et al., 2024; Lai et al., 2024). However, like RL-based methods, these works mainly explore training in isolated environment. With AGENTGYM, our work explores agent exploration and learning across multiple environments.

3 Preliminaries

We define the collection of environments as $\mathcal{E}$. For a specific $e \in \mathcal{E}$, we formalize the agent task in the environment as a partially observable Markov decision process (POMDP) $(\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, r)_e$ with instruction space $\mathcal{U}$, state space $\mathcal{S}$, action space $\mathcal{A}$, observation space $\mathcal{O}$, deterministic state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, and reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Given a task instruction u in environment e, the LLM-based agent parameterized by θ generates an action $a_1 \sim \pi_\theta(\cdot|e, u)$ based on its policy π_θ. Then, the state space is transitioned to $s_1 \in \mathcal{S}$, and the agent receives feedback $o_1 \in \mathcal{O}$. Subsequently, the agent interacts with the environment until the task ends or exceeds the maximum number of steps. We adopt ReAct (Yao et al., 2023) to model the outputs of agent, where the LLM-based agent generates a reasoning thought before outputting an action. Thus, at time step t, given the history and current feedback, the agent generates the thought $h_{t+1} \sim \pi_\theta(\cdot|e, u, h_1, a_1, o_1, ..., h_t, a_t, o_t)$ first and the subsequent action $a_{t+1} \sim \pi_\theta(\cdot|e, u, h_1, a_1, o_1, ..., h_t, a_t, o_t, h_{t+1})$. Hence, the trajectory can be represented as:

$$\tau = (h_1, a_1, o_1, ..., o_{T-1}, h_T, a_T) \sim \pi_\theta(\tau|e, u),$$

$$\pi_\theta(\tau|e, u) = \prod_{t=1}^{T} \pi_\theta(h_t, a_t|e, u, c_{t-1}),$$

where T is the number of interaction rounds, and $c_{t-1} = (h_1, a_1, o_1, ..., h_{t-1}, a_{t-1}, o_{t-1})$ represents the interactive history. The reward $r(e, u, \tau) \in [0, 1]$ is computed after the interaction ends or the maximum interactive round number is met.

4 AGENTGYM: Framework Architecture, Instruction Set, Benchmark Suite, and Trajectory Set

AGENTGYM is a framework built for the community to facilitate the evaluation, interactive exploration and learning of generally-capable LLM-based agents. It features diverse interactive environments and tasks with ReAct format (Yao et al., 2023). The framework supports real-time feedback and concurrency, and is easily scalable and extendable. AgentGym also incorporates a diverse set of instructions across multiple environments and tasks. Moreover, it includes a comprehensive benchmark suite, AgentEval, for evaluating agent performance, and two trajectory datasets, AgentTraj and AgentTraj-L, which equip agents with foundational capabilities and prior knowledge.

4.1 Diverse Targeted Environments and Tasks for LLM-based Agents

To ensure the comprehensiveness of the framework, we identify 7 real-world scenarios, including web navigating, text games, house-holding tasks, digital games, embodied tasks, tool-using, and programming. These scenarios are represented by 14 environments and 89 tasks in our framework, as shown in Table 2. More environments details are presented in Appendix A.

Our selection of environments is based on the definition of an LLM-based agent—an agent with a decision-making core that extends its input and actions (Wooldridge and Jennings, 1995; Xi et al., 2025). The three key dimensions are:

Input Side. Agents must process diverse inputs, (e.g., plain text, HTML, code). Our framework integrates textual (ALF, TC), web-based (WS, WA), and coding (BD) environments to evaluate this capability.

Decision-making Side. LLM-based agents require core reasoning and planning capabilities. AgentGym evaluates these through hierarchical task designs. For example, WS demands information extraction and planing when interacting with a web page.

Action Side. Agent outputs can take various forms, such as plain text, code, API calls and embodied actions. We incorporate environments requiring tool use (WT, MV), SQL generation (DB), embodied actions (Baby, ALF), and natural language responses (WD).

4.2 Framework Architecture of AgentGym

We adopt a hierarchical, decoupled architecture in AgentGym to facilitate unified, real-time, and concurrent agent-environment interactions, as illustrated in Figure 4 in Appendix B.

Specifically, at the core of this architecture is the controller, which facilitates interactions between agents and environmental services, providing a unified and encapsulated interface for agents to invoke environmental functions or operations. Additionally, we have implemented user-friendly components such as the evaluator, trainer, and data collection pipeline to support further development of the community. Clients communicate with the servers via the HTTP protocol, which enables real-time interactions. On the server side, we have implemented 14 types of environments and 89 tasks. Researchers can easily develop new environments and add them to AgentGym by encapsulating the aforementioned interfaces. These environments are implemented to offer standardized and parallelizable functions, such as `/createEnv` to create an environment, `/observation` to obtain the current observation from the environment, `/available_actions` to retrieve the currently available actions, `/step` to perform an action, and `/reset` to reset the environment. To accommodate the distinct dependencies across different environments, AgentGym deploys separate services for each environment, ensuring user-friendly deployment and preventing conflicts.

4.3 Database and Benchmark Construction

Regarding database construction, we first gather $20,494$ instructions using rule-based and AI-based generation. Then, we construct a benchmark suite with a size of $1,160$ named AgentEval to evaluate the capabilities of LLM-based agents. As for the trajectory set, we use a gather-and-filter pipeline to obtain $6,130$ high-quality trajectories from 11 environments with various strategies. The set AgentTraj, is used to train a base agent with preliminary abilities and prior knowledge. For a fair comparison, we also perform the same pipeline to get a larger trajectory set, AgentTraj-L, which represents the performance upper bound of SFT.

Instruction collection and generation. We gather $20,494$ instructions across the aforementioned environments using appropriate strategies. ***(i)*** For environments whose original datasets contain sufficient instructions, we use their original instruction sets or subsets (WA, ALF, Sci, AM, ST, BD). ***(ii)*** For certain environments, we generate instructions using rule-based automated pipelines (TC, WS, MZ, WD, Baby). For example, in TC, we first construct rule trees for forging different items and generate instructions of varying difficulty (levels 1-4) based on these rules. For WS, we generate

Env.	Scenario	Task Num.	Eval. Metric	Inst. Size	Eval. Size	Traj. Size	Traj-L Size	Rounds
WebArena (WA, Zhou et al. 2023a)	Web Navigating	3	Success rate	812	20	0	0	–
WebShop (WS, Yao et al. 2022)	Web Navigating	1	Success rate	6910	200	1000	3930	5.1
MAZE (MZ, Abdulhai et al. 2023b)	Text Game	1	Success rate	240	25	100	215	4.3
Wordle (WD, Abdulhai et al. 2023b)	Text Game	1	Success rate	980	25	500	955	4.3
ALFWorld (ALF, Shridhar et al. 2021)	House-holding	6	Success rate	3827	200	500	2420	13.3
SciWorld (Sci, Wang et al. 2022)	Embodied Tasks	30	Reward	2320	200	1000	2120	19.9
BabyAI (Baby, Chevalier-Boisvert et al. 2019)	Embodied Tasks	40	Reward	900	90	400	810	5.7
TextCraft (TC, Prasad et al. 2023)	Digital Game	1	Success rate	544	100	300	374	8.0
Tool-Weather (WT, Ma et al. 2024)	Tool Use	1	Success rate	331	20	160	311	5.5
Tool-Movie (MV, Ma et al. 2024)	Tool Use	1	Success rate	235	20	100	215	4.0
Tool-Academia (AM, Ma et al. 2024)	Tool Use	1	Success rate	20	20	0	0	–
Tool-Sheet (ST, Ma et al. 2024)	Tool Use	1	Reward	20	20	0	0	–
Tool-TODOList (TL, Ma et al. 2024)	Tool Use	1	Success rate	155	20	70	135	5.6
BIRD (BD, Zheng et al. 2023a)	Programming	1	Success rate	3200	200	2000	3000	1.0
Total	–	89	–	20494	1160	6130	14485	–

Table 2: Statistics of AGENTGYM, including scenarios, count of task types, evaluation metric, instruction set size, evaluation set size, trajectory set size (AGENTTRAJ and AGENTTRAJ-L), and the average interactive rounds of each environment in AGENTTRAJ-L.

instructions based on available products by fixing the random seed. For MZ, we randomly select starting points in the maze and construct instructions accordingly. For WD, we fix the seed and generate words for guessing. For Baby, we pass a fixed seed to the generator provided by the environment to generate instructions. ***(iii)*** In environments where instructions are relatively scarce and difficult to construct through rules, we use self-instruct (Wang et al., 2023) and instruction evolution (Xu et al., 2023) methods. These methods provide an LLM (GPT-4-Turbo) with available actions and instruction examples, and query it to generate diverse and challenging instructions that might be needed in real-world scenarios (WT, MV, TL). Note that we manually verify the instructions generated by these AI-based techniques to ensure that they can be successfully completed.

Benchmark construction. To evaluate the general capability of LLM-based agents on diverse tasks, we then construct a benchmark suite with a size of 1160 named AGENTEVAL. Specifically, ***(i)*** for environments that have different task categories or varying difficulty levels, we either uniformly sample test examples from different subsets or use them all (ALF, Sci, Baby, TC); ***(ii)*** for remained environments with existing test sets, we use the original test sets or randomly sample from them (WA, AM, ST, BD); ***(iii)*** for others, we randomly sampled from the collected or augmented instructions (WS, MZ, WD, WT, MV, TL).

Trajectory collecting and filtering. To train a base agent with preliminary abilities and prior knowledge, we collect a training set AGENTTRAJ containing 6130 trajectories from 11 environments with different strategies. ***(i)*** For environments with human annotated trajectories or where the correct action sequences can be obtained using a rule-based solver, we use GPT-4-Turbo to add thought step by step for each action, thus forming outputs in the ReAct-Style (MZ, WD, Sci, BD). ***(ii)*** For environments where only instructions are provided and the correct trajectories are neither available nor can be derived through rules, we annotate the correct trajectories with commercial models (e.g., GPT-4-Turbo) and crowdsourcing. Then, we rigorously filter the trajectories based on rewards and correctness to ensure their quality (WS, ALF, Baby, TC, WT, MV, TL). For a fair comparison, we perform the same gather-and-filter pipeline on all instructions and get a larger training set AGENTTRAJ-L to represent the performance upper bound of SFT.

5 Training Agents that can Handle Diverse Tasks via Exploration and Learning

Here, we introduce exploration & learning methods to train LLM-based agents across environments within AGENTGYM, aiming to provide a solid foundation for further research. We include SFT for initialize a base agent self-improvement-based and RL-based methods.

5.1 Supervised Fine-tuning with Collected Trajectories

Learning everything from scratch through trial and error is inefficient for LLM-based agents (Fan et al., 2022; Song et al., 2024). Hence, we employ the SFT method to train a base agent using AGENTTRAJ. Specifically, the agent learns the basic interaction capability and prior knowledge by imitating

Method	WS	ALF	TC	Sci	Baby	MZ	WD	WT	MV	TL	BD
					Closed-sourced Models & Agents						
DeepSeek-Chat	11.00	51.00	23.00	**16.80**	45.67	4.00	24.00	70.00	70.00	75.00	13.50
Claude-3-Haiku	5.50	0.00	0.00	0.83	1.93	4.00	16.00	55.00	50.00	65.00	13.50
Claude-3-Sonnet	1.50	13.00	38.00	2.78	**79.25**	0.00	36.00	65.00	80.00	80.00	**17.00**
GPT-3.5-Turbo	12.50	26.00	47.00	7.64	71.36	4.00	20.00	25.00	70.00	40.00	12.50
GPT-4-Turbo	**15.50**	**67.50**	**77.00**	14.38	72.83	**68.00**	**88.00**	**80.00**	**95.00**	**95.00**	16.00
					Open-source Models & Agents						
Llama2-Chat-7B	0.50	2.00	0.00	0.83	0.23	0.00	0.00	0.00	0.00	0.00	1.50
Llama2-Chat-13B	1.00	3.50	0.00	0.83	0.10	0.00	0.00	0.00	0.00	0.00	1.50
AgentLM-7B	36.50	71.00	**4.00**	1.63	0.49	**12.00**	4.00	0.00	**5.00**	15.00	5.00
AgentLM-13B	39.50	**73.00**	0.00	2.75	0.45	8.00	0.00	**10.00**	**5.00**	5.00	3.00
AgentLM-70B	**49.50**	67.00	**4.00**	**10.68**	**0.66**	8.00	4.00	0.00	0.00	**40.00**	**7.50**
					Ours (Based on Llama2-Chat-7B)						
AGENTTRAJ-SFT	66.50	77.50	44.00	26.42	69.31	**12.00**	12.00	25.00	5.00	45.00	8.00
AGENTTRAJ-L-SFT	73.50	83.00	60.00	**74.47**	74.19	**12.00**	**36.00**	**45.00**	5.00	65.00	8.50
AGENTSTAR	**76.50**	**88.00**	**64.00**	38.00	**82.70**	**12.00**	12.00	25.00	**60.00**	**70.00**	**9.00**

Table 3: Evaluating results on diverse tasks. AGENTTRAJ-SFT provides a base agent with basic ability and prior knowledge. AGENTTRAJ-L-SFT represents the performance upper limit of SFT in this paper. AGENTSTAR is conducted based on the model after AGENTTRAJ-SFT. The best performance of each part is highlighted in **bold**.

the collected trajectories step-by-step. We maximize the following objective:

$$\mathcal{J}_{\text{SFT}}(\theta) = \mathbb{E}_{(e,u,\tau)\sim\mathcal{D}_{\text{SFT}}}\Big[\log \pi_\theta(\tau|e,u)\Big].$$

Note that the dataset $\mathcal{D}_{\text{SFT}}$ include AGENTTRAJ and a general domain dataset $\mathcal{D}_{\text{general}}$ as in Zeng et al. (2023) to maintain the agent's ability in language generation. And the resulting agent $\pi_{\theta_{\text{base}}}$ serves as a starting point for later exploration and learning across diverse environments.

5.2 Interative Exploration and Learning

Here, we focus on online RL-based (Schulman et al., 2017) and self-improvement-based (Gülçehre et al., 2023; Zelikman et al., 2022) methods through iterative policy optimization.

Online RL for LLM-based agents in isolated environment. In online RL, the objective is to find an optimal policy that maximizes the cumulative reward through interactions with environments (Sutton and Barto, 2018). We apply policy gradient and use the proximal policy optimization (PPO) (Sutton and Barto, 2018) as our basic algorithm as it has proved effective in the area of RLHF for LLMs (Ouyang et al., 2022; Bai et al., 2022b; Zheng et al., 2023b). The general form of the policy gradient is:

$$\mathcal{J}_{\text{RL}}(\theta) = \mathbb{E}_{(e,u,\tau)\sim\mathcal{D}_{\text{RL}}}[r(e,u,\tau)\log \pi_\theta(\tau|e,u)].$$

where $\mathcal{D}_{\text{RL}}$ is the dataset for RL. However, our preliminary experiments show that performing RL across multiple environments often leads to training instability and complicates credit assignment. Therefore, we primarily conduct RL in isolated environments in subsequent experiments.

Self-improvement for LLM-based agents across diverse environments. Due to the limitations of online RL methods, we were inspired by the RL as Inference framework (Dayan and Hinton, 1997) and adopted a self-improvement-based approach (Gülçehre et al., 2023; Zelikman et al., 2022) to enable LLM-based agents to explore and learn in diverse environments, which we call AGENTSTAR. In AGENTSTAR, the process comprises two steps of loop iteration: **Exploration step** and **Learning step** like Singh et al. (2023).

In the m-th exploring iteration, for each environment e, the current policy agent π_{θ^m} interacts with the environment, generating a collection of interaction trajectories $\mathcal{D}^e_m = \{(e,u^j,\tau^j)\ |u^j \sim \mathcal{Q}_e, \tau^j \sim \pi_{\theta^m}(\tau|e,u^j)\}_{j=1}^{|\mathcal{D}^e_m|}$, where $\mathcal{Q}_e$ is an instruction set. Then, the environment returns the reward $r(e,u,\tau)$ for each trajectory and we filter out the trajectories with rewards less than 1. Subsequently, we merge the remained dataset from each environment and the original trajectory set in Section 5.1, resulting in $\mathcal{D}_m = \Big(\bigcup_{e\in\mathcal{E}}\mathcal{D}^e_m\Big)\bigcup\mathcal{D}_{\text{SFT}}$.

In the m-th learning iteration, we use the dataset $\mathcal{D}_m$ to optimize $\pi_{\theta^{m+1}}$ via SFT. The objective is:

$$\mathcal{J}_{\text{learn}}(\theta) = \mathbb{E}_{(e,u,\tau)\sim\mathcal{D}_m}\Big[\log \pi_\theta(\tau|e,u)\Big].$$

We optimize the initial agent π_θ at each iteration, aiming to minimize overfitting and prevent drift from the base agent. In this learning step, the agent

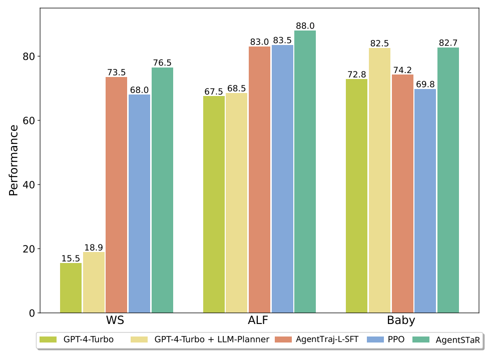


Figure 2: Comparison with exploration-based baselines.

is improved, similar to previous works on LLM reasoning (Zelikman et al., 2022; Singh et al., 2023).

6 Experiments and Discussion

In this section, we perform experiments with AGENTGYM to comprehensively evaluate LLM-based agents. We also leverage our constructed database and the exploration & learning methods for training agents that can handle diverse tasks.

6.1 Experimental Setup

Environments and Tasks. We evaluate the performance of LLM-based agents with the AGENTGYM framework. Main experiments cover the following environments: WS, ALF, Sci, Baby, TC, BD, MZ, WD, TL, WT, and MV.

Baselines. We include commercial models like GPT-3.5-Turbo (Ouyang et al., 2022), GPT-4-Turbo (OpenAI, 2023), Claude 3 (Anthropic, 2024), and DeepSeek-Chat (DeepSeek-AI, 2024). We also include open-source models like Llama-2-Chat (Touvron et al., 2023), and agents trained on expert trajectories, i.e., AgentLM (Zeng et al., 2023).

Implementation Details. Experiments are conducted with eight A100-80GB GPUs. Our main backbone model is Llama-2-Chat-7B. Different environment services are deployed on different ports of the same server. We set the iteration number M to 4. Each instruction is sampled once during the self-improvement process for efficiency. Note that some environments provide dense rewards $r \in [0, 1]$, while others give only binary feedback $r \in \{0, 1\}$. For simplicity and consistency, we follow previous work (Singh et al., 2023) and use binary rewards. We set $r = 0$ for trajectories where $r < 1$, while for those with $r = 1$, we keep it unchanged. See Appendix D for more implementation details. Prompts for each environment are in Appendix F.

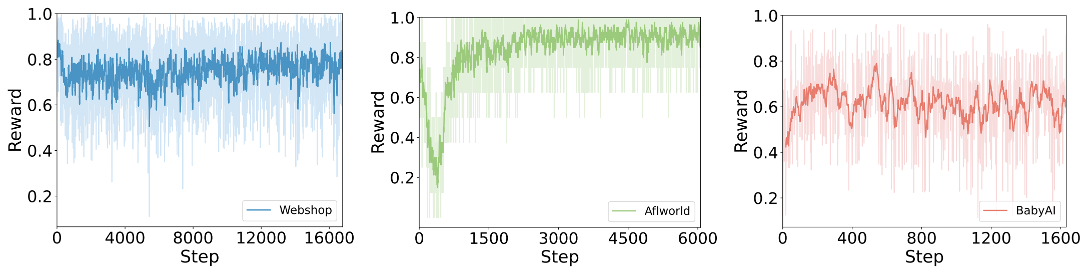
6.2 Main Results

Even commercial models fail to achieve satisfactory performance on all tasks. Experiment results in Table 3 demonstrate that, overall, closed-source commercial models can outperform open-source models. However, even strong closed-source models fail to achieve satisfactory performance on all tasks. For example, GPT-4-Turbo performs only 15.50% and 14.38% on WebShop (Yao et al., 2022) and SciWorld (Wang et al., 2022), respectively. This highlights the effectiveness of AGENTEVAL and the need for developing more generally-capable agents.

Agents fine-tuned on AGENTTRAJ-L can achieve performance comparable to commercial models. Models trained on agent trajectories, like AgentLM (Zeng et al., 2023), can perform on par with GPT-4-Turbo on many tasks, particularly the 70B version. However, they do not match performance on tasks like TextCraft (Prasad et al., 2023) or SciWorld (Wang et al., 2022), which can be attributed to the quality and coverage of data. Instead, the agent trained on AGENTTRAJ-L achieves excellent performance, matching or even surpassing commercial models. This further validates the quality and coverage of our constructed database.

Through exploration and learning with AGENTSTAR, open-source models can handle diverse tasks and surpass SFT on most tasks. Although AGENTSTAR is initialized with limited trajectories, i.e., AGENTTRAJ, it is able to handle diverse tasks and achieve better performance than AGENTTRAJ-L-SFT by continuously exploring, receiving feedback, and learning in the environment. Moreover, it also performs on par with commercial models. For instance, agents with AGENTSTAR outperform GPT-4-Turbo and AGENTTRAJ-L-SFT by 61.00 and 3.00 points on WebShop (Yao et al., 2022), 20.50 and 5.00 points on ALFWorld (Shridhar et al., 2021), and 9.87 and 8.51 points on BabyAI (Chevalier-Boisvert et al., 2019). This validates the superiority and promise of the exploration and learning paradigm in developing agents (Zelikman et al., 2022).

Moreover, we report the number of interactive rounds required to solve the task, demonstrating the efficiency of AGENTSTAR (Appendix E.1). We



(a) Webshop (b) Alfworld (c) BabyAI

Figure 3: Mean training reward during online RL process with PPO.

Method	ALF-OOD	Baby-OOD	AM	ST
Llama2-Chat-7B	0.0	2.2	0.0	0.0
AgentLM-7B	57.7	4.4	10.0	14.3
AGENTTRAJ-SFT	60.8	6.2	20.0	24.3
AGENTTRAJ-L-SFT	64.9	6.1	20.0	25.2
AGENTSTAR	**67.5**	**6.2**	**25.0**	**26.2**

Table 4: Evaluating results on out-of-domain tasks.

Model	WS	ALF	TC	Baby	MZ	WD
Qwen2.5-Max	**35.00**	16.00	34.00	74.20	68.00	60.00
GPT-4o	21.00	**32.00**	**85.00**	80.33	60.00	72.00
DeepSeek-R1	5.00	8.00	70.00	**89.17**	**88.00**	**100.00**

Table 5: Performance comparison of State-of-the-Art models.

also perform qualitative analysis and case study in Appendix E.4).

6.3 Discussion & Analysis

AGENTSTAR can handle more tasks and perform better than online RL and other exploration-based methods. In Section 5, we note that online RL is a fundamental exploration-based learning method. Therefore, we evaluate its performance. We employ PPO algorithm (Schulman et al., 2017) that is widely used in the LLM and run it in each isolated environment, as it tends to experience training instability and exhibit poor performance across multiple environments. Moreover, we introduce LLM-Planner (Song et al., 2023) as a prompt-based exploration-based baseline.

Results in Figure 2 show that: (1) LLM-Planner outperforms GPT-4-Turbo but doesn't surpass AGENTSTAR. This suggests that prompt-based methods has limitations in handling all tasks. (2) PPO performs well but lags behind AGENTSTAR by a significant margin. This indicates that online RL, in addition to being limited to exploration and learning in isolated environments, also performs worse than the training paradigm of AGENTSTAR. We provide a detailed analysis of PPO training dynamics in Figure 3, where we find that the PPO fails to effectively optimize the reward signals across all tasks. Therefore, further research is needed to address this challenge.

Models after exploration and learning demonstrates strong generalization to Out-of-Domain tasks. To evaluate the generalization of trained agents, we conduct experiments on tasks and environments that were not encountered during the exploration and learning phases. Specifically, the used tasks in ALF and Baby are unseen by the agent during training, and the AM and ST environments are completely new for the agent. As shown in Table 4, AGENTSTAR outperforms other baselines on OOD tasks. We find the model can effectively explore unseen tasks and instructions and learn from new experiences, significantly enhancing its generalization capabilities.

Performance analysis of State-of-the-Art models. To further evaluate the capabilities of advanced models, we conduct experiments with Qwen2.5-Max (Team, 2024), GPT-4o (OpenAI, 2024), and DeepSeek-R1 (DeepSeek-AI et al., 2025). As shown in Table 5, DeepSeek-R1 excels in tasks requiring long-term reasoning, such as text-based games (MZ/WD) and embodied tasks (Baby). However, it exhibits notable performance gaps compared to GPT-4o and Qwen2.5-Max on tasks involving complex input processing and precise instruction following. Through further experiments, we find that although DeepSeek-R1 demonstrates strong capabilities in long thinking, its outputs often fail to follow the expected ReAct format and tend to overthink by inaccurately simulating environmental observations. These issues lead to suboptimal performance in structured agent tasks.

Effectiveness on different models with AGENTGYM. To demonstrate the effectiveness of

Model	Method	WS	ALF	Baby	TC
DeepSeek-Coder-1.3B	AgentTraj-SFT	54.0	33.0	68.9	31.0
	AgentTraj-L-SFT	65.0	**62.5**	73.8	37.0
	AgentStar	**67.5**	54.5	**77.3**	**38.0**
Llama2-Chat-13B	AgentTraj-SFT	65.5	81.5	76.6	59.0
	AgentTraj-L-SFT	74.0	85.0	81.1	61.0
	AgentStar	**78.5**	**89.5**	**86.8**	**71.0**

Table 6: Evaluating results on different models.

AgentGym in supporting diverse backbone models, we conduct experiments on Llama-2-13B (Touvron et al., 2023) and DeepSeek-Coder-1.3B (Guo et al., 2024). The experimental results in Table 6 show the same trends as in Section 6.2. This highlights that AgentGym facilitates model training, and its support for real-time, interactive exploration and learning enables consistent improvements regardless of the model architecture.

We include additional ablation and analysis experiments in Appendix E.

7 Conclusion

In this work, we present a new interative framework named AgentGym that encompasses 14 environments and 89 tasks, covering 7 key scenarios for agent evaluation and development. We include a rich database with a high-coverage instruction set, a comprehensive benchmark suite, and a high-quality trajectory set. We take the initial step to investigate training LLM-based agents that can handle various tasks through exploration and learning within diverse environments. We perform extensive evaluation, training, and analysis to show the effectiveness of our AgentGym framework, database and the training pipelines. We hope our work can help the AI community develop more advanced generally-capable LLM-based agents.

Limitations

This paper proposes a new framework named AgentGym for comprehensively evaluating and developing LLM-based agents. It includes an interactive platform with diverse environments and tasks, a high-coverage instruction set, a benchmark suite, and two collections of high-quality trajectories. Additionally, we explore the self-improvement of generally-capable LLM-based agents. Despite the contributions and the fact that our method performs well, our work still has some limitations. **Firstly**, although we have evaluated a wide range of models, we aim to include more open-source and closed-source models in future work to provide better references for the community. **Secondly**, while we have demonstrated that models can master diverse tasks through exploration and learning, we hope to develop more advanced algorithms in the future to further optimize performance. **Thirdly**, although we validate the effectiveness of AgentStar on three different models (Llama2-Chat-7B, Llama-2-Chat-13B, and DeepSeek-Coder-1.3B), we aim to verify it on stronger base models in the future to explore the potential for building more generally-capable agents.

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by the Major Key Project of PCL under Grant PCL2024A06, National Natural Science Foundation of China (No. 62476061,62206057,62076069), Shanghai Rising-Star Program (23QA1400200), Natural Science Foundation of Shanghai (23ZR1403500), Program of Shanghai Academic Research Leader under grant 22XD1401100.

References

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

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

Renat Aksitov, Sobhan Miryoosefi, Zonglin Li, Daliang Li, Sheila Babayan, Kavya Kopparapu, Zachary Fisher, Ruiqi Guo, Sushant Prakash, Pranesh Srinivasan, Manzil Zaheer, Felix X. Yu, and Sanjiv Kumar. 2023. Rest meets react: Self-improvement for multi-step reasoning LLM agent. *CoRR*, abs/2312.10003.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Benjamin Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. 2021. A general language assistant as a laboratory for alignment. *CoRR*, abs/2112.00861.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, Benjamin Mann, and Jared Kaplan. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *CoRR*, abs/2204.05862.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022b. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wichers, Yinxiao Liu, and Lei Meng. 2024. DRLC: reinforcement learning with dense rewards from LLM critic. *CoRR*, abs/2401.07382.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *CoRR*, abs/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. *CoRR*, abs/2403.12881.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. Babyai: A platform to study the sample efficiency of grounded language learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, Zheng Xiong, Yicheng Luo, Jianye Hao, Kun Shao, Haitham Bou-Ammar, and Jun Wang. 2023. Pangu-agent: A fine-tunable generalist agent with structured reasoning. *CoRR*, abs/2312.14878.

Peter Dayan and Geoffrey E Hinton. 1997. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278.

DeepSeek-AI. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In *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*.

Çaglar Gülçehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. 2023. Reinforced self-training (rest) for language modeling. *CoRR*, abs/2308.08998.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: Bootstrap and reinforce A large language model-based web navigating agent. *CoRR*, abs/2404.03648.

WB Langdon. 2005. Pfeiffer–a distributed open-ended evolutionary system. In *AISB*, volume 5, pages 7–13. Citeseer.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2023a.

Agentbench: Evaluating llms as agents. *CoRR*, abs/2308.03688.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2023b. BOLAA: benchmarking and orchestrating llm-augmented autonomous agents. *CoRR*, abs/2308.05960.

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. *Preprint*, arXiv:2401.08967.

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

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

OpenAI. 2024. Gpt-4o system card. *CoRR*, abs/2410.21276.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *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*.

Jan Peters and Stefan Schaal. 2007. Reinforcement learning by reward-weighted regression for operational space control. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, volume 227 of *ACM International Conference Proceeding Series*, pages 745–750. ACM.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. Adapt: As-needed decomposition and planning with language models. *CoRR*, abs/2311.05772.

Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A generalist agent. *Trans. Mach. Learn. Res.*, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2021. 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.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin F. Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *CoRR*, abs/2312.06585.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for LLM agents. *CoRR*, abs/2403.02502.

Russell K Standish. 2003. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(02):167–175.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*.

Tim Taylor, Mark A. Bedau, Alastair Channon, David H. Ackley, Wolfgang Banzhaf, Guillaume Beslon, Emily L. Dolson, Tom Froese, Simon J. Hickinbotham, Takashi Ikegami, Barry McMullin, Norman H. Packard, Steen Rasmussen, Nathaniel Virgo, Eran Agmon, Edward Clark, Simon McGregor, Charles Ofria, Glen E. P. Ropella, Lee Spector, Kenneth O. Stanley, Adam Stanton, Christopher Steven Timperley, Anya E. Vostinar, and Michael J. Wiser. 2016. Open-ended evolution: Perspectives from the OEE workshop in york. *Artif. Life*, 22(3):408–423.

Qwen Team. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*.

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

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345.

Ruoyao Wang, Peter A. 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, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 11279–11298. Association for Computational Linguistics.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Michael J. Wooldridge and Nicholas R. Jennings. 1995. Intelligent agents: theory and practice. *Knowl. Eng. Rev.*, 10(2):115–152.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2).

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *CoRR*, abs/2304.12244.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. 2024. React meets actre: When language agents enjoy training data autonomy. *CoRR*, abs/2403.14589.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *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*.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *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*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. Agenttuning: Enabling generalized agent abilities for llms. *CoRR*, abs/2310.12823.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. 2024. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. 2023b. Secrets of RLHF in large language models part I: PPO. *CoRR*, abs/2307.04964.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023a. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854.

Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. 2023b. SOTOPIA: interactive evaluation for social intelligence in language agents. *CoRR*, abs/2310.11667.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024. Archer: Training language model agents via hierarchical multi-turn RL. *CoRR*, abs/2402.19446.

A Details of Environments in AgentGym

WebShop (WS) (Yao et al., 2022). WebShop is an interactive web environment for web shopping. The agents are given instructions, and need to buy a product that matches the specifications. The agents can click a button on the webpage or search for something by the search engine. WebShop contains 12k instructions and provides over one million real products from amazon.com. We select 6910 instructions. For AgentTraj, we collect 1000 trajectories with commercial models (700) and human annotations (300). For AgentTraj-L, we collect 3930 trajectories with commercial models (3430) and human annotations (500). We take the success rate as the evaluation metric and set the maximum round to 10.[1]

WebArena (WA) (Zhou et al., 2023a). WebArena is a realistic and reproducible web environment. It contains four categories: E-commerce platforms, social forum platforms, collaborative development platforms, and content management systems. It supports 12 different web browsing actions. The observation space consists of a web page URL, the opened tabs, and the web page content. Completing tasks in this highly realistic environment requires the agent to possess strong memory, high-level planning, common sense, and reasoning abilities. The reward from the environment is consistent with the original paper. We filter 20 evaluating test instances from the original dataset for three main sub-tasks: Information-seeking, Site Navigation, and Content & configuration operation. We take the success rate as the evaluation metric and set the maximum round to 25.[2]

MAZE (MZ) (Abdulhai et al., 2023b). MAZE is a word game. Agents, acting as players, can know their own position, the goal position, and the directions where there are walls around them. Agents decide to move one square in one of four directions (up, down, left, or right) each time, receiving a reward of -1 for every move until they reach the goal position. We use GPT-4-Turbo to add thoughts to the trajectories sampled by LMRL-Gym and create our dataset. For AgentTraj, we include 100 trajectories. For AgentTraj-L, we include 215 trajectories. We take the success rate as the evaluation metric and set the maximum round to 15.[3]

Wordle (WD) (Abdulhai et al., 2023b). Wordle is a word-guessing game that tests agents' ability to reason at the level of individual letters. Agents guess the target word from a given vocabulary containing some five-letter words. After each guess, agents are told whether each letter in the guessed word is in the target word and whether its position is correct and receive a reward of -1 for each step until they guess the target word or run out of attempts. We take the success rate as the evaluation metric and set the maximum round to 8. We also use GPT-4-Turbo to add thoughts to the trajectories sampled by LMRL-Gym. For AgentTraj, we include 500 trajectories. For AgentTraj-L, we include 955 trajectories.

ALFWorld (ALF) (Shridhar et al., 2021). ALFWorld is a household environment based on TextWorld, where agents need to explore rooms and use common sense reasoning to execute tasks. The action space of ALFWorld includes picking up and placing items, observing surroundings, using furniture, and more. The environment provides feedback on the execution of actions based on predefined logic. We take the success rate as the evaluation metric and set the maximum round to 30. ALFWorld has six types of tasks. We get 3827

[1] https://github.com/princeton-nlp/WebShop/blob/master/LICENSE.md

[2] https://github.com/web-arena-x/webarena/blob/main/LICENSE

[3] https://github.com/abdulhaim/LMRL-Gym/blob/main/LICENSE

instructions from the original work. For AGENTTRAJ, we collect 500 trajectories with commercial models(400) and human annotations (100). For AGENTTRAJ-L, we collect 2420 trajectories with commercial models(1920) and human annotations (500). [4]

SciWorld (Sci) (Wang et al., 2022). ScienceWorld is a benchmark for testing agents' scientific reasoning abilities in a new interactive text environment at the standard elementary science curriculum level. ScienceWorld includes 30 types of tasks, such as using measurement instruments and conducting mechanics experiments. Its action space is task-related, with the environment simulator providing the effects of actions. Because the ScienceWorld repository provides golden paths and existing models cannot achieve high performance, we use GPT-4-Turbo to generate thoughts for golden paths of 22 types of interactions that are not too long. For AGENTTRAJ, we include 1000 trajectories. For AGENTTRAJ-L, we include 2120 trajectories. We take reward as the evaluation metric and set the maximum round to 30.[5]

BabyAI (Baby) (Chevalier-Boisvert et al., 2019). The BabyAI platform is an interactive grid world simulator with 40 instruction-following tasks where the agent is asked to interact with objects. The agent has a limited 7x7 sight of view and can only operate objects in front. The original implementation of BabyAI presents observations in the form of images and low-level actions like "move forward" and "turn left". The implementation from AgentBoard converts graphic observations into textual instructions and expands the action space with high-level actions like "pickup green key 1" and"go through blue locked door 2". The agent receives a non-zero reward discounted by the number of steps when reaching the goal, and 0 otherwise. For AGENTTRAJ, we annotate 400 trajectories of 18 out of all 40 tasks with commercial models. For AGENTTRAJ-L, we annotate 810 trajectories with commercial models. We take reward as the evaluation metric and set the maximum round to 20.[6]

TextCraft (TC) (Prasad et al., 2023). Similar to WordCraft, TextCraft is a text-only environment for crafting Minecraft items. This environment constructs a crafting tree based on Minecraft's crafting recipes, comprising 544 nodes, each representing a target item. In TextCraft, each task provides a specific target item alongside a list of crafting commands generated by the tree. These tasks are structured compositionally, incorporating crafting recipes of varying complexity ranging from 1 to 4 steps. The environment supports three valid actions: craft <item> using <ingredients>, get <item>, and inventory. Each round, the environment checks the agent's actions and returns the execution state. Apart from craftable items and their ingredients, all other items are obtainable from the environment. Agents can get a reward of 1 only upon successfully crafting the target item. We select 100 tasks for the test set and use the remaining tasks for training. For AGENTTRAJ, we annotate 300 trajectories with commercial models (254) and human annotation (46), with every action in the trajectories verified by the environment. For AGENTTRAJ-L, we annotate 374 trajectories with commercial models (299) and human annotation (75). We take the success rate as the evaluation metric and set the maximum round to 20.[7]

Weather (WT) (Ma et al., 2024). The Weather Environment allows LLM agents to utilize a weather tool to access data on temperature, precipitation, and air quality for various locations and time periods. It includes 18 different actions that agents can use to achieve weather-related objectives. This environment leverages Python code to integrate the Open-Meteo API and implement the necessary functions. If the agent's final answer matches the reference answer, it receives a reward of 1; otherwise, it receives a reward of 0. We expand the original dataset of 20 queries to a total of 331 queries by using GPT-3.5-Turbo and GPT-4-Turbo for augmentation using self-instruct and instruction evolution. Finally, we select 20 questions as the evaluating set, leaving the remaining questions as the training set. For AGENTTRAJ, we annotate 160 trajectories with commercial models (140) and human annotators (20). We also refine the annotations with human review to ensure accuracy. For AGENTTRAJ-L, we annotate 311 trajectories with commercial models (230) and human annotators (81). We take the success rate as the evaluation metric and set the maximum round to

[4] https://github.com/alfworld/alfworld/blob/master/LICENSE

[5] https://github.com/allenai/ScienceWorld/blob/main/LICENSE

[6] https://github.com/mila-iqia/babyai/blob/master/LICENSE

[7] https://github.com/archiki/ADaPT/blob/main/LICENSE

10.[8]

Movie (MV) (Ma et al., 2024). The Movie Environment grants LLM agents to utilize the movie tool for accessing cinematic data, including film details, personnel, and production companies. It offers 16 distinct actions that agents can use to achieve various movie-related objectives. This tool integrates the API and data from The Movie Database, implementing the necessary functions to establish its capabilities. If the agent's final answer matches the reference answer, it receives a reward of 1; otherwise, it receives a reward of 0. To enhance the dataset, we expand the original 20 questions to 235 by using GPT-3.5-Turbo and GPT-4-Turbo for query augmentation. GPT-4-Turbo is employed to annotate 100 trajectories in AGENTTRAJ, and the annotations are further corrected through human annotations to ensure accuracy. We also use GPT-4-Turbo to annotate 215 trajectories for AGENTTRAJ-L. We select 20 questions for the evaluating set, with the remaining questions designated as the training set. We take the success rate as the evaluation metric and set the maximum round to 12.

Academia (AM) (Ma et al., 2024). The Academia Environment equips LLM agents with the academic tools to query information related to computer science research, including academic papers and author details. It offers 7 different actions for achieving various academic research objectives. During its development, it utilizes data from the Citation Network Dataset, crafts the necessary functions, and subsequently constructs the Academia tool. If the agent's final answer matches the reference answer, it receives a reward of 1; otherwise, it receives a reward of 0. The original 20 questions are used as the evaluating set. We take the success rate as the evaluation metric and set the maximum round to 12.

TODOList (TL) (Ma et al., 2024). The TodoEnvironment enables LLM agents to query and amend personal agenda data through the todo tool, offering 11 different actions. This tool is implemented based on the TodoList API. If the agent's final answer or operations matches the reference ones, it receives a reward of 1; otherwise, it receives a reward of 0. To enhance the dataset, we expand the original 20 questions to 155 using GPT-3.5-Turbo and GPT-4-Turbo for data augmentation. For AGENTTRAJ, we annotate 70 trajectories with GPT-4-Turbo. For AGENTTRAJ-L, we annotate the queries to get 135 trajectories with GPT-4-Turbo (96) and human annotators (39). The annotations are further refined by human review to ensure accuracy. Finally, we select 20 questions for the evaluating set, with the remaining questions designated as the training set. We take the success rate as the evaluation metric and set the maximum round to 15.

Sheet (ST) (Ma et al., 2024). The Sheet Environment allows LLM agents to use the sheet tool to access and modify spreadsheet data, providing 20 different actions for operating on an Excel sheet. This tool is built upon the Google Sheets API. The reward returned by the environment is based on the similarity between the table manipulated by the agent and the reference table, with a value range of $[0, 1]$. The original 20 questions are used as the evaluating set. We take reward as the evaluation metric and set the maximum round to 15.

BIRD (BD) (Zheng et al., 2023a). Code ability is a crucial aspect of capability for LLM-based agents. In this environment, we focus on database management ability. We wrap the BIRD-SQL dataset and provide a unified API for agents to interact with. BIRD-SQL is a bench for large-scale database-grounded text-to-SQL evaluation. It requires the agent to query a database using a SELECT statement to get the correct answer. It contains 9428 unique problems with a golden answer for training. We select 3200 of them as the instruction set. For AGENTTRAJ, we employ GPT-4-Turbo to add thoughts for 2000 of the training set problems. For AGENTTRAJ-L, we employ GPT-4-Turbo to add thoughts for 3000 of the training set problems. We take success rate as the evaluation metric and the maximum round is 1 as BD is a single-round programming task.[9]

B Platform Architecture of AGENTGYM

The architectural framework of AGENTGYM is illustrated in Figure 4.

[8]https://github.com/hkust-nlp/AgentBoard. The codebase is licensed under an Apache-2.0 License and the dataset is licensed under a GNU General Public License, version 2.

[9]https://github.com/AlibabaResearch/DAMO-ConvAI/tree/main/bird. The bench is under a CC BY-NC 4.0 License.

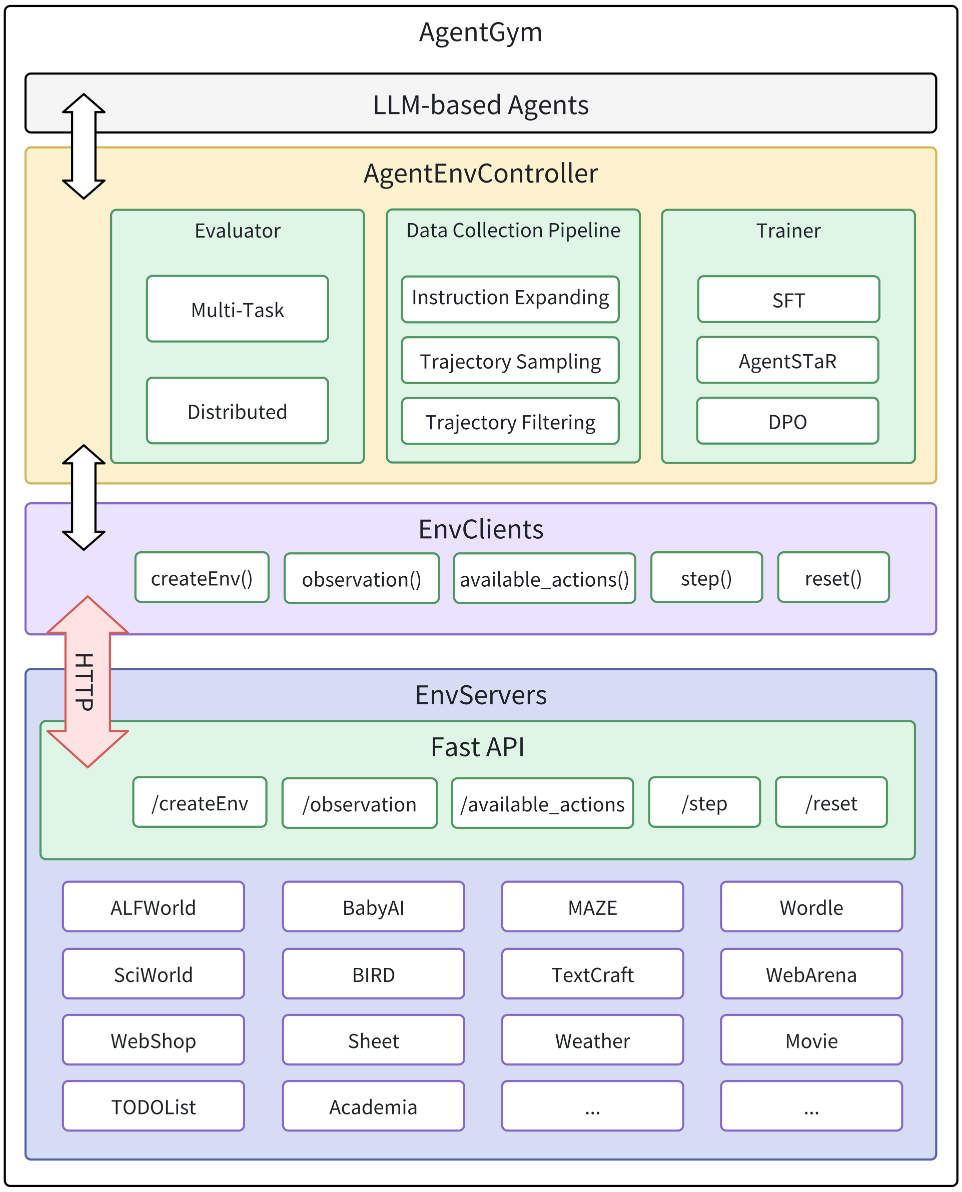


Figure 4: An illustration of the architecture of AGENTGYM.

C Algorithm

The algorithm of AGENTSTAR is summarized in Algorithm 1.

D More Implementation Details

SFT. We train the model for 3 epochs with a learning rate of 1×10^{-5}. The batch size is set to 2, and gradient accumulation is performed over 2 steps. We do not employ weight decay or learning rate warmup.

AGENTSTAR. First, we train a base agent on the AGENTTRAJ set, running SFT for 3 epochs with a learning rate of 1×10^{-5}. Then, we perform the self-improvement phase. In Learning Step, we run 1 epoch per iteration. In Exploration Step, we set the temperature to 0.7 to sample trajectories across environments. We perform a total of M=4 iterations. All other parameters remain the same as in SFT. All experiments are conducted on eight A100-80GB GPUs.

LLM-Planner. LLM-Planner is a prompt-based baseline. In our experiments, we enhance the initial System Prompt with exploration-based guidance. We directly prompt the agent to generate a high-level plan in the first turn, mapping the instruction into subgoals, and let it interact with the environment in a ReAct-style during subsequent turns.

PPO. We use full parameter fine-tuning instead of LoRA tuning. We load three models: the actor model, the reference model, and the critic model. We do not use a reward model, as our environment

Algorithm 1: AGENTSTAR

```
Input: Initialized policy LLM-based agent $\pi_\theta$, environment set $\mathcal{E}$, trajectory subset $\mathcal{D}_s$, full
       instruction set $\mathcal{Q}$, reward function $r$.
Procedure Supervised fine-tuning:
    Maximize objective $\mathcal{J}_{\text{SFT}}(\theta) = \mathbb{E}_{(e,u,\tau)\sim\mathcal{D}_{\text{SFT}}}\Big[\log \pi_\theta(\tau|e,u)\Big]$ to get $\pi_{\theta_{base}}$;
Procedure Evolution :
    $\pi_{\theta^1} \leftarrow \pi_{\theta_{base}}$;
    for iteration $m = 1$ to $M$ do
        // Perform Exploration Step
        $\mathcal{D}_m = \bigcup_{e\in\mathcal{E}} \mathcal{D}_m^e$, where $\mathcal{D}_m^e = \{(e, u^j, \tau^j) \,|u^j \sim \mathcal{Q}_e, \tau^j \sim \pi_{\theta^m}(\tau|e, u^j)\}_{j=1}^{|\mathcal{D}_m^e|}$;
        Compute reward for $\mathcal{D}_m$ with $r$;
        $\mathcal{D}_m \leftarrow \mathcal{D}_m \cup \mathcal{D}_{\text{SFT}}$;
        // Perform Learning Step
        Maximize objective $\mathcal{J}_{\text{learn}}(\theta) = \mathbb{E}_{(e,u,\tau)\sim\mathcal{D}_m}[\log \pi_\theta(\tau|e,u)]$ to get $\pi_{\theta^{m+1}}$;
    end
```

automatically assigns rewards to the agent based on interaction results. We follow the implementation from the TRL library [10], where the actor and critic models share the same backbone. On top of this, we add a trainable value head as the output for the critic model. The learning rate is set to 5×10^{-7}, with a batch size of 1 and gradient accumulation steps of 2. We do not use weight decay or learning rate warmup. We adhere to OpenAI's implementation of the PPO algorithm (Ouyang et al., 2022), where `KL_coef` $= 0.01$, `gamma` $= 1.0$, `lambda` $= 0.95$, and `ppo_epoch` $= 2$. For all environments, we first perform SFT for 1∼2 epochs as a warm-up, followed by PPO training for 5∼10 epochs. To alleviate memory constraints, we employ gradient checkpointing and flash-attention 2 techniques.

DPO. We also use full parameter fine-tuning. During the data sampling phase, we perform two rounds of sampling on the base model to construct DPO training data pairs. Responses with a reward exceeding the expert threshold are labeled as "chosen responses". Responses with a reward gap greater than that of the "chosen responses" are labeled as "rejected responses", and these form the data pairs. In our experiments, the expert threshold is set to 0.9, and the reward gap is set to 0.1. We train for 3 epochs with a learning rate of 5×10^{-7}, a batch size of 2, gradient accumulation steps of 4, weight decay of 0.1, and a warmup ratio of 0.1. Additionally, we include a SFT objective to stabilize the training procedure, following previous work (Lai et al., 2024). Both the DPO and SFT objectives are assigned equal weights.

Evaluation. We set `do_sample = False` during evaluation. When evaluating models that have not been fine-tuned on expert trajectories, we use a few-shot approach; when evaluating models that have been trained on expert trajectories, we use a zero-shot approach.

E More Experiments

E.1 Interactive Rounds in Main Experiments

Interactive rounds reflect the efficiency of an agent in solving tasks. Table 7 shows the interactive rounds of each model/agent across tasks. We also present the evaluation performance in Table 7 for better and clearer illustration. We find that agents trained with AGENTTRAJ-L and AGENTSTAR both demonstrate high efficiency, indicating that they can complete tasks in a small number of rounds. Additionally, we observe a trend: agents that require fewer interactive rounds to complete the same task generally perform better. This may be because underperforming agents often struggle to find the optimal path to achieve the final goal or exceed the maximum number of rounds. For example, in ALFWorld and BabyAI, AGENTSTAR achieves the best performance as well as the fewest interactive rounds.

E.2 More Ablation Studies

Ablation on sample number K. In the exploration step, we perform sampling on each instruction once per iteration. Here, we conduct ablation on sample number K with four tasks. The results in

[10]https://github.com/huggingface/trl

Method	WS	ALF	TC	Sci	Baby	MZ	WD	WT	MV	TL	BD
					Closed-source Models & Agents						
DeepSeek-Chat	11.00	51.00	23.00	**16.80**	45.67	4.00	24.00	70.00	70.00	75.00	13.50
	6.9	20.4	15.1	20.7	11.7	14.5	5.2	6.1	5.9	4.4	1.0
Claude-3-Haiku	5.50	0.00	0.00	0.83	1.93	4.00	16.00	55.00	50.00	65.00	13.50
	8.0	30.0	20.0	29.8	19.9	14.4	5.7	7.3	6.0	4.0	1.0
Claude-3-Sonnet	1.50	13.00	38.00	2.78	**79.25**	0.00	36.00	65.00	80.00	80.00	**17.00**
	9.5	27.9	14.6	28.7	6.6	15.0	5.2	6.9	5.1	4.5	1.0
GPT-3.5-Turbo	12.50	26.00	47.00	7.64	71.36	4.00	20.00	25.00	70.00	40.00	12.50
	4.9	25.2	13.1	16.5	8.4	14.4	5.3	6.6	4.6	3.4	1.0
GPT-4-Turbo	**15.50**	**67.50**	**77.00**	14.38	72.93	**68.00**	**88.00**	**80.00**	**95.00**	**95.00**	16.00
	8.2	18.3	9.9	18.1	9.1	9.0	4.0	6.0	4.5	4.0	1.0
					Open-source Models & Agents						
Llama2-Chat-7B	0.50	2.00	0.00	0.83	0.23	0.00	0.00	0.00	0.00	0.00	1.50
	6.4	22.6	14.5	27.5	9.5	15.0	6.0	9.9	12.0	15.0	1.0
Llama2-Chat-13B	1.00	3.50	0.00	0.83	0.10	0.00	0.00	0.00	0.00	0.00	1.50
	8.1	19.6	16.5	21.3	10.9	13.4	6.0	10.0	12.0	15.0	1.0
AgentLM-7B	36.50	71.00	**4.00**	1.63	0.49	**12.00**	4.00	0.00	**5.00**	15.00	5.00
	4.7	17.7	19.4	28.5	7.5	13.9	2.0	8.3	11.7	10.6	1.0
AgentLM-13B	39.50	**73.00**	0.00	2.75	0.45	8.00	0.00	**10.00**	**5.00**	5.00	3.00
	4.8	17.8	19.4	28.5	7.6	13.9	6.0	6.6	10.7	8.4	1.0
AgentLM-70B	**49.50**	67.00	**4.00**	**10.68**	**0.66**	8.00	4.00	0.00	0.00	**40.00**	**7.50**
	4.9	18.5	18.8	28.2	6.3	13.9	5.2	6.6	11.6	6.7	1.0
					Ours						
AgentTraj-SFT	66.50	77.50	44.00	26.42	69.31	**12.00**	12.00	25.00	5.00	45.00	8.00
	5.6	16.4	13.7	21.3	6.7	14.3	5.9	6.2	10.8	5.4	1.0
AgentTraj-L-SFT	73.50	83.00	60.00	**74.47**	74.19	**12.00**	**36.00**	**45.00**	5.00	65.00	8.50
	5.5	16.1	14.3	29.3	6.2	14.3	5.7	6.4	10.2	5.0	1.0
AgentStar	**76.50**	**88.00**	**64.00**	38.00	**82.70**	**12.00**	12.00	25.00	**60.00**	**70.00**	**9.00**
	5.1	14.0	11.8	18.9	4.3	13.8	5.7	5.9	3.2	5.1	1.0

Table 7: Evaluating performance and interactive rounds on diverse tasks. The first row of each method indicates performance, while the second row of each method shows the number of interaction rounds between the model/agent and the environment.

Table 8 show no significant performance increases with higher K. So we select $K = 1$ for computational efficiency.

Method	WS	ALF	Baby	TC
AgentTraj-SFT	66.5	77.5	69.3	44.0
AgentStar				
-w $K = 1$	77.0	88.0	82.9	65.0
-w $K = 2$	76.0	88.0	83.1	67.0
-w $K = 3$	**78.5**	**89.0**	**83.6**	**68.0**
-w Limited Scope for Exploration	70.0	80.5	70.7	49.0

Table 8: Ablation study on sample number K and the exploration scope with four tasks.

Ablation on exploration scope. In our experiment, we first train a base agent using $\mathcal{D}_{\text{SFT}}$ and then let it explore a wider range of instructions and tasks. We conduct an ablation study on four tasks to see how well the agent evolves with limited instructions as in the SFT phase. Table 8 shows that even in a limited scope, the base agent's performance improves, which may be attributed to more diverse trajectories sampled from the agent. However, the improvement is not significant, indicating that effective evolution needs a more extensive environment.

Ablation on base model selection. In our experiments, we optimize the initial agent at each iteration rather than continuing training from the last iteration's agent. To explore this further, we conduct an ablation study to compare these two training strategies. As shown in Figure 5, continuous fine-tuning provides short-term performance gains but often results in performance degradation in later iterations, likely due to overfitting. In contrast, training from the initial agent ensures more consistent and stable performance.

E.3 Analysis on RL and other exploration-based methods

We have conducted a detailed comparison of AgentStar with other RL methods and

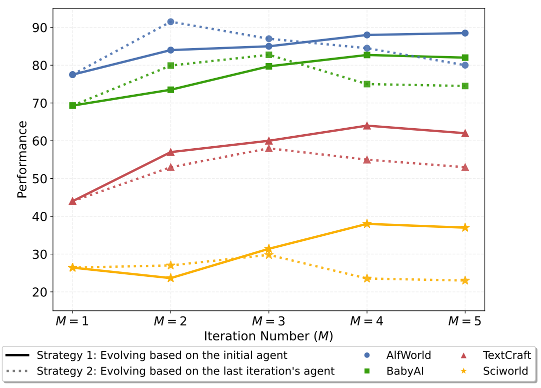


Figure 5: Ablation study regarding the choice of the base model.

exploration-based methods as an additional discussion.

Selection of baselines. Our evaluation is comprehensive and sufficient, including Prompt-based, SFT, offline-RL, and online-RL methods. Experimental results demonstrate that AGENTSTAR achieves superior performance across various tasks when compared with representative algorithms.

Training cost. As shown in Table 9, the training costs for AGENTTRAJ-L-SFT, Reward Weighted Regression (RWR) (Peters and Schaal, 2007), and AGENTSTAR are set as baselines, as they all optimize the policy with SFT loss. In contrast, DPO and PPO methods have significantly higher training costs. DPO requires loading both the actor and reference models and computing probability distributions for chosen and rejected responses. PPO, being an online RL method, involves sampling and policy optimization simultaneously, leading to more intensive training times.

Method	Type	Avg. Training Cost	Accuracy		
			WS	ALF	Baby
LLM-Planner	prompt-based	/	18.9	68.5	82.5
AGENTTRAJ-L-SFT	supervised fine-tuning	1×	73.5	83.0	74.2
RWR	offline-RL	1×	68.0	76.5	82.1
DPO	offline-RL	4.3×	75.0	86.5	78.3
PPO	online-RL	**15**×	68.0	83.5	69.8
AGENTSTAR	offline-RL	1×	**76.5**	**88.0**	**82.7**

Table 9: Comparison between AGENTSTAR and other RL / exploration-based methods

Learning stability. For consistency, we set the smallest unit of the x-axis for training time as an epoch. As shown in Figure 6, it is clear that algorithms optimized with SFT objectives are more stable in performance improvements, leading to faster convergence. While DPO shows significant improvement in the early stages, overfitting occurs quickly as training progresses. PPO, on the other hand, exhibits noticeable instability throughout the training process, with no clear learning trend during the same number of epochs.

Training Reward Curves. Additionally, we provide the mean training reward curves in Figure 3. We also observe that PPO encounters instability and fluctuations in training rewards. This could be due to the standard PPO algorithm, which only uses outcome-based rewards and struggles with optimizing sparse, long-term, and multi-turn trajectories, limiting the model's exploration and learning.

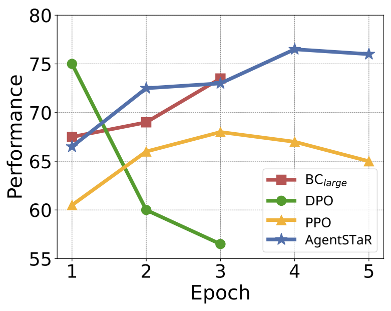
E.4 Case Study

Here, we select three cases to demonstrate the performance comparison before and after the agent evolution, illustrating the effectiveness of AGENTSTAR.

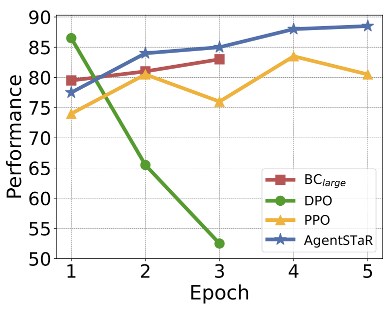
The first case is shown in Figure 7. In this case, the user's instruction is "Find me slim fit, straight leg men's pants with elastic waist, long sleeve, relaxed fit for everyday wear with color: black, and size: large, and price lower than 50.00 dollars." Before evolution, the agent can not effectively utilize specific information from the environment's feedback and directly chooses an item that exceeds the target price, resulting in task failure. However, after evolution, the agent is able to engage in multiple rounds of interaction with the environment, accurately parse the details of the items returned by the environment, and select a product with the correct color, size, and price attributes.

The second case comes from the BabyAI environment, as shown in Figure 8. In this environment, the agent's task is to pick up the green box in a room. The agent before evolution cannot effectively understand spatial relationships and fails to perceive that the target object is right in front of it, leading to incorrect decisions. After receiving the positional information returned by the environment, it repeatedly moves forward until it reaches the interaction limit. After evolution, the agent can accurately determine its position and directly execute the correct "pickup green box 1" action.

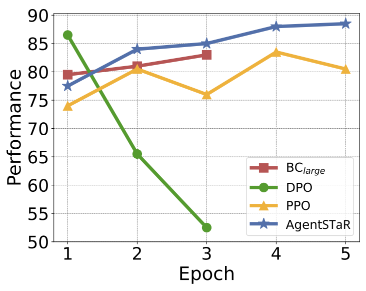
To compare AGENTSTAR with other baselines, we analyze the third case shown in Figure 9. The task is to find a long-lasting, lead-free soy candle within a price range. RWR and DPO baselines fail by selecting the first item without considering the price, while PPO fails by aimlessly clicking "Next Page". In contrast, the agent after evolution accurately parses product details and successfully identifies a suitable item.



(a) Webshop



(b) Alfworld



(c) BabyAI

Figure 6: Learning stability of AGENTSTAR and other exploration-based methods

F Prompt Details

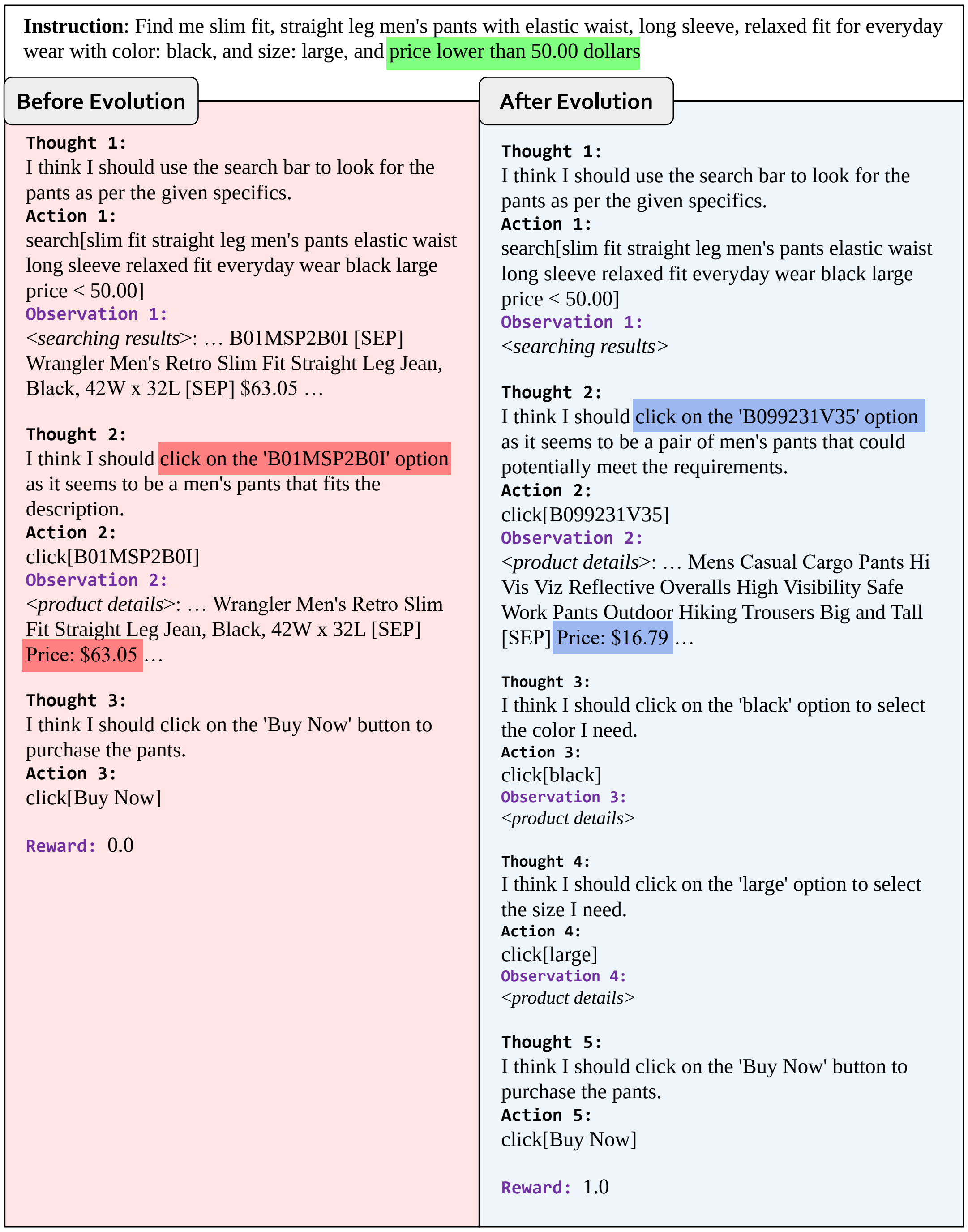
The prompt for each environment and task comprises two components: the system prompt and the instruction. The system prompt provides the initial scenario for each task. The instruction provides specific queries for each task. For consistency, the same prompt template is utilized for human annotation, AI-based annotation of trajectories, and evaluation across all tasks. The prompt details for the WebShop are shown in Table 10. Table 11 presents the specifications for ALFWorld. The TextCraft's prompt details are outlined in Table 12. The prompt details for the SciWorld are shown in Table 13. The prompt details for the BabyAI are shown in Table 14. The prompt details for the MAZE are shown in Table 15. The prompt details for the Wordle are shown in Table 16. The prompt details for the BIRD are shown in Table 17. Table 18 and 19 show the prompt details for WebArena. The prompt details for the Weather are shown in Table 20, 21, 22, 23. The prompt details for the TODOList are shown in Table 24, 25, 26. The prompt details for the Movie are shown in Table 27, 28, 29. The prompt details for the Academia are shown in Table 30, 31. The prompt details for the Sheet are shown in Table 32, 33, 34, 35.

System Prompt:
You are web shopping. I will give you instructions about what to do. You have to follow the instructions. Every round I will give you an observation and a list of available actions, you have to respond to an action based on the state and instruction. You can use search action if the search is available. You can click one of the buttons in clickables. An action should be of the following structure: search[keywords] click[value]. If the action is not valid, perform nothing. Keywords in search are up to you, but the value in click must be a value in the list of available actions. Remember that your keywords in search should be carefully designed. Your response should use the following format:

Thought: I think ...
Action: click[something]

Instruction:
WebShop [SEP] Instruction: [SEP] Find me machine wash women's swimsuits & cover-ups with drawstring closure, elastic waistband, tummy control with color: black, and size: medium, and price lower than 30.00 dollars [SEP] Search

Table 10: Prompt details for WebShop.



Instruction: Find me slim fit, straight leg men's pants with elastic waist, long sleeve, relaxed fit for everyday wear with color: black, and size: large, and price lower than 50.00 dollars

Before Evolution

Thought 1:
I think I should use the search bar to look for the pants as per the given specifics.
Action 1:
search[slim fit straight leg men's pants elastic waist long sleeve relaxed fit everyday wear black large price < 50.00]
Observation 1:
<*searching results*>: … B01MSP2B0I [SEP] Wrangler Men's Retro Slim Fit Straight Leg Jean, Black, 42W x 32L [SEP] $63.05 …

Thought 2:
I think I should click on the 'B01MSP2B0I' option as it seems to be a men's pants that fits the description.
Action 2:
click[B01MSP2B0I]
Observation 2:
<*product details*>: … Wrangler Men's Retro Slim Fit Straight Leg Jean, Black, 42W x 32L [SEP] Price: $63.05 …

Thought 3:
I think I should click on the 'Buy Now' button to purchase the pants.
Action 3:
click[Buy Now]

Reward: 0.0

After Evolution

Thought 1:
I think I should use the search bar to look for the pants as per the given specifics.
Action 1:
search[slim fit straight leg men's pants elastic waist long sleeve relaxed fit everyday wear black large price < 50.00]
Observation 1:
<*searching results*>

Thought 2:
I think I should click on the 'B099231V35' option as it seems to be a pair of men's pants that could potentially meet the requirements.
Action 2:
click[B099231V35]
Observation 2:
<*product details*>: … Mens Casual Cargo Pants Hi Vis Viz Reflective Overalls High Visibility Safe Work Pants Outdoor Hiking Trousers Big and Tall [SEP] Price: $16.79 …

Thought 3:
I think I should click on the 'black' option to select the color I need.
Action 3:
click[black]
Observation 3:
<*product details*>

Thought 4:
I think I should click on the 'large' option to select the size I need.
Action 4:
click[large]
Observation 4:
<*product details*>

Thought 5:
I think I should click on the 'Buy Now' button to purchase the pants.
Action 5:
click[Buy Now]

Reward: 1.0

Figure 7: Case study of WebShop. Before evolution, the agent can not effectively utilize specific information from the environment's feedback and directly chooses an item that exceeds the target price, resulting in task failure. After the evolution, the agent is able to engage in multiple rounds of interaction with the environment, accurately parse the details of the items returned by the environment, and select a product with the correct color, size, and price attributes.

Instruction: Your goal: pick up the green box. In front of you in this room, you can see several objects: The room has walls around you. You are facing a wall 1 steps away. You are not carrying anything. Available actions: ["turn left", "turn right", "check available actions"]

Before Evolution

Thought 1:
I need to find the green box to pick it up. I should check my surroundings to locate it.
Action 1:
check available actions
Observation 1:
You can take the following actions: turn left, turn right, check available actions. Available actions: ["turn left", "turn right", "check available actions"]

Thought 2:
I need to find the green box to pick it up. I should turn left to explore a different direction.
Action 2:
turn left
Observation 2:
<*room details*>: …In front of you in this room, you can see several objects: There is a green box 1 3 steps in front of you…

Thought 3:
I need to pick up the green box to complete my goal. I should move forward to reach it.
Action 3:
move forward
Observation 3:
<*room details*>

Thought 4:
I need to pick up the green box to complete my goal. I should move forward to reach it.
Action 4:
move forward
Observation 4:
<*room details*>

…(more interactions)

Thought 20:
I need to pick up the green box to complete my goal. I should move forward to reach it.
Action 20:
move forward
…(reach max_interaction_rounds)

Reward: 0.0

After Evolution

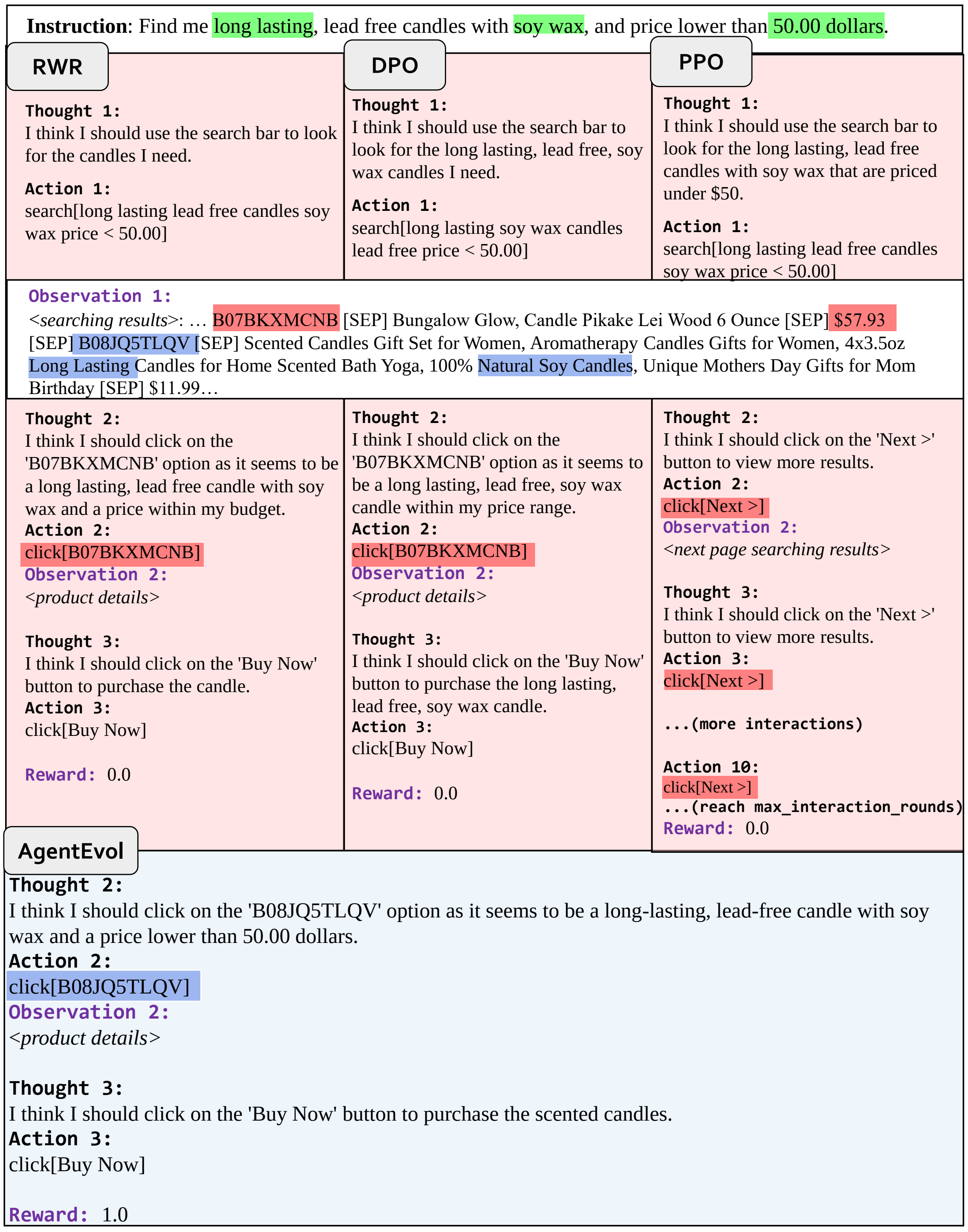
Thought 1:
I need to find the green box in this room.
Action 1:
check available actions
Observation 1:
You can take the following actions: turn left, turn right, check available actions. Available actions: ["turn left", "turn right", "check available actions"]

Thought 2:
I need to turn left to explore the room further and find the green box.
Action 2:
turn left
Observation 2:
In front of you in this room, you can see several objects: There is a red key 1 0 steps in front of you and 1 steps to your left. There is a green box 1 3 steps in front of you and 1 steps to your left. There is a purple key 1 3 steps in front of you and 3 steps to your left. The room has walls around you. You are facing a wall 5 steps away. You are not carrying anything. Available actions: ["turn left", "turn right", "move forward", "pickup red key 1", "pickup green box 1", "pickup purple key 1", "go to red key 1", "go to green box 1", \go to purple key 1", "check available actions"]

Thought 3:
I need to pick up the green box to complete my goal.
Action 3:
pickup green box 1

Reward: 1.0

Figure 8: Case study of BabyAI. The agent before evolution cannot effectively understand spatial relationships and fails to perceive that the target object is right in front of it, leading to incorrect decisions. After receiving the positional information returned by the environment, it repeatedly moves forward until it reaches the interaction limit. After evolution, the agent can accurately determine its position and directly execute the correct "pickup green box 1" action.


Instruction: Find me long lasting, lead free candles with soy wax, and price lower than 50.00 dollars.

RWR

Thought 1:
I think I should use the search bar to look for the candles I need.

Action 1:
search[long lasting lead free candles soy wax price < 50.00]

DPO

Thought 1:
I think I should use the search bar to look for the long lasting, lead free, soy wax candles I need.

Action 1:
search[long lasting soy wax candles lead free price < 50.00]

PPO

Thought 1:
I think I should use the search bar to look for the long lasting, lead free candles with soy wax that are priced under $50.

Action 1:
search[long lasting lead free candles soy wax price < 50.00]

Observation 1:
<searching results>: … B07BKXMCNB [SEP] Bungalow Glow, Candle Pikake Lei Wood 6 Ounce [SEP] $57.93 [SEP] B08JQ5TLQV [SEP] Scented Candles Gift Set for Women, Aromatherapy Candles Gifts for Women, 4x3.5oz Long Lasting Candles for Home Scented Bath Yoga, 100% Natural Soy Candles, Unique Mothers Day Gifts for Mom Birthday [SEP] $11.99…

RWR

Thought 2:
I think I should click on the 'B07BKXMCNB' option as it seems to be a long lasting, lead free candle with soy wax and a price within my budget.
Action 2:
click[B07BKXMCNB]
Observation 2:
<product details>

Thought 3:
I think I should click on the 'Buy Now' button to purchase the candle.
Action 3:
click[Buy Now]

Reward: 0.0

DPO

Thought 2:
I think I should click on the 'B07BKXMCNB' option as it seems to be a long lasting, lead free, soy wax candle within my price range.
Action 2:
click[B07BKXMCNB]
Observation 2:
<product details>

Thought 3:
I think I should click on the 'Buy Now' button to purchase the long lasting, lead free, soy wax candle.
Action 3:
click[Buy Now]

Reward: 0.0

PPO

Thought 2:
I think I should click on the 'Next >' button to view more results.
Action 2:
click[Next >]
Observation 2:
<next page searching results>

Thought 3:
I think I should click on the 'Next >' button to view more results.
Action 3:
click[Next >]

...(more interactions)

Action 10:
click[Next >]
...(reach max_interaction_rounds)
Reward: 0.0

AgentEvol

Thought 2:
I think I should click on the 'B08JQ5TLQV' option as it seems to be a long-lasting, lead-free candle with soy wax and a price lower than 50.00 dollars.
Action 2:
click[B08JQ5TLQV]
Observation 2:
<product details>

Thought 3:
I think I should click on the 'Buy Now' button to purchase the scented candles.
Action 3:
click[Buy Now]

Reward: 1.0

Figure 9: Case study of WebShop. The RWR and DPO baselines lead to the selection of the first item without considering the price constraint, resulting in task failure. In comparison, the PPO baseline continuously clicks “Next Page” without effectively extracting relevant information from the environment, also failing to find a suitable item. After evolution, the agent demonstrates improved capabilities by accurately parsing product details, conducting effective multi-round interactions, and successfully identifying a long-lasting, lead-free soy candle within the price range.

System Prompt:
Interact with a household to solve a task. Imagine you are an intelligent agent in a household environment and your target is to perform actions to complete the task goal. At the beginning of your interactions, you will be given a detailed description of the current environment and your goal to accomplish. For each of your turns, you will be given a list of actions and you can choose one to perform in this turn. You should choose from two actions: "THOUGHT" or "ACTION". If you choose "THOUGHT", you should first think about the current condition and plan for your future actions, and then output your action in this turn. Your output must strictly follow this format:

Thought: your thoughts.
Action: your next action.

If you choose "ACTION", you should directly output the action in this turn. Your output must strictly follow this format: "Action: your next action". After each turn, the environment will give you immediate feedback based on which you plan your next few steps. If the environment outputs "Nothing happened", that means the previous action is invalid and you should try more options. Reminder: the action must be chosen from the given available actions. Any actions except provided available actions will be regarded as illegal. Think when necessary, try to act directly more in the process.

Instruction:
You are in the middle of a room. Looking quickly around you, you see a armchair 1, a coffeetable 1, a diningtable 2, a diningtable 1, a drawer 6, a drawer 5, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a dresser 1, a garbagecan 1, a sidetable 1, a sofa 2, a sofa 1, and a tvstand 1.

Your task is to: find two tissuebox and put them in coffeetable.

AVAILABLE ACTIONS: go to armchair 1, go to coffeetable 1, go to diningtable 1, go to diningtable 2, go to drawer 1, go to drawer 2, go to drawer 3, go to drawer 4, go to drawer 5, go to drawer 6, go to dresser 1, go to garbagecan 1, go to sidetable 1, go to sofa 1, go to sofa 2, go to tvstand 1, inventory, look.

Table 11: Prompt details for ALFWorld.

System Prompt:
You are given a few useful crafting recipes to craft items in Minecraft. Crafting commands are of the format “craft [target object] using [input ingredients]”. Every round I will give you an observation, you have to respond to an action based on the state and instruction. You can “get” an object (ingredients) from the inventory or the environment, look up the game “inventory” by inventory, or “craft” (target) using any of the crafting commands. You can use ONLY these crafting commands provided, do not use your own crafting commands. However, if the crafting command uses a generic ingredient like “planks”, you can use special types of the same ingredient e.g. dark oak “planks” in the command instead. Your response should use the following format:

Thought: ...
Action: ...

Instruction:
Crafting commands:
craft 1 golden shovel using 2 stick, 1 gold ingot
craft 1 golden chestplate using 8 gold ingot
craft 1 golden sword using 1 stick, 2 gold ingot
craft 1 netherite ingot using 4 netherite scrap, 4 gold ingot
craft 1 light weighted pressure plate using 2 gold ingot
craft 1 golden boots using 4 gold ingot
craft 1 golden axe using 2 stick, 3 gold ingot
craft 9 gold nugget using 1 gold ingot
Goal: craft gold nugget.

Table 12: Prompt details for TextCraft.

System Prompt:
You are an agent for the science world. Every round I will give you an observation, you have to respond with an action based on the observation to finish the given task.
Here are the actions you may take:
{"action": "open/close OBJ", "description": "open/close a container",}
{"action": "de/activate OBJ", "description": "activate/deactivate a device",}
{"action": "connect OBJ to OBJ", "description": "connect electrical components", }
{"action": "disconnect OBJ", "description": "disconnect electrical components",}
{"action": "use OBJ [on OBJ]", "description": "use a device/item",}
{"action": "look around", "description": "describe the current room",}
{"action": "look at OBJ", "description": "describe an object in detail",}
{"action": "look in OBJ", "description": "describe a container's contents",}
{"action": "read OBJ", "description": "read a note or book",}
{"action": "move OBJ to OBJ", "description": "move an object to a container", }
{"action": "pick up OBJ", "description": "move an object to the inventory", }
{"action": "put down OBJ", "description": "drop an inventory item",}
{"action": "pour OBJ into OBJ", "description": "pour a liquid into a container", }
{"action": "dunk OBJ into OBJ", "description": "dunk a container into a liquid", }
{"action": "mix OBJ", "description": "chemically mix a container",}
{"action": "go to LOC", "description": "move to a new location",}
{"action": "eat OBJ", "description": "eat a food",}
{"action": "flush OBJ", "description": "flush a toilet",}
{"action": "focus on OBJ", "description": "signal intent on a task object",}
{"action": "wait", "description": "take no action for 10 iterations",}
{"action": "wait1", "description": "take no action for 1 iteration", }
{"action": "task", "description": "describe current task",}
{"action": "inventory", "description": "list your inventory"}

Your response should use the following format:

Thought: your thoughts.
Action: your next action.

Instruction:
Your task is to find a(n) non-living thing. First, focus on the thing. Then, move it to the orange box in the living room. This room is called the bedroom. In it, you see: the agent, a substance called air, a bed. On the bed is: a mattress. On the mattress is: a white pillow. a book shelf (containing A book (Beowulf) titled Beowulf by Beowulf poet, A book (Pride and Prejudice) titled Pride and Prejudice by Jane Austen, A book (Sherlock Holmes) titled Sherlock Holmes by Arthur Conan Doyle), a closet. The closet door is closed. a finger painting, a table. On the table is: nothing. You also see: A door to the hallway (that is closed)

Table 13: Prompt details for SciWorld.

System Prompt:
You are an exploration master who wants to finish every goal you are given. Every round I will give you an observation, and you have to respond to an action and your thought based on the observation to finish the given task. You are placed in a room and you need to accomplish the given goal with actions. You can use the following actions:
- turn right
- turn left
- move forward
- go to <obj> <id>
- pick up <obj> <id>
- go through <door> <id>: <door> must be an open door.
- toggle and go through <door> <id>: <door> can be a closed door or a locked door. If you want to open a locked door, you need to carry a key that is of the same color as the locked door.
- toggle: there is a closed or locked door right in front of you and you can toggle it.

Your response should use the following format:

Thought: <Your Thought>
Action: <Your Action>

Instruction:
Your goal: go to the red ball
In front of you in this room, you can see several objects: There is a grey box 1 1 steps in front of you and 1 steps to your left. There is a grey ball 1 1 steps in front of you and 2 steps to your right. There is a grey key 1 1 steps in front of you and 3 steps to your right. The room has walls around you. You are facing a wall 3 steps away. You are not carrying anything.
Available actions: ["turn left", "turn right", "move forward", "pickup red ball 1", "pickup red box 1", "go to red ball 1", "go to red box 1", "check available actions"]

Table 14: Prompt details for BabyAI.

System Prompt:
You are an expert maze solver. Your objective is to reach the goal in as few steps as possible. At each step you will be given information about where the goal is, your current position, and the walls that surround you. When you move right you increase your y position by 1, when you move down you increase your x position by 1. Your possible actions are "move up", "move down", "move left", "move right". Formally, your return should be in this format:

Thought: <Your Thought>
Action: <Your Action>

Instruction:
Now let's start a new game. Return your action and your thought in the format above strictly. Now, make the optimal action given the current environment state: The goal is at position 8, 6. Your current position is at position 1, 1. There are walls to your left, above you, below you.

Table 15: Prompt details for MAZE.

System Prompt:
You are an expert wordle player. Welcome to the game of Wordle. Your objective is to guess a hidden 5 letter word. You have 6 attempts to guess it correctly and you should try to guess it in as few attempts as possible. When guessing the word, you should format your word as a space separated sequence of letters, like “s h i r e” for example. After guessing the word, you will receive feedback from the game environment in the form of a sequence of 5 space separated letters like “b y g g b”, where each letter indicates some information about the hidden word. The environment will return one of three letters - “b”, “g”, or “y” – for each letter in the word you guessed. We describe the meaning of each letter below:
“b”: If the environment returns a “b”, it means that the letter at that position in your guessed word is not in the hidden word.
“y”: If the environment returns a “y”, it means that the letter at that position in your guessed word is in the hidden word but is not in the correct position.
“g”: If the environment returns a “g”, it means that the letter at that position in your guessed word is in the hidden word and is in the correct position.
As a note, if you guess an invalid word (e.g. not a 5 letter word or a word not in the vocabulary), the environment will respond with an “invalid word” message. In general though, you should use this information returned by the environment to update your belief about what the hidden word might be and adjust your next guess accordingly.

Instruction:
Now let’s start a new game. Remember, the word you guess should be strictly in the vocabulary. You should return your thought and your word strictly in the formation mentioned above.

Table 16: Prompt details for Wordle.

System Prompt:
Given you a description of a SQLite database system, I will ask you a question, then you should help me operate the SQLite database with SQL to answer the question.
You have to explain the problem and your solution to me and write down your thoughts. After thinking and explaining thoroughly, you should give a SQL statement to solve the question. Your response should be like this:

Thought: Your thought here.
Action: `SELECT * FROM table WHERE condition;`

You MUST put SQL in markdown format without any other comments. Your SQL should be in one line. Every time you can only execute one SQL statement.

Instruction:
debit_card_specializing contains tables such as customers, gasstations, products, transactions_1k, yearmonth. Table customers has columns such as customerid, client segment, currency. customerid is the primary key. Table gasstations has columns such as gas station id, chain id, country, chain segment. gas station id is the primary key. Table products has columns such as product id, description. product id is the primary key. Table transactions_1k has columns such as transaction id, date, time, customer id, card id, gas station id, product id, amount, price. transaction id is the primary key. Table yearmonth has columns such as customer id, date, consumption. is the primary key. The date of yearmonth is the foreign key of client segment of customers.

Among the transactions made in the gas stations in the Czech Republic, how many of them are taken place after 2012/1/1?

Table 17: Prompt details for BIRD.

System Prompt:
You are an autonomous intelligent agent tasked with navigating a web browser. You will be given web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:
The user's objective: This is the task you're trying to complete.
The current web page's accessibility tree: This is a simplified representation of the webpage, providing key information.
The current web page's URL: This is the page you're currently navigating.
The open tabs: These are the tabs you have open.
The previous action: This is the action you just performed. It may be helpful to track your progress.

The actions you can perform fall into several categories:

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

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

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

Completion Action:
stop [answer]: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Homepage:
If you want to visit other websites, check out the homepage at http://homepage.com. It has a list of websites you can visit.
http://homepage.com/password.html lists all the account name and password for the websites. You can use them to log in to the websites.

Table 18: Prompt details for WebArena (Part 1/2).

To be successful, it is very important to follow the following rules:
1. You should only issue an action that is valid given the current observation.
2. You should only issue one action at a time.
3. You should follow the examples to reason step by step and then issue the next action.
4. Generate the action in the correct format. Start with a “In summary, the next action I will perform is” phrase, followed by action inside. For example, “In summary, the next action I will perform is click [1234]”.
5. Issue stop action when you think you have achieved the objective. Don’t generate anything after stop.

Instruction:
Observation:
Tab 0 (current): Projects · Dashboard · GitLab

```
[1] RootWebArea 'Projects · Dashboard · GitLab' focused: True
    [271] link 'Skip to content'
    [398] link 'Dashboard'
    [482] button '' hasPopup: menu expanded: False
    [1947] textbox 'Search GitLab' required: False
    [1907] generic 'Use the shortcut key <kbd>/</kbd> to start a search'
...
```

URL: http://gitlab.com/
OBJECTIVE: Checkout merge requests assigned to me
PREVIOUS ACTION: None

Table 19: Prompt details for WebArena (Part 2/2).

System Prompt:
You are an autonomous intelligent agent. You can use actions to help people solve problems. We detail name, description, input(parameters) and output(returns) of each action as follows:
Name: get_user_current_date()
Description: Get the user's current date.
Returns:
The current date in 'YYYY-MM-DD' format.

Name: get_user_current_location()
Description: Get the user's current city.
Returns:
The user's current city.

Name: get_historical_temp(latitude, longitude, start_date, end_date)
Description: Get historical temperature data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the historical data (YYYY-MM-DD).
- end_date (Type: string): The end date of the historical data (YYYY-MM-DD).

Returns:
Historical temperature data.

Name: get_historical_rain(latitude, longitude, start_date, end_date)
Description: Get historical rainfall data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the historical data (YYYY-MM-DD).
- end_date (Type: string): The end date of the historical data (YYYY-MM-DD).

Returns:
Historical rainfall data.

Name: get_historical_snow(latitude, longitude, start_date, end_date)
Description: Get historical snowfall data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the historical data (YYYY-MM-DD).
- end_date (Type: string): The end date of the historical data (YYYY-MM-DD).

Returns:
Historical snowfall data.

Table 20: Prompt details for Weather (Part 1/4).

Name: get_snow_forecast(latitude, longitude, start_date, end_date)
Description: Get snowfall forecast data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the forecast (YYYY-MM-DD).
- end_date (Type: string): The end date of the forecast (YYYY-MM-DD).

Returns:
Snowfall forecast data.

Name: get_current_snow(latitude, longitude, current_date)
Description: Get current snowfall data for a specified location and date.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- current_date (Type: string): The current date to retrieve snowfall data (YYYY-MM-DD).

Returns:
Current snowfall data.

Name: get_current_temp(latitude, longitude, current_date)
Description: Get current temperature data for a specified location and date.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- current_date (Type: string): The current date to retrieve temperature data (YYYY-MM-DD).

Returns:
Current temperature data.

Name: get_latitude_longitude(name)
Description: Get latitude and longitude information for a specified location name.
Parameters:
- name (Type: string): The name of the location. (e.g., city name)

Returns:
latitude and longitude information for the specified location.

Name: get_elevation(latitude, longitude)
Description: Get elevation data for a specified location.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.

Returns:
Elevation data for the specified location.

Table 21: Prompt details for Weather (Part 2/4).

Name: get_temp_forecast(latitude, longitude, start_date, end_date)
Description: Get temperature forecast data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the forecast (YYYY-MM-DD).
- end_date (Type: string): The end date of the forecast (YYYY-MM-DD).

Returns:
Temperature forecast data.

Name: get_rain_forecast(latitude, longitude, start_date, end_date)
Description: Get rainfall forecast data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the forecast (YYYY-MM-DD).
- end_date (Type: string): The end date of the forecast (YYYY-MM-DD).

Returns:
Rainfall forecast data.

Name: get_current_rain(latitude, longitude, current_date)
Description: Get current rainfall data for a specified location and date.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- current_date (Type: string): The current date to retrieve rainfall data (YYYY-MM-DD).

Returns:
Current rainfall data.

Name: get_distance(latitude1, longitude1, latitude2, longitude2)
Description: Calculate the distance between two sets of latitude and longitude coordinates.
Parameters:
- latitude1 (Type: number): The latitude of the first location.
- longitude1 (Type: number): The longitude of the first location.
- latitude2 (Type: number): The latitude of the second location.
- longitude2 (Type: number): The longitude of the second location.

Returns:
The distance between the two sets of coordinates in kilometers.

Name: get_historical_air_quality_index(latitude, longitude, start_date, end_date)
Description: Get historical air quality index data for a specified location and date range.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- start_date (Type: string): The start date of the historical data (YYYY-MM-DD).
- end_date (Type: string): The end date of the historical data (YYYY-MM-DD).

Returns:
Historical air quality index (PM2.5) data.

Table 22: Prompt details for Weather (Part 3/4).

Name: get_current_air_quality_index(latitude, longitude, current_date)
Description: Get current air quality index data for a specified location and date.
Parameters:
- latitude (Type: number): The latitude of the location.
- longitude (Type: number): The longitude of the location.
- current_date (Type: string): The current date to retrieve air quality index data (YYYY-MM-DD).
Returns:
Current air quality index (PM2.5) data.

Name: get_air_quality_level(air_quality_index)
Description: Determine the air quality level based on the air quality index (AQI).
Parameters:
- air_quality_index (Type: number): The air quality index (AQI) value.
Returns:
The air quality level, which can be 'good', 'fair', 'moderate', 'poor', 'very poor', or 'extremely poor'.

Name: check_valid_actions()
Description: Get supported actions for current tool.
Returns:
- actions (Type: array): Supported actions for current tool.

Name: finish(answer)
Description: Return an answer and finish the task
Parameters:
- answer (Type: ['string', 'number', 'array']): The answer to be returned
If you want to get the latitude and longitude information of a city, you must call "get_latitude_longitude", do not generate it by yourself which maybe wrong.
If you are finished, you will call "finish" action
Please refer to the format of examples below to solve the requested goal. Your response must be in the format of "Action: [your action] with Action Input: [your action input]"

Instruction:
Now new trial starts. You should perform actions to accomplish the goal: Will there be snowfall and rainfall on the same day next week? Tell me Yes or No. Give me one action.

Table 23: Prompt details for Weather (Part 4/4).

System Prompt:
You are an autonomous intelligent agent. You can use actions to help people solve problems. We detail name, description, input(parameters) and output(returns) of each action as follows:
Name: get_user_current_date()
Description: Get the user's current date.
Returns:
The current date in 'YYYY-MM-DD' format.

Name: get_user_current_location()
Description: Get the user's current city.
Returns:
The user's current city.

Name: get_projects()
Description: Get all projects in the TodoList account
Returns:
- Array of objects with properties:
- id (Type: string)
- name (Type: string)
- order (Type: integer)
- color (Type: string)
- is_favorite (Type: boolean)

Name: update_project(project_id, is_favorite)
Description: Update a project
Parameters:
- project_id (Type: string)
- is_favorite (Type: string, Enum: [True, False])
Returns:
Information of the updated project

Name: get_tasks(project_id)
Description: Get all tasks for a given project
Parameters:
- project_id (Type: string)
Returns:
- Array of objects with properties:
- id (Type: string)
- project_id (Type: string)
- order (Type: integer)
- content (Type: string): Name of the task.
- is_completed (Type: boolean)
- priority (Type: integer): Task priority from 1 (normal) to 4 (urgent).
- due_date (Type: string): The due date of the task.

Table 24: Prompt details for TODOList (Part 1/3).

Name: get_task_description(task_id)
Description: Get the description of a specific task in the TodoList account.
Parameters:
- task_id (Type: string)
Returns:
- id (Type: string): Unique identifier of the task.
- content (Type: string): Name of the task.
- description (Type: string): Description of the task. Including the Place, Tips, etc.

Name: get_task_duration(task_id)
Description: Get the duration of a specific task in the TodoList account.
Parameters:
- task_id (Type: string)
Returns:
- id (Type: string)
- content (Type: string): Name of the task.
- duration (Type: string): Duration of the task in the format of ‘amount(unit)’.

Name: complete_task(task_id)
Description: Mark a task as completed
Parameters:
- task_id (Type: string)
Returns:
information of the completed task

Name: update_task(task_id, due_date)
Description: Update a task
Parameters:
- task_id (Type: string)
- due_date (Type: string)
Returns:
Information of the updated task

Name: delete_task(task_id)
Description: Delete a specific task from the TodoList account.
Parameters:
- task_id (Type: string): Unique identifier of the task to delete.
Returns:
Information of the deleted task.

Name: check_valid_actions()
Description: Get supported actions for current tool.
Returns:
Supported actions for current tool.

Table 25: Prompt details for TODOList (Part 2/3).

Name: finish(answer)
Description: Call this action, when find the answer for the current task or complete essential operations.
Parameters:
- answer (Type: [‘string’, ‘number’, ‘array’]): If the task is a question answering task, this is the answer to be returned. If the task is an operation task, the answer in ‘done’

If you are finished, you will call “finish” action
Please refer to the format of examples below to solve the requested goal. Your response must be in the format of “Action: [your action] with Action Input: [your action input]”

Instruction:
Now new trial starts. You should perform actions to accomplish the goal: Could you provide the due date for the task ‘Tidy up the living room’ in the Household Chores project? Please answer in ‘YYYY-MM-DD’ format. Give me one action.

Table 26: Prompt details for TODOList (Part 3/3).

System Prompt:
You are an autonomous intelligent agent. You can use actions to help people solve problems. We detail name, description, input(parameters) and output(returns) of each action as follows:
Name: get_search_movie(movie_name)
Description: Search for a movie by name and return basic details
Parameters:
- movie_name (Type: string): The name of the movie to search for.
Returns:
- id : The ID of the found movie.
- overview : The overview description of the movie.
- title : The title of the movie.

Name: get_movie_details(movie_id)
Description: Get detailed information about a movie by ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- budget : The budget of the movie.
- genres : The genres of the movie.
- revenue : The revenue of the movie.
- vote_average : The average vote score of the movie.
- release_date : The release date of the movie.

Name: get_movie_production_companies(movie_id)
Description: Get the production companies of a movie by its ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- production_companies : The production companies of the movie.

Name: get_movie_production_countries(movie_id) Description: Get the production countries of a movie by its ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- production_countries : The production countries of the movie.

Name: get_movie_cast(movie_id)
Description: Retrieve the list of the top 10 cast members from a movie by its ID.
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- cast : List of the top 10 cast members.

Table 27: Prompt details for Movie (Part 1/3).

Name: get_movie_crew(movie_id)
Description: Retrieve the list of crew members (limited to 10) from a movie by its ID. The list primarily includes Director, Producer, and Writer roles.
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- crew : List of the top 10 of crew members

Name: get_movie_keywords(movie_id)
Description: Get the keywords associated with a movie by ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- keywords : The keywords associated with the movie.

Name: get_search_person(person_name)
Description: Search for a person by name.
Parameters:
- person_name (Type: string): The name of the person to search for.
Returns:
- id : The ID of the found person.
- name : The name of the person.

Name: get_person_details(person_id)
Description: Get detailed information about a person by ID
Parameters:
- person_id (Type: string): The ID of the person.
Returns:
- biography : The biography of the person.
- birthday : The birthday of the person.
- place_of_birth : The place of birth of the person.

Name: get_person_cast(person_id)
Description: Retrieve the top 10 movie cast roles of a person by their ID
Parameters:
- person_id (Type: string): The ID of the person.
Returns:
- cast : A list of movies where the person has acted, limited to top 10

Name: get_person_crew(person_id)
Description: Retrieve the top 10 movie crew roles of a person by their ID
Parameters:
- person_id (Type: string): The ID of the person.
Returns:
- crew : A list of movies where the person has participated as crew, limited to top 10

Table 28: Prompt details for Movie (Part 2/3).

Name: get_person_external_ids(person_id)
Description: Get the external ids for a person by ID
Parameters:
- person_id (Type: string): The ID of the person.
Returns:
- imdb_id : The IMDB id of the person.
- facebook_id : The Facebook id of the person.
- instagram_id : The Instagram id of the person.
- twitter_id : The Twitter id of the person.

Name: get_movie_alternative_titles(movie_id)
Description: Get the alternative titles for a movie by ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- titles : The alternative titles of the movie.
- id : The ID of the movie.
Name: get_movie_translation(movie_id)
Description: Get the description translation for a movie by ID
Parameters:
- movie_id (Type: string): The ID of the movie.
Returns:
- translations : The description translation of the movie.
- id : The ID of the movie.
Name: check_valid_actions()
Description: Get supported actions for current tool.
Returns:
- actions (Type: array): Supported actions for current tool.

Name: finish(answer)
Description: Return an answer and finish the task
Parameters:
- answer (Type: [‘string’, ‘number’, ‘array’]): The answer to be returned

If you are finished, you will call “finish" action
Please refer to the format of examples below to solve the requested goal. Your response must be in the format of “Action: [your action] with Action Input: [your action input]"

Instruction:
Now new trial starts. You should perform actions to accomplish the goal: Do the movies “The Godfather” and “Pulp Fiction” share a common genre? Please answer me with Yes or No. Give me one action.

Table 29: Prompt details for Movie (Part 3/3).

System Prompt: You are an autonomous intelligent agent. You can use actions to help people solve problems. We detail name, description, input(parameters) and output(returns) of each action as follows:
Name: loadPaperNet()
Description: Load PaperNet. In this net, nodes are papers and edges are citation relationships between papers.

Name: loadAuthorNet()
Description: Load AuthorNet. In this net, nodes are authors and edges are collaboration relationships between authors.

Name: neighbourCheck(graph, node)
Description: List the first-order neighbors connect to the node. In paperNet, neigbours are cited papers of the paper. In authorNet, neigbours are collaborators of the author.
Parameters:
- graph (Type: string, Enum: [PaperNet, AuthorNet]): The name of the graph to check
- node (Type: string): The node for which neighbors will be listed
Returns:
- neighbors (Type: array)

Name: paperNodeCheck(node)
Description: Return detailed attribute information of a specified paper in PaperNet
Parameters:
- node (Type: string): Name of the paper.
Returns:
- authors : The authors of the paper
- year : The puslished year of the paper
- venue : The published venue of the paper
- n_citation : The number of citations of the paper
- keywords : The keywords of the paper
- doc_type : The document type of the paper

Name: authorNodeCheck(node)
Description: Return detailed attribute information of a specified author in AuthorNet
Parameters:
- node (Type: string): name of the author.
Returns:
- name : The name of the author
- org : The organization of the author

Name: authorEdgeCheck(node1, node2)
Description: Return detailed attribute information of the edge between two specified nodes in a AuthorNet.
Parameters:
- node1 (Type: string): The first node of the edge
- node2 (Type: string): The second node of the edge
Returns:
- papers : All papers that the two authors have co-authored

Table 30: Prompt details for Academia (Part 1/2).

Name: paperEdgeCheck(node1, node2)
Description: Return detailed attribute information of the edge between two specified nodes in a PaperNet.
Parameters:
- node1 (Type: string): The first node of the edge
- node2 (Type: string): The second node of the edge
Returns:
None

Name: check_valid_actions()
Description: Get supported actions for current tool.
Returns:
- actions (Type: array): Supported actions for current tool.

Name: finish(answer)
Description: Return an answer and finish the task
Parameters:
- answer (Type: [‘string’, ‘number’, ‘array’]): The answer to be returned

If you are finished, you will call “finish” action
Please refer to the format of examples below to solve the requested goal. Your response must be in the format of “Action: [your action] with Action Input: [your action input]”

Instruction:
Now new trial starts. You should perform actions to accomplish the goal: How many mutual collaborators do Florian Kirchbuchner and Fadi Boutros share? Please give me a numerical value as an answer. Give me one action.

Table 31: Prompt details for Academia (Part 2/2).

System Prompt:
You are an autonomous intelligent agent. You can use actions to help people solve problems. We detail name, description, input(parameters) and output(returns) of each action as follows:
Name: open_sheet(name)
Description: Open a sheet by name
Parameters:
- name (Type: string): The name of the sheet to open.
Returns:
- result (Type: object): The opened worksheet object or an error message.

Name: del_sheet(name)
Description: Deletes the specified sheet.
Parameters:
- name (Type: string): The name of the sheet to be deleted.
Returns:
- result (Type: object): Whether the operation was successful.

Name: freeze_data(dimension, num)
Description: Freeze rows and/or columns on the worksheet
Parameters:
- dimension (Type: string): The dimension to freeze, either 'rows' or 'columns'
- num (Type: integer): Number of rows/cols to freeze.
Returns:
- result (Type: object): Whether the operation was successful.

Name: get_A1_annotation(row, col)
Description: Translate the cell position (row,col) into A1 annotation
Parameters:
- row (Type: integer): Row index.
- col (Type: integer): Column index.
Returns:
- result (Type: string): The A1 notation of the cell or an error message.

Name: insert_cols(values_list, col_idx)
Description: Insert columns into sheet at specified column index
Parameters:
- values_list (Type: array[array[string]]): A list of lists, each list containing one column's values, which can be expressions
- col_idx (Type: integer): Start column to update. Defaults to 1.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: insert_rows(values_list, row_idx)
Description: Insert rows into sheet at specified row index
Parameters:
- values_list (Type: array[array[string]]): A list of lists, each list containing one row's values, which can be expressions
- row_idx (Type: integer): Start row to update. Defaults to 1.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Table 32: Prompt details for Sheet (Part 1/4).

Name: delete_batch_data(dimension, index_list)
Description: Delete a batch of data in the sheet
Parameters:
- dimension (Type: string): The dimension to delete, either ‘row’ or ‘col’.
- index_list (Type: array[integer]): List of the indexes of rows/cols for deletion.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: update_cell(position, value)
Description: Update the value of the cell
Parameters:
- position (Type: string): A1 notation of the cell position.
- value: The value to set.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: update_cell_by_formula(start_position, end_position, position_list, result_position, operator)
Description: Update the value of the target cell by applying formulas on some specified cells. Note: Either specify position_list or start_position and end_position.
Parameters:
- start_position (Type: string): The starting position of the range. Default: ‘B1’.
- end_position (Type: string): The ending position of the range. Default: ‘D2’.
- position_list (Type: array[string]): A list of cell positions in A1 notation.
- result_position (Type: string): The position of the cell where the result of the formula will be stored in. Default: ‘G2’.
- operator (Type: string): The operator to be applied on selected cells. Choose one from [‘SUM’, ‘AVERAGE’, ‘COUNT’, ‘MAX’, ‘MIN’, ‘MINUS’, ‘PRODUCT’].
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: update_range(start_position, end_position, values_list)
Description: Update a range of the cells from a list
Parameters:
- start_position (Type: string): A1 notation of the start cell.
- end_position (Type: string): A1 notation of the end cell.
- values_list (Type: array[array[Any]]): List of values to be inserted, which can be expressions
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: sort_sheet_by_col(col_num, order)
Description: Sorts the current sheet using given sort orders
Parameters:
- col_num (Type: integer): The index of the sort column.
- order (Type: string): The sort order. Possible values are ‘asc’ or ‘des’.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Table 33: Prompt details for Sheet (Part 2/4).

Name: merge_cells(start_position, end_position)
Description: Merge cells in sheet
Parameters:
- start_position (Type: string): Starting cell position(top left) in A1 annotation.
- end_position (Type: string): Ending cell position(bottom right) in A1 annotation.
Returns:
- result (Type: object): The updated worksheet data or an error message.

Name: update_note(position, content)
Description: Update a note in a certain cell
Parameters:
- position (Type: string): cell position in A1 annotation.
- content (Type: string): The text note to insert.
Returns:
- result (Type: string): The updated note or an error message.

Name: get_all_values()
Description: Display all cell values in current sheet
Returns:
- result (Type: array[array[Any]]): Return all cell values or an error message.

Name: get_range_values(start_position, end_position)
Description: Returns a list of cell data from a specified range.
Parameters:
- start_position (Type: string): Starting cell position in A1 annotation.
- end_position (Type: string): Ending cell position in A1 annotation.
Returns:
- result (Type: array[array[Any]]): List of cell data from the specified range or an error message.

Name: get_cell_value(position)
Description: Get the value of a specific cell
Parameters:
- position (Type: string): Cell position in A1 annotation.
Returns:
- result : Cell value or an error message.

Name: get_value_by_formula(start_position, end_position, position_list, operator)
Description: Calculate a value applying formulas on specified cells. Note: Either specify position_list or start_position and end_position.
Parameters:
- start_position (Type: string): The starting position of the range. Default: 'B1'.
- end_position (Type: string): The ending position of the range. Default: 'D2'.
- position_list (Type: array[string]): A list of cell positions in A1 notation.
- operator (Type: string): The operator to be applied on selected cells. Choose one from ['SUM', 'AVERAGE', 'COUNT', 'MAX', 'MIN', 'MINUS', 'PRODUCT'].
Returns:
- result (Type: string): Calculated result or an error message.

Table 34: Prompt details for Sheet (Part 3/4).

Name: filter_cells(query, in_row, in_column)
Description: Find all cells matching the query, return all cells' position.
Parameters:
- query (Type: ['string', 're.RegexObject']): A string to match or compiled regular expression.
- in_row (Type: ['integer', 'None']): Row number to scope the search. Default is all rows
- in_column (Type: ['integer', 'None']): Column number to scope the search. Default is all columns
Returns:
- result (Type: array[string]): List of cell addresses that match the query or an error message.

Name: get_note(position)
Description: Get the note at the certain cell, or return empty string if the cell does not have a note.
Parameters:
- position (Type: string): Cell position in A1 annotation.
Returns:
- result (Type: string): Note content or an error message.
Name: finish()
Description: Return an answer and finish the task
Returns:
- result (Type: array[array[Any]]): Return all cell values or an error message.

Instruction:
Now new trial starts. You should perform actions to accomplish the goal: Product Update: The table in "Sheet1" contains the product inventory information, and [['Product', 'Today Sold'], ['beef', '5'], ['pork', '2'], ['chicken', '8'], ['lamb', '12'], ['duck', '3'], ['fish', '23'], ['shrimp', '21'], ['salmon', '12'], ['apple', '100'], ['banana', '287'], ['orange', '234'], ['carrot', '12']] is today's sales data. Please update the product information in "Sheet1" in time and then sort by "Quantity" in descending order. Give me one action.

Table 35: Prompt details for Sheet (Part 4/4).