

AGENTGYM: Evolving Large Language Model-based Agents across Diverse Environments

Zhiheng Xi^{*†}, Yiwen Ding^{*}, Wenxiang Chen^{*},
Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao,

Xin Guo, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou,
Tao Gui[†], Qi Zhang[†], Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, Yu-Gang Jiang

Fudan NLP Lab & Fudan Vision and Learning Lab

Abstract

Building generalist agents that can handle diverse tasks and evolve themselves across different environments is a long-term goal in the AI community. Large language models (LLMs) are considered a promising foundation to build such agents due to their generalized capabilities. Current approaches either have LLM-based agents imitate expert-provided trajectories step-by-step, requiring human supervision, which is hard to scale and limits environmental exploration; or they let agents explore and learn in isolated environments, resulting in specialist agents with limited generalization. In this paper, we take the first step towards building generally-capable LLM-based agents with self-evolution ability. We identify a trinity of ingredients: 1) diverse environments for agent exploration and learning, 2) a trajectory set to equip agents with basic capabilities and prior knowledge, and 3) an effective and scalable evolution method. We propose AGENTGYM, a new framework featuring a variety of environments and tasks for broad, real-time, uni-format, and concurrent agent exploration. AGENTGYM also includes a database with expanded instructions, a benchmark suite, and high-quality trajectories across environments. Next, we propose a novel method, AGENTEVOL, to investigate the potential of agent self-evolution beyond previously seen data across tasks and environments. Experimental results show that the evolved agents can achieve results comparable to SOTA models. We release the AGENTGYM suite, including the platform, dataset, benchmark, checkpoints, and algorithm implementations.

Project site: <https://agentgym.github.io>

AGENTGYM suite: <https://github.com/WooooDyy/AgentGym>

1 Introduction

Developing agents capable of performing a wide spectrum of tasks across diverse environments at human-level has been a long-standing goal for AI community, and significant endeavors have been undertaken [1; 2; 3; 4; 5]. Similar to human learning, an agent starts by acquiring basic knowledge and skills through imitation [4; 6]. As it progresses, the agent is expected to continuously learn and adapt to previously unseen tasks by interacting with different environments [7; 8; 6; 9]. Moreover, it may harness a rich source of insights and wisdom from its own experiences as well as those of others, developing a certain level of generalization ability [10; 11]. Figure 1 illustrates this evolution process.

^{*} Equal Contribution.

[†] Correspondence to: zhxi22@m.fudan.edu.cn, {tgui, qz}@fudan.edu.cn

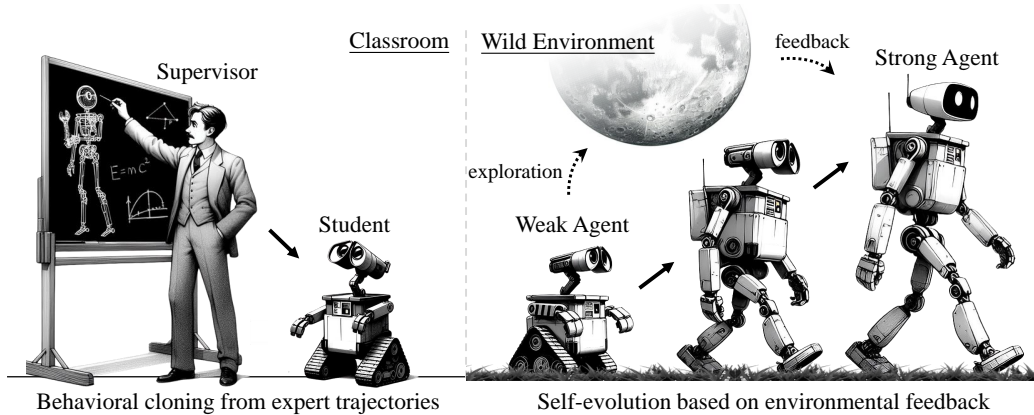


Figure 1: An illustration of self-evolution for generally-capable LLM-based agents in our paper. The agent first performs behavioral cloning according to human supervision, and then performs exploring and learning across environments and tasks to evolve themselves.

Large language models (LLMs) are considered a promising foundation for constructing such generalist agents due to their generalized abilities [12; 13; 14], and many efforts have been made in this realm [5; 15]. One line of work primarily relies on human supervision, where LLM-based agents mimic expert-provided trajectories from various environments step-by-step, i.e. behavioral cloning (BC) [16; 17; 18]. This method, while effective, requires skilled annotators and significant financial resources, making it hard to scale [19]. Moreover, such a paradigm may encounter bottlenecks in performance, adaptability, and generalization due to insufficient exploration of the environments [20]. Another line of research allows LLM-based agents to improve themselves based on environmental feedback (i.e., self-improvement), reducing reliance on human supervision while enriching exploration of the environment [21; 10; 22]. Yet, they typically train agents in isolated environments on specific tasks such as web navigation, and these specialist agents can not generalize beyond narrow tasks.

In this paper, we take the initial step to investigate the potential of self-evolution in generally capable LLM-based agents across various tasks and environments, progressing from imitation to interactive learning, akin to humans (Figure 1). We identify three key pillars necessary for this research goal. First, diverse environments and tasks that allow the agents to evolve dynamically and comprehensively, rather than being confined to an isolated world, which may limit generalization [7; 23; 8; 6]. Second, a trajectory set of an appropriate size to train a base agent with preliminary instruction-following abilities and knowledge. This facilitates further exploration as in diverse, complex environments, it would be extremely inefficient for an agent to learn everything from scratch through trial and error [6; 22]. Third, an effective and flexible evolution method can adapt to environments of varying difficulty and elicit the generalizing ability of LLM-based agents. This involves how the agent interacts with the environment and how it utilizes the feedback [19; 20].

Considering the three pillars, we present AGENTGYM (see Figure 2), a new framework designed to help the community develop generally-capable LLM-based agents and explore self-evolution. Our main contributions are:

1. An interactive platform that includes diverse environments, tasks, and goals for LLM-based agents. AGENTGYM offers convenient APIs through HTTP services, standardizing task specifications, environment settings, and the observation/action spaces for agents. Within this platform, we have implemented a unified interface for multi-round interactions and real-time feedback across different environments to support online evaluation, trajectory sampling, and interactive training. Specifically, it includes 14 types of agent environments, 89 types of tasks, spanning web tasks [24; 25], embodied tasks [26; 27], and more [28; 29; 30; 31; 32], with high flexibility to expand to additional ones.

2. Expanded instructions, benchmark suite, and high-quality interactive trajectories across environments. We collect instructions from various environments and tasks, expanding them through crowdsourcing and AI-based methods such as self-instruct [33] and instruction evolution [34]. Subsequently, we select a diverse and challenging subset to form the test set to construct a benchmark suite named AGENTEval. Next, using crowdsourcing procedures and state-of-the-art (SOTA) models, we annotate and filter a trajectory set in a uniform format named AGENTTRAJ.

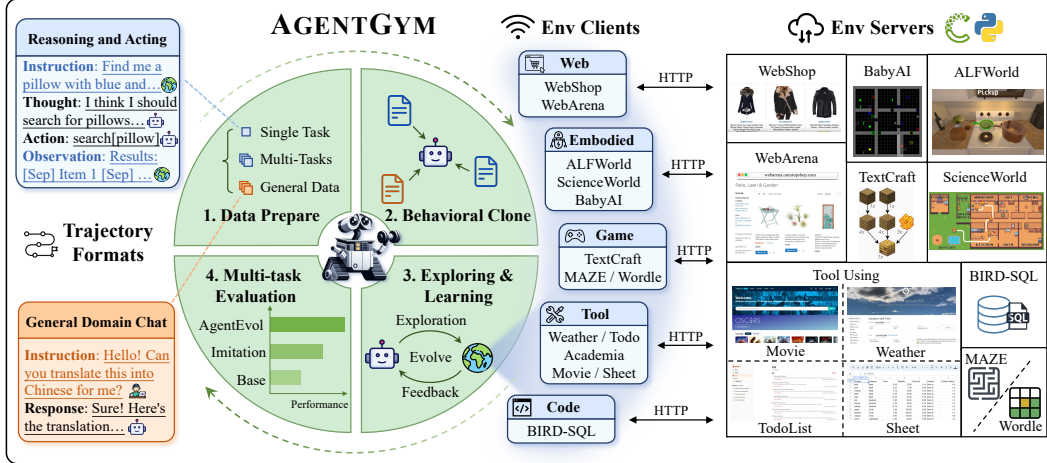


Figure 2: Overview of the AGENTGYM framework. It covers fourteen environments spanning diverse categories. Each environment is deployed as an HTTP service, and clients provide encapsulated, unified interfaces for agents, facilitating interaction with environments. We gather expert-annotated trajectories from diverse environments, called AGENTTRAJ. We then let the agent perform behavioral cloning on this set to obtain a base generally-capable agent. With our AGENTEVOL method, we explore the agent’s evolution across various environments and tasks. Finally, we evaluate the agent comprehensively using the proposed benchmark suite AGENTEVAL.

This set is used to train a base generally-capable agent, bootstrapping further agent exploration and evolution. For a fair comparison, we also collect a larger trajectory set AGENTTRAJ-L with the same pipeline to train an agent that serves as the maximum performance achievable through BC. Note that AGENTTRAJ-L is an extension of AGENTTRAJ.

3. Initial investigation of self-evolution for generally-capable LLM-based agents based on environmental feedback. Starting from the base generally-capable agent, we propose AGENTEVOL, a novel method to explore agent evolution across multiple environments. Our focus is on investigating whether agents can evolve themselves when facing previously unseen tasks and instructions, which requires them to perform exploration and learn from new experiences. Experimental results show that the evolution of agents is very pronounced, even achieving comparable or better performance than SOTA models. Moreover, we perform sufficient ablation and analysis to show how our method works.

In summary, we present AGENTGYM, a new framework that includes an interactive platform of multiple agent environments, a benchmark suite AGENTEVAL, and two trajectory sets AGENTTRAJ and AGENTTRAJ-L. We also propose a new algorithm AGENTEVOL to explore self-evolution in generally-capable LLM-based agents. We will release the whole suite, algorithm implementations, and agent checkpoints. We hope that AGENTGYM will help the community to develop new algorithms and advancements towards better generalist LLM-based agents.

2 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 $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, and reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

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 [35] to modeling agent outputs, 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 $h_{t+1} \sim \pi_\theta(\cdot|e, u, h_1, a_1, o_1, \dots, h_t, a_t, o_t)$ first and the subsequent action $a_{t+1} \sim \pi_\theta(\cdot|e, u, h_1, a_1, o_1, \dots, h_t, a_t, o_t, h_{t+1})$. The trajectory is represented as:

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

$$\pi_\theta(\tau|e, u) = \prod_{t=1}^T \pi_\theta(h_t, a_t|e, u, c_{t-1}) = \prod_{t=1}^T \pi_\theta(a_t|e, u, c_{t-1}, h_t) \cdot \pi_\theta(h_t|e, u, c_{t-1}), \quad (2)$$

where T is the interaction rounds, and $c_{t-1} = (h_1, a_1, o_1, \dots, h_{t-1}, a_{t-1}, o_{t-1})$ represents the interactive history up to $t - 1$. The final reward $r(e, u, \tau) \in [0, 1]$ is then computed.

3 AGENTGYM: Platform, Benchmark Suite and Trajectory Set

AGENTGYM is a framework designed to help the community easily evaluate and develop generally-capable LLM-based agents. It features diverse interactive environments and tasks with a unified format, i.e., ReAct format [35]. It supports real-time feedback and concurrency, and is easily scalable. We include 14 environments and 89 tasks across web navigating, text games, house-holding tasks, digital games, embodied tasks, tool-using and programming. They are challenging for current LLM-based agents. For web navigating task, we introduce WebArena (WA) [24] and WebShop (WS) [25]. We include MAZE (MZ) and Wordle (WD) [28] in text games. We choose ALFWorld (ALF) [29] for house-holding tasks. We include SciWorld (Sci) [26] and BabyAI (Baby) [27] in embodied tasks. We choose TextCraft (TC) [30] for digital games. We get Tool-Weather (WT), Tool-Movie (MV), Tool-Academia (AM), Tool-Sheet (ST) and Tool-TODOList (TL) [31] for tool using tasks. We establish BIRD (BD) [32] for programming tasks. See Appendix C for environment details. The comparison between AGENTGYM and other frameworks is demonstrated in Table 1.

Platform architecture and components.

Recognizing the diverse dependencies inherent to different agent environments, AGENTGYM deploys separate services for each environment in a user-friendly manner to prevent conflicts. The clients can communicate with environments using HTTP protocol. At the core of this architecture is the controller, which acts as a conduit for interactions between agents and environmental services, providing an encapsulated, unified interface of environmental functions or operations for agents to invoke. Additionally, we implement user-friendly components such as the evaluator, trainer, and data collection pipeline to support community development. Figure 4 in Appendix D illustrate the architecture design of the platform.

Instruction collecting and benchmark construction. We have collected 20509 instructions and queries across environments and tasks. For tasks that already have an abundance of instructions, such as WebShop and ALFWorld, we primarily rely on their original source. Meanwhile, for tasks with fewer instructions, like tool-using tasks, we expand upon them using self-instruct and instruction evolution methods, specifically by prompting GPT-4 to generate new instructions [33; 34]. The details are in Appendix C. We then extract a diverse and challenging subset \mathcal{Q}_{eval} of 1160 instructions from each environment to construct a benchmark suite AGENTVAL, which can comprehensively evaluate LLM-based agents. The remained instruction set is denoted as $\mathcal{Q} = \bigcup_{e \in \mathcal{E}} \mathcal{Q}_e$, where \mathcal{Q}_e means the remained instructions of environment e .

Table 1: Comparison of AGENTGYM with other agent frameworks covers several aspects: the number of environments, presence of an interactive platform and its usage, availability of trajectory sets, support for evolution, and the evolution mode.

Frameworks	Env.	Inter. Plat.	Traj.	Evol.
AgentBench [36]	8	Eval	No	No
AgentBoard [31]	12	Eval	No	No
AgentOhana [18]	10	No	Yes	No
Pangu-Agent [37]	6	No	Yes	Single-Env
AGENTGYM (Ours)	14	Eval & Train	Yes	Multi-Env

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

Env.	Task Num	Instr.	Eval.	Traj	Traj-L	Rounds
WA	3	812	20	0	0	—
WS	1	6910	200	1000	3930	5.1
MZ	1	240	25	100	215	4.3
WD	1	980	25	500	955	4.3
ALF	6	3827	200	500	2420	13.3
Sci	30	2320	200	1000	2120	19.9
Baby	40	900	90	400	810	5.7
TC	1	544	100	300	374	8.0
WT	1	343	20	160	311	5.5
MV	1	238	20	100	215	4.0
AM	1	20	20	0	0	—
ST	1	20	20	0	0	—
TL	1	155	20	70	135	5.6
BD	1	3200	200	2000	3000	1.0
Total	89	20509	1160	6130	14485	—

Trajectory collecting and filtering. In AGENTGYM, the server provides instructions including task description, environment setup, and problem to the agent. Next, as described in Section 2, the agent interacts with the environment in ReAct-Style until the task is completed. We collect trajectories with SOTA models (e.g., GPT-4-Turbo) and crowdsourced annotations. Details are in Appendix C. We rigorously filter the trajectories to ensure data quality based on the reward or correctness, and get a set of 6130 trajectories. This set, named AGENTTRAJ, is used to train a base generally-capable agent in Section 4.1. For a fair comparison, we perform annotation and filtering on all instructions using the same pipeline and get AGENTTRAJ-L to show the performance upper bound of BC.

Detailed statistics of AGENTGYM framework are shown in Table 2.

4 AGENTEVOL for Evolution of Generally-capable LLM-based Agents

In this section, we first train a base generally-capable agent through behavioral cloning to equip it with basic interactive ability in agent tasks. Building on this agent, we take the initial steps to explore the comprehensive evolution of LLM-based agents across multiple environments and tasks. We summarize the algorithm in Algorithm 1.

4.1 Behavioral Cloning with Collected Trajectories

Behavioral cloning fine-tunes LLM-based agents by having them mimic the collected expert trajectories step-by-step. In practice, we expect the agent to accomplish the appropriate inner thought h and action a . We use AGENTTRAJ (denoted as \mathcal{D}_s) to train a base generally-capable agent with basic instruction-following ability and prior knowledge. We maximize the following objective:

$$\begin{aligned}\mathcal{J}_{BC}(\theta) &= \mathbb{E}_{(e,u,\tau) \sim \mathcal{D}_s} \left[\log \pi_{\theta}(\tau|e, u) \right] \\ &= \mathbb{E}_{(e,u,\tau) \sim \mathcal{D}_s} \sum_{t=1}^T \left[\log \pi_{\theta}(a_t|e, u, c_{t-1}, h_t) + \log \pi_{\theta}(h_t|e, u, c_{t-1}) \right].\end{aligned}\quad (3)$$

Note that we include a general domain dataset $\mathcal{D}_{general}$ as in Zeng et al. [38] to maintain the agent’s ability in language understanding and generation. And the resulting agent $\pi_{\theta_{base}}$ serves as a starting point for later evolution across diverse environments and tasks.

4.2 Evolution through Exploration and Learning

This work tries to explore the potential of self-evolution in generally-capable LLM-based agents across multiple environments and tasks. **More importantly, the agents will face previously unseen tasks and instructions during evolution.** Hence, agents are required to explore environments, receive feedback, and optimize themselves based on the feedback. To achieve our goal, reinforcement learning (RL) [39] is worth considering, and the corresponding objective is:

$$\mathcal{J}_{RL}(\theta) = \mathbb{E}_{e \in \mathcal{E}, u \sim \mathcal{Q}_e, \tau \sim \pi_{\theta}(\tau|e, u)} [r(e, u, \tau)]. \quad (4)$$

However, in our setting, standard RL faces significant challenges due to large sampling space and long-term nature of agent tasks, leading to high computational complexity and training instability, which hampers scalability [40; 41; 42]. Hence, we draw inspiration from the well-established connection between RL and probabilistic inference [43; 44; 45; 46], and propose a method called AGENTEVOL for agent evolution, which involves agents alternating between exploration and learning.

Learning from estimated optimal policy. In this work, we view RL as an inference problem within a specific probabilistic model [43; 46; 47]. Differing from traditional RL formulations that focus on identifying a trajectory that maximizes the expected reward, inference-based approaches start with an optimal distribution over trajectories. We initially define $P(O = 1)$ to represent the event of “obtained optimal policy by maximum expected rewards” or “achieving success in the RL task”, which can be calculated by integrating the optimal policy probability at each sampling point. Given the policy agent π_{θ} , the optimal policy can be obtained by maximizing:

$$\log P_{\pi_{\theta}}(O = 1) = \log \int \pi_{\theta}(\tau) p(O = 1|\tau) d\tau. \quad (5)$$

Algorithm 1: AGENTEVOL

Input: Initialized policy LLM-based agent π_θ , environment set \mathcal{E} , trajectory subset \mathcal{D}_s , full instruction set \mathcal{Q} , reward function r .

Procedure Behavioral cloning:

 Maximize objective $\mathcal{J}_{BC}(\theta) = \mathbb{E}_{(e,u,\tau) \sim \mathcal{D}_s} [\log \pi_\theta(\tau|e, u)]$ 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 $\mathcal{D}_m^e = \{(e, u^j, \tau^j) \mid u^j \sim \mathcal{Q}_e, \tau^j \sim \pi_{\theta^m}(\tau|e, u^j)\}_{j=1}^{|\mathcal{D}_m^e|}$;

 Compute reward for \mathcal{D}_m with r ;

$\mathcal{D}_m \leftarrow \mathcal{D}_m \cup \mathcal{D}_s$;

// Perform Learning Step

 Maximize objective $\mathcal{J}_{Evol}(\theta) = \mathbb{E}_{(e,u,\tau) \sim \mathcal{D}_m} [r(e, u, \tau) \log \pi_\theta(\tau|e, u)]$ to get $\pi_{\theta^{m+1}}$;

end

However, the above optimization process is difficult to proceed directly due to the fact that LLM-based agents require token-wise feedback to perform gradient updates. In this paper, we alternatively construct the variational lower bound of Eq.5 by introducing an estimation function q on the optimal policy. With Jensen’s inequality, we soon have:

$$\begin{aligned} \log \int \pi_\theta(\tau) p(O = 1|\tau) d\tau &= \log \mathbb{E}_{q(\tau)} \left[\frac{\pi_\theta(\tau)}{q(\tau)} p(O = 1|\tau) \right] \geq \mathbb{E}_q \left[\log \frac{\pi_\theta(\tau)}{q(\tau)} p(O = 1|\tau) \right] \\ &= \mathbb{E}_q [\log p(O = 1|\tau)] - \text{KL}[q(\tau) \parallel \pi_\theta(\tau)] = \mathcal{J}(q, \pi_\theta), \end{aligned} \quad (6)$$

where π_θ is the trajectory distribution induced by the agent, and $q(\tau)$ is a variational distribution.

Due to the monotonicity of the logarithmic function, by maximizing the lower bound $\mathcal{J}(q, \pi_\theta)$, we can obtain a policy with an expected return higher than before. Generally, our framework can be divided into two steps of loop iteration. The former step of $\mathcal{J}(q, \pi_\theta)$ can be explained as estimating the optimal policy distribution on the sampled trajectories by maximizing the expected reward over the state space. The later step relates to updating the current agent’s parameters θ towards the optimal policy q , thus completing the optimization of one single iteration. In analogy to SGD [48], the estimation process introduces noise to the policy optimization due to the presence of unseen decision trajectories. This error gradually decreases as the algorithm proceeds and converges to zero when the current agent becomes optimal [43].

In the AGENTEVOL algorithm, we refer to the two steps as **Exploration step** and **Learning step**. Specifically, with current agent parameters θ^m and the variational distribution q^m [47], at **exploration step**, the estimation of optimal policy q is updated by maximizing the expected reward: $q^{m+1} = \arg \max_q \mathcal{J}(q, \pi_{\theta^m})$. As $\max_q \mathcal{J}(q, \pi_{\theta^m}) = \min_q [\text{KL}(q(\tau) \parallel p(O = 1|\tau) \pi_{\theta^m}(\tau))]$, we have $q^{m+1} \propto p(O = 1|\tau) \pi_{\theta^m}(\tau)$. This step is equivalent to evaluating the likelihood that the samples generated from the current agent’s policy achieve best rewards, and observe the returns of q by empirically estimating on a pre-constructed training set. And at **learning step**, we optimize $\mathcal{J}(q^{m+1}, \pi_\theta)$ by updating θ . This process is similar to learning a new distribution sampled from the optimal policy on the original training data. Since the first term of $\mathcal{J}(q^{m+1}, \pi_\theta)$ relates only to q as well as τ , the training objective is equivalent to measuring the KL divergence between the estimated policy $q^{m+1}(\cdot)$ and the current policy $\pi_\theta(\cdot)$ over all training samples. We finally derive:

$$\begin{aligned} \theta^{m+1} &:= \arg \min_{\theta} \text{KL}[q^{m+1}(\tau) \parallel \pi_\theta(\tau)] \\ &= \arg \min_{\theta} \sum_{\tau} -q^{m+1}(\tau) \log \pi_\theta(\tau). \end{aligned} \quad (7)$$

This involves optimizing a weighted negative log-likelihood function based on q^{m+1} , which adjusts the agent policy to increase the likelihood of generating higher-reward trajectories, thereby improving the agent’s performance.

Practical evolution for LLM-based agent. In our LLM-based agent scenario, the trajectory is conditioned on the environment e and instruction u . Considering our non-negative reward function

$r(e, u, \tau)$, we can get $P(O|e, u, \tau) \propto r(e, u, \tau)$ [47]. Consequently, $q^{m+1}(\tau|e, u) \propto r(e, u, \tau) \cdot \pi_{\theta^m}(\tau|e, u)$. Thus, the policy update in the learning is:

$$\begin{aligned}\theta^{m+1} &:= \arg \min_{\theta} \sum_{\tau} -(r(e, u, \tau) \cdot \pi_{\theta}(\tau|e, u)) \log \pi_{\theta}(\tau|e, u) \\ &= \arg \max_{\theta} \mathbb{E}_{e \in \mathcal{E}, u \sim \mathcal{Q}_e, \tau \sim \pi_{\theta^m}(\tau|e, u)} [r(e, u, \tau) \log \pi_{\theta}(\tau|e, u)].\end{aligned}\tag{8}$$

This can be viewed as a supervised learning objective weighted by the reward. This approach uses the fixed policy agent from the previous iteration to sample data, thereby separating data collection and policy optimization. In contrast, standard RL performs on-policy data sampling and optimization.

Now we describe the two steps of evolution part in AGENTEVOL in practice:

Exploration Step. In the m -th exploring iteration, for each environment e , we have an instruction set \mathcal{Q}_e which is larger than that used in the BC phase, allowing us to investigate agents evolving to unseen tasks and instructions. The current policy agent interacts with this environment, generating a collection of interaction trajectories $\mathcal{D}_m^e = \{(e, u^j, \tau^j) \mid u^j \sim \mathcal{Q}_e, \tau^j \sim \pi_{\theta^m}(\tau|e, u^j)\}_{j=1}^{|\mathcal{D}_m^e|}$. Subsequently, based on the reward function of the environment, we calculate the reward $r(e, u, \tau)$ for each trajectory. The generated dataset from each environment is then merged, resulting in $\mathcal{D}_m = \bigcup_{e \in \mathcal{E}} \mathcal{D}_m^e$. Note that we also include the original trajectory set in Section 4.1 for the learning step, i.e., $\mathcal{D}_m = \mathcal{D}_m \cup \mathcal{D}_s$.

Learning Step. In the m -th learning iteration, we utilize the dataset \mathcal{D}_m obtained from the exploration step to fine-tune the agent with the objective $\mathcal{J}_{Evol}(\theta) = \mathbb{E}_{(e, u, \tau) \sim \mathcal{D}_m} [r(e, u, \tau) \log \pi_{\theta}(\tau|e, u)]$ to get $\pi_{\theta^{m+1}}$. We also include the general domain dataset as in the BC phase. 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 is improved, similar to previous work done on LLM reasoning [49; 47; 20].

By alternating between the two steps, empirical results show that our method facilitates the evolution of an LLM-based agent across both seen and unseen tasks and instructions.

5 Experiments and Discussion

5.1 Experimental Setup

Environments and Tasks. We explore the self-evolution of generally-capable LLM-based agents with the AGENTGYM framework. Main experiments cover 11 environments: WebShop [25], ALF-World [29], SciWorld [26], BabyAI [27], TextCraft [30], BIRD [32], MAZE, Wordle [28], Tool-TODOList, Tool-Weather, and Tool-Movie [31]. Note that instructions used in BC are fewer than those in evolution, to study the agent’s ability to generalize when performing exploration.

Baselines. We include closed-source models like GPT-3.5-Turbo [50], GPT-4-Turbo [12], Claude 3 [13], and DeepSeek-Chat [51]. We also include open-source models like Llama-2-Chat [52], and agents trained on expert trajectories, i.e., AgentLM [38]. For a fair comparison, we include a baseline that performs BC on AGENTTRAJ-L, serving as the maximum performance achievable through BC.

Implementation Details. All experiments are conducted with eight A100-80GB GPUs. Our main backbone model is Llama-2-Chat-7B. Different environments services are deployed on different ports of the same server. We set the iteration number M to 4. To conserve computational resources, each instruction is sampled once during the evolution process. 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 [47] 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 E for more details.

5.2 Main Results

Experiment results in Table 3 demonstrate that: (1) While closed-source models perform well, even SOTA closed-source models like GPT-4-Turbo fail to achieve satisfactory performance on all tasks, highlighting the need for developing more capable agents. (2) Open-source models, represented by

Table 3: Evaluating results on diverse tasks. BC_{base} means the agent trained with AGENTTRAJ, providing a base agent with basic ability and prior knowledge. BC_{large} means the agent that performs BC on AGENTTRAJ-L, representing the performance upper limit of BC in this paper. It rivals, or even surpasses SOTA models and agents. Our evolution method, AGENTEVOL, outperforms BC_{large} on most tasks and environments through exploration and learning. The best performance of each part is highlighted in **bold**.

Method	WS	ALF	TC	Sci	Baby	MZ	WD	WT	MV	TL	BD
Close-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-sourced 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											
BC_{base}	66.50	77.50	44.00	26.42	69.31	12.00	12.00	25.00	5.00	45.00	8.00
BC_{large}	73.50	83.00	60.00	74.47	74.19	12.00	36.00	45.00	5.00	65.00	8.50
AGENTEVOL	76.50	88.00	64.00	38.00	82.70	12.00	12.00	25.00	60.00	70.00	9.00

Llama2-Chat, perform poorly on all tasks, highlighting the importance of the initialization step of BC. (3) Models trained on agent trajectories, like AgentLM [38], 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 [30] or SciWorld [26], which can be attributed to the lack of training data. (4) The agent trained on AGENTTRAJ-L, i.e., BC_{large} , achieves excellent performance, matching or even surpassing SOTA models, showing that it is a strong baseline. (5) AGENTEVOL, despite having limited trajectories for imitation, surpasses BC_{large} and SOTA models on many tasks like WebShop [25], ALFWorld [29] and BabyAI [27], validating the superiority and promise of agent evolution.

Moreover, we report the number of interactive rounds required by different models to solve the task, in order to demonstrate the efficiency of our method (Appendix F.1).

5.3 Discussion & Analysis

Ablation on data merging strategies and iteration number M . In our experiments, we merge the trajectories sampled during each iteration with the initial trajectories to train the agent, rather than merging it with the trajectories generated in the previous iteration. Here, we conduct an ablation study to show the impact of this merging strategy and the iteration number M . Experimental results in Figure 3 show that merging with the initial data provides more stable improvements, while merging with the trajectories from the previous iteration leads to performance fluctuations, possibly due to overfitting [53; 47]. Additionally, as M increases, performance tends to improve but gradually converges in later iterations. Therefore, we choose $M = 4$ to balance performance and efficiency.

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 Table 4 show that performance increases with higher K , but the improvement is not significant. So we select $K = 1$ for computational efficiency.

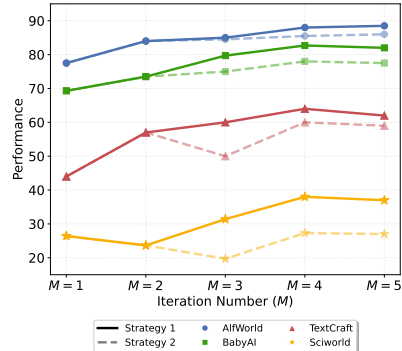


Figure 3: Ablation on data merging strategies and iteration number M . Strategy 1 means merging trajectories generated by the current agent with the initial trajectory set; Strategy 2 means merging current trajectories with the trajectories generated in the previous iteration.

Table 4: Ablation study on sample number K and the exploration scope with four tasks.

Method	WS	ALF	Baby	TC
BC _{base}	66.5	77.5	69.3	44.0
AGENTEVOL				
-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 5: Effectiveness of our method on different models.

Model	Method	WS	ALF	Baby	TC
DeepSeek-Coder-1.3B	BC _{base}	54.0	33.0	68.9	31.0
	BC _{large}	65.0	62.5	73.8	37.0
	AGENTEVOL	67.5	54.5	77.3	38.0
Llama2-Chat-13B	BC _{base}	65.5	81.5	76.6	59.0
	BC _{large}	74.0	85.0	81.1	61.0
	AGENTEVOL	78.5	89.5	86.8	71.0

Ablation on exploration scope. In our experiment, we first train a base agent using \mathcal{D}_s 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 BC phase. Table 4 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.

Effectiveness on different models. To demonstrate the generalizability of our method across different backbone models, we conduct experiments on Llama-2-13B [52] and DeepSeek-Coder-1.3B [54]. The entire evolution process is still based on AGENTGYM. The experimental results in Table 5 show that our AGENTEVOL maintains its evolutionary capabilities across different backbone models, achieving performance that is comparable to or surpasses BC_{large}.

Evolution with both successful and failed trajectories.

In learning step, we only utilize the sampled trajectories with high rewards (success) and do not use failed trajectories. Inspired by previous work [55; 56; 22; 19; 57], we explore whether failed trajectories can be included for better evolution. Specifically, we construct pairs of successful and failed trajectories and optimize the agent using the DPO method [58], which fits models to the pair-wise dataset [57; 55; 59]. Results in Table 6 show that using both types of trajectories can still bring about evolutionary effects, but the performance is not as good as our method, indicating that preference optimization in multi-task setting is more challenging compared to single-task [55; 22]. In the future, we hope to explore more algorithms to make full use of all trajectories for comprehensive evolution.

Table 6: Experiments on evolution with both successful and failed trajectories.

Method	WS	ALF	Baby	TC
BC _{base}	66.5	77.5	69.3	44.0
AGENTEVOL	77.0	88.0	82.9	65.0
DPO with failed traj	75.0	86.5	78.3	58.0

6 Related Work

With the development of LLMs [12; 13; 14], developing agents based on them has become an important research direction [5; 15]. These agents are typically endowed with reasoning and acting capabilities and can perform many types of tasks [35; 20; 60]. To evaluate these agents, researchers have proposed benchmarks that include various tasks [25; 36; 31; 61]. Our benchmark suite AGENTEVOL offers a more diverse set of environments and tasks, providing a more comprehensive evaluation.

Closed-source LLMs, equipped with prompting methods like ReAct [35] and PlanAct [62], can achieve great performance in agent tasks, while agents based on open-source methods perform poor on these tasks [36; 37]. To address this challenge, a series of work collects expert trajectories from diverse environments and tasks and trains LLM-based agents through behavioral cloning [38; 60; 17; 18]. However, obtaining these expert trajectories is often costly and they lack sufficient exploration of the environment [19; 20].

Another line of work trains LLM-based agents based on environmental feedback, referred to as interactive learning methods [21; 37; 22; 63]. Specifically, they involve training LLMs or agents through exploration and learning. As a representative method, RL has succeeded in LLM alignment [64; 65; 50; 41; 66], and has been introduced to reasoning and agent tasks, achieving excellent results [42; 67; 21; 37]. However, in our multi-environment scenarios, reward consistency and training stability can become problematic [21; 22; 68]. Another line of work uses self-evolution/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 [69; 47; 49; 70; 20; 19; 22; 10; 71; 59]. However, like RL-based methods, these works only explore within a single environment or task. With AGENTGYM, this work explores agent evolution using the novel AGENTEVOL method, conducting exploration across multiple environments.

7 Conclusion

In this work, we present a new framework named AGENTGYM that includes an interactive platform with diverse agent environments and tasks, a benchmark named AGENTVAL, and trajectory sets called AGENTTRAJ and AGENTTRAJ-L. Additionally, we propose a novel algorithm AGENTEVOL, and take the initial step in exploring the self-evolution of generally-capable LLM-based agents across multiple environments. Empirical results demonstrate the effectiveness of our framework and our method. We also perform sufficient ablation and analysis to investigate how our method works. We hope our work can help the AI community develop more advanced generalist LLM-based agents.

References

- [1] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *Knowl. Eng. Rev.*, 10(2):115–152, 1995.
- [2] 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. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.
- [3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharmashan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [4] 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. A generalist agent. *Trans. Mach. Learn. Res.*, 2022, 2022.
- [5] 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, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huan, and Tao Gui. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864, 2023.
- [6] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- [7] Russell K Standish. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications*, 3(02):167–175, 2003.
- [8] 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. Open-ended evolution: Perspectives from the OEE workshop in york. *Artif. Life*, 22(3):408–423, 2016.

- [9] SIMA Team, Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, Stephanie C. Y. Chan, Jeff Clune, Adrian Collister, Vikki Copeman, Alex Cullum, Ishita Dasgupta, Dario de Cesare, Julia Di Trapani, Yani Donchev, Emma Dunleavy, Martin Engelcke, Ryan Faulkner, Frankie Garcia, Charles Gbadamosi, Zhitao Gong, Lucy Gonzalez, Kshitij Gupta, Karol Gregor, Arne Olav Hallingstad, Tim Harley, Sam Hayes, Felix Hill, Ed Hirst, Drew A. Hudson, Jony Hudson, Steph Hughes-Fitt, Danilo J. Rezende, Mimi Jasarevic, Laura Kampis, Nan Rosemary Ke, Thomas Keck, Junkyung Kim, Oscar Knagg, Kavya Kopparapu, Andrew K. Lampinen, Shane Legg, Alexander Lerchner, Marjorie Limont, Yulan Liu, Maria Loks-Thompson, Joseph Marino, Kathryn Martin Cussons, Loic Matthey, Siobhan McLoughlin, Piermaria Mendolicchio, Hamza Merzic, Anna Mitenkova, Alexandre Moufarek, Valéria Oliveira, Yanko Gitahy Oliveira, Hannah Openshaw, Renke Pan, Aneesh Pappu, Alex Platonov, Ollie Purkiss, David P. Reichert, John Reid, Pierre Harvey Richemond, Tyson Roberts, Giles Ruscoe, Jaume Sanchez Elias, Tasha Sandars, Daniel P. Sawyer, Tim Scholtes, Guy Simmons, Daniel Slater, Hubert Soyer, Heiko Strathmann, Peter Stys, Allison C. Tam, Denis Teplyashin, Tayfun Terzi, Davide Vercelli, Bojan Vujatovic, Marcus Wainwright, Jane X. Wang, Zhengdong Wang, Daan Wierstra, Duncan Williams, Nathaniel Wong, Sarah York, and Nick Young. Scaling instructable agents across many simulated worlds. *CoRR*, abs/2404.10179, 2024.
- [10] Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.
- [11] Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.
- [12] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.
- [13] Anthropic. The claude 3 model family: Opus, sonnet, haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf, 2024.
- [14] 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. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023.
- [15] 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. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.*, 18(6):186345, 2024.
- [16] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.
- [17] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language models. *CoRR*, abs/2403.12881, 2024.
- [18] Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, et al. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*, 2024.
- [19] Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. React meets actre: When language agents enjoy training data autonomy. *CoRR*, abs/2403.14589, 2024.

- [20] 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. Rest meets react: Self-improvement for multi-step reasoning LLM agent. *CoRR*, abs/2312.10003, 2023.
- [21] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn RL. *CoRR*, abs/2402.19446, 2024.
- [22] Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for LLM agents. *CoRR*, abs/2403.02502, 2024.
- [23] WB Langdon. Pfeiffer—a distributed open-ended evolutionary system. In *AISB*, volume 5, pages 7–13. Citeseer, 2005.
- [24] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854, 2023.
- [25] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- [26] Ruoyao Wang, Peter A. Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Science-world: Is your agent smarter than a 5th grader? In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *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, 2022.
- [27] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 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, 2019.
- [28] Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. Lmrl gym: Benchmarks for multi-turn reinforcement learning with language models, 2023.
- [29] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.
- [30] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. *CoRR*, abs/2311.05772, 2023.
- [31] Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. *CoRR*, abs/2401.13178, 2024.
- [32] 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. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.
- [33] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *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, 2023.
- [34] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *CoRR*, abs/2304.12244, 2023.
 - [35] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 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, 2023.
 - [36] 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. Agentbench: Evaluating llms as agents. *CoRR*, abs/2308.03688, 2023.
 - [37] Filippou 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. Pangu-agent: A fine-tunable generalist agent with structured reasoning. *CoRR*, abs/2312.14878, 2023.
 - [38] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. *CoRR*, abs/2310.12823, 2023.
 - [39] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
 - [40] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*, 2022.
 - [41] 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. Secrets of RLHF in large language models part I: PPO. *CoRR*, abs/2307.04964, 2023.
 - [42] Zhiheng Xi, Wenxiang Chen, Boyang Hong, Senjie Jin, Rui Zheng, Wei He, Yiwen Ding, Shichun Liu, Xin Guo, Junzhe Wang, Honglin Guo, Wei Shen, Xiaoran Fan, Yuhao Zhou, Shihan Dou, Xiao Wang, Xinbo Zhang, Peng Sun, Tao Gui, Qi Zhang, and Xuanjing Huang. Training large language models for reasoning through reverse curriculum reinforcement learning. *CoRR*, abs/2402.05808, 2024.
 - [43] Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.
 - [44] Gerhard Neumann. Variational inference for policy search in changing situations. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 817–824. Omnipress, 2011.
 - [45] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference (extended abstract). In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 3052–3056. IJCAI/AAAI, 2013.
 - [46] Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Rémi Munos, Nicolas Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

- [47] 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. Beyond human data: Scaling self-training for problem-solving with language models. *CoRR*, abs/2312.06585, 2023.
- [48] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.
- [49] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- [50] 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. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
- [51] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.
- [52] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [53] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.
- [54] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.
- [55] Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-math: Unlocking the potential of slms in grade school math. *CoRR*, abs/2402.14830, 2024.
- [56] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *CoRR*, abs/2402.06457, 2024.
- [57] Peiyi Wang, Lei Li, Liang Chen, Feifan Song, Binghuai Lin, Yunbo Cao, Tianyu Liu, and Zhifang Sui. Making large language models better reasoners with alignment. *arXiv preprint arXiv:2309.02144*, 2023.
- [58] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.
- [59] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: Bootstrap and reinforce A large language model-based web navigating agent. *CoRR*, abs/2404.03648, 2024.

- [60] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *CoRR*, abs/2310.05915, 2023.
- [61] Xuhui Zhou, Hao Zhu, Leena Mathur, Ruohong Zhang, Haofei Yu, Zhengyang Qi, Louis-Philippe Morency, Yonatan Bisk, Daniel Fried, Graham Neubig, and Maarten Sap. SOTOPIA: interactive evaluation for social intelligence in language agents. *CoRR*, abs/2310.11667, 2023.
- [62] 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. BOLAA: benchmarking and orchestrating llm-augmented autonomous agents. *CoRR*, abs/2308.05960, 2023.
- [63] Marwa Abdulhai, Isadora White, Charlie Snell, Charles Sun, Joey Hong, Yuexiang Zhai, Kelvin Xu, and Sergey Levine. LMRL gym: Benchmarks for multi-turn reinforcement learning with language models. *CoRR*, abs/2311.18232, 2023.
- [64] 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. A general language assistant as a laboratory for alignment. *CoRR*, abs/2112.00861, 2021.
- [65] 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. Training a helpful and harmless assistant with reinforcement learning from human feedback. *CoRR*, abs/2204.05862, 2022.
- [66] Binghai Wang, Rui Zheng, Lu Chen, Yan Liu, Shihan Dou, Caishuang Huang, Wei Shen, Senjie Jin, Enyu Zhou, Chenyu Shi, Songyang Gao, Nuo Xu, Yuhao Zhou, Xiaoran Fan, Zhiheng Xi, Jun Zhao, Xiao Wang, Tao Ji, Hang Yan, Lixing Shen, Zhan Chen, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Secrets of RLHF in large language models part II: reward modeling. *CoRR*, abs/2401.06080, 2024.
- [67] Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft: Reasoning with reinforced fine-tuning, 2024.
- [68] Meng Cao, Lei Shu, Lei Yu, Yun Zhu, Nevan Wichers, Yinxiao Liu, and Lei Meng. DRLLC: reinforcement learning with dense rewards from LLM critic. *CoRR*, abs/2401.07382, 2024.
- [69] Çağlar 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. Reinforced self-training (rest) for language modeling. *CoRR*, abs/2308.08998, 2023.
- [70] Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *CoRR*, abs/2308.01825, 2023.
- [71] Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*, 2024.

A Limitations

This paper proposes a novel framework named AGENTGYM. It includes an interactive platform with diverse environments and tasks, an agent benchmark AGENTVAL, and two collections of expert trajectories AGENTTRAJ and AGENTTRAJ-L. Additionally, we introduce a novel algorithm, AGENTEVOL, to explore the evolutionary capabilities of generally-capable LLM-based agents. Despite the contributions and the fact that our method performs well, our work still has some limitations. **Firstly**, for computational efficiency, we do not perform multiple samplings in each iteration. However, in the analysis in Section 5.3, we find that more sampling leads to better results, although the improvement is not significant. In the future, we hope to increase the number of samples K to a larger value when sufficient computational resources are available, to explore the upper limits of our method. **Secondly**, although we validate the effectiveness of our method on three different models (Llama2-Chat-7B, Llama-2-Chat-13B, and DeepSeek-Coder-1.3B), we hope to verify it on stronger and larger base models in the future to explore the potential for building more generally-capable agents.

B Broader Impacts

The proposed AGENTGYM and AGENTEVOL facilitate the self-evolution of generally-capable agents, and our focus is on the self-evolution of capabilities, but it is crucial to consider safety and ethical issues during usage. Agents must not be allowed to violate human values. Therefore, it is essential to strengthen supervision and regulation when eliciting agents’ self-evolution capabilities. In the future, we hope to improve the framework’s functionality to align agents with human values.

C Details of Environments in AGENTGYM

WebShop (WS) [25]. 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 SOTA models (700) and human annotations (300). For AGENTTRAJ-L, we collect 3930 trajectories with SOTA models (3430) and human annotations (500). We take the success rate as the evaluation metric and set the maximum round to 10.³

WebArena (WA) [24]. 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 25.⁴

MAZE (MZ) [28]. 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 15.⁵

³<https://github.com/princeton-nlp/WebShop/blob/master/LICENSE.md>

⁴<https://github.com/web-arena-x/webarena/blob/main/LICENSE>

⁵<https://github.com/abdulhaim/LMRL-Gym/blob/main/LICENSE>

Wordle (WD) [28]. 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) [29]. 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 have six types of tasks. We get 3827 instructions from the original work. For AGENTTRAJ, we collect 500 trajectories with SOTA models(400) and human annotations (100). For AGENTTRAJ-L, we collect 2420 trajectories with SOTA models(1920) and human annotations (500).⁶

SciWorld (Sci) [26]. 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 30.⁷

BabyAI (Baby) [27]. 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 SOTA models. For AGENTTRAJ-L, we annotate 810 trajectories with SOTA models. We take reward as the evaluation metric and set the maximum round to 20.⁸

TextCraft (TC) [30]. 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 SOTA models (254) and human annotation (46), with every action in the trajectories verified by the environment. For AGENTTRAJ-L, we annotate 374 trajectories with SOTA models (299) and human annotation (75). We take the success rate as the evaluation metric and set the maximum round to 20.⁹

⁶<https://github.com/alfworld/alfworld/blob/master/LICENSE>

⁷<https://github.com/allenai/ScienceWorld/blob/main/LICENSE>

⁸<https://github.com/mila-iqia/babyai/blob/master/LICENSE>

⁹<https://github.com/archiki/ADaPT/blob/main/LICENSE>

Weather (WT) [31]. 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 343 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 SOTA 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 SOTA models (230) and human annotators (81). We take the success rate as the evaluation metric and set the maximum round to 10.¹⁰

Movie (MV) [31]. 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 238 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) [31]. 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) [31]. 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) [31]. 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) [32]. 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

¹⁰<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.

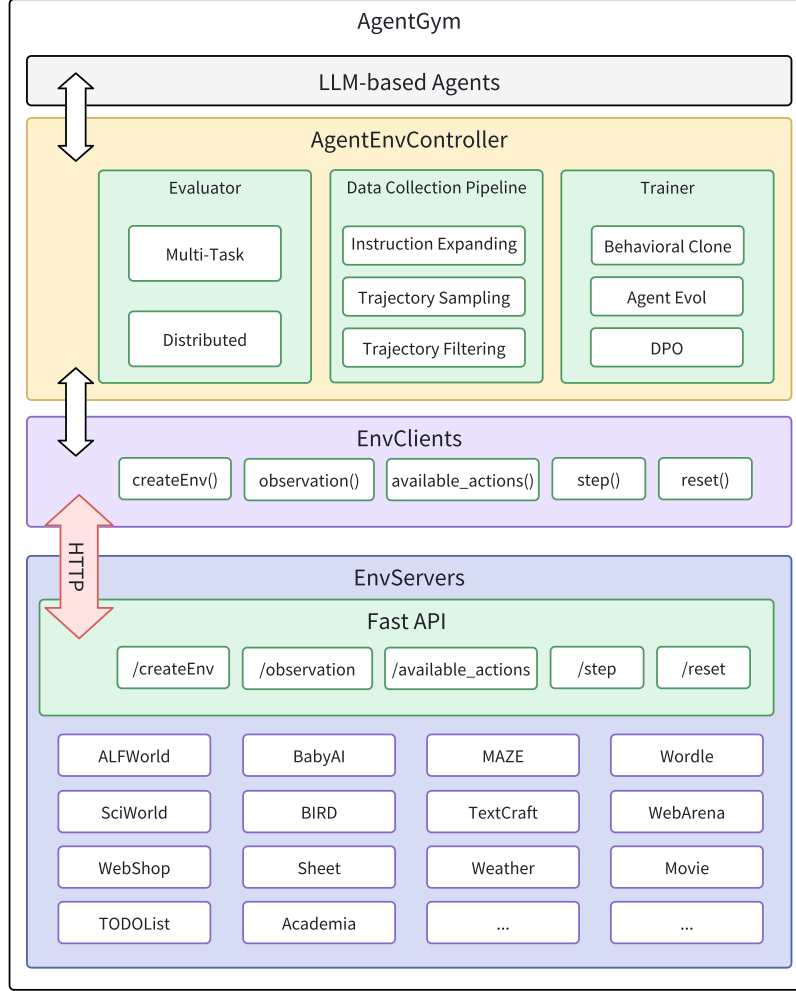


Figure 4: An illustration of the architecture of AGENTGYM platform.

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.¹¹

D Platform Architecture of AGENTGYM

The platform architecture of AGENTGYM is illustrated in Figure 4. In AGENTGYM, different environments are deployed on different servers or ports and provide encapsulated HTTP services externally. This decouples the environments from other parts. These services include APIs such as `/createEnv` to create an environment, `/observation` to get the current observation from the environment, `/available_actions` to get the currently available actions, `/step` to perform an action, and `/reset` to reset the environment. We have implemented 14 types of environments and 89 tasks, and developers can easily develop new environments and add them to AGENTGYM by encapsulating the aforementioned interfaces. EnvClients have the responsibility of receiving services provided by the server and encapsulating them into functions for user calls. AgentController is our

¹¹<https://github.com/AlibabaResearch/DAMO-ConvAI/tree/main/bird>. The bench is under a CC BY-NC 4.0 License.

core component that connects the agent and the environment. It is responsible for evaluating the agent, collecting data, and training the agent.

E More Implementation Details

The learning rate for both the BC phase and the evolution phase is set to 1×10^{-5} . For the BC baseline on the complete trajectory set, we run 3 epochs, while for evolution, we run 1 epoch per iteration. 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. In evaluation, we set the temperature to 0. In the exploration step of AGENTEVOL, we set the temperature to 0.7. With eight A100-80GB GPUs, the complete evolution process based on Llama-2-Chat-7B (four iterations) takes approximately twenty hours, and testing takes about one hour. When performing evolution with both successful and failed trajectories, we include a BC objective to stabilize the training procedure following previous work [59]. The DPO objective and the BC objective are assigned equal weights. We use a learning rate of 1×10^{-6} here because it provides more stable training.

F More Experiments

F.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 AGENTEVOL 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, AGENTEVOL achieves the best performance as well as the fewest interactive rounds.

F.2 Case Study

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

The first case is shown in Figure 5. 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 6. 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.

G Prompt Details

The prompt for each 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 8. Table 9 presents the specifications for ALFWorld. The TextCraft’s prompt details

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.

Method	WS	ALF	TC	Sci	Baby	MZ	WD	WT	MV	TL	BD
Close-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
	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-sourced 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											
BC _{base}	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
BC _{large}	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
AGENTEVOL	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

are outlined in Table 10. The prompt details for the SciWorld are shown in Table 11. The prompt details for the BabyAI are shown in Table 12. The prompt details for the MAZE are shown in Table 13. The prompt details for the Wordle are shown in Table 14. The prompt details for the BIRD are shown in Table 15. Table 16 and 17 show the prompt details for WebArena. The prompt details for the Weather are shown in Table 18, 19, 20, 21. The prompt details for the TODOList are shown in Table 22, 23. The prompt details for the Movie are shown in Table 24, 25, 26. The prompt details for the Academia are shown in Table 27, 28. The prompt details for the Sheet are shown in Table 29, 30, 31, 32.

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	After Evolution
<p>Thought 1: I think I should use the search bar to look for the pants as per the given specifics.</p> <p>Action 1: search[slim fit straight leg men's pants elastic waist long sleeve relaxed fit everyday wear black large price < 50.00]</p> <p>Observation 1: <searching results>: ... B01MSP2B0I [SEP] Wrangler Men's Retro Slim Fit Straight Leg Jean, Black, 42W x 32L [SEP] \$63.05 ...</p> <p>Thought 2: I think I should click on the 'B01MSP2B0I' option as it seems to be a men's pants that fits the description.</p> <p>Action 2: click[B01MSP2B0I]</p> <p>Observation 2: <product details>: ... Wrangler Men's Retro Slim Fit Straight Leg Jean, Black, 42W x 32L [SEP] Price: \$63.05 ...</p> <p>Thought 3: I think I should click on the 'Buy Now' button to purchase the pants.</p> <p>Action 3: click[Buy Now]</p> <p>Reward: 0.0</p>	<p>Thought 1: I think I should use the search bar to look for the pants as per the given specifics.</p> <p>Action 1: search[slim fit straight leg men's pants elastic waist long sleeve relaxed fit everyday wear black large price < 50.00]</p> <p>Observation 1: <searching results></p> <p>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.</p> <p>Action 2: click[B099231V35]</p> <p>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 ...</p> <p>Thought 3: I think I should click on the 'black' option to select the color I need.</p> <p>Action 3: click[black]</p> <p>Observation 3: <product details></p> <p>Thought 4: I think I should click on the 'large' option to select the size I need.</p> <p>Action 4: click[large]</p> <p>Observation 4: <product details></p> <p>Thought 5: I think I should click on the 'Buy Now' button to purchase the pants.</p> <p>Action 5: click[Buy Now]</p> <p>Reward: 1.0</p>

Figure 5: 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.

<p>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"]</p>	
Before Evolution	After Evolution
<p>Thought 1: I need to find the green box to pick it up. I should check my surroundings to locate it.</p> <p>Action 1: check available actions</p> <p>Observation 1: You can take the following actions: turn left, turn right, check available actions. Available actions: ["turn left", "turn right", "check available actions"]</p> <p>Thought 2: I need to find the green box to pick it up. I should turn left to explore a different direction.</p> <p>Action 2: turn left</p> <p>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...</p> <p>Thought 3: I need to pick up the green box to complete my goal. I should move forward to reach it.</p> <p>Action 3: move forward</p> <p>Observation 3: <room details></p> <p>Thought 4: I need to pick up the green box to complete my goal. I should move forward to reach it.</p> <p>Action 4: move forward</p> <p>Observation 4: <room details></p> <p>...(more interactions)</p> <p>Thought 20: I need to pick up the green box to complete my goal. I should move forward to reach it.</p> <p>Action 20: move forward</p> <p>...(reach max_interaction_rounds)</p> <p>Reward: 0.0</p>	<p>Thought 1: I need to find the green box in this room.</p> <p>Action 1: check available actions</p> <p>Observation 1: You can take the following actions: turn left, turn right, check available actions. Available actions: ["turn left", "turn right", "check available actions"]</p> <p>Thought 2: I need to turn left to explore the room further and find the green box.</p> <p>Action 2: turn left</p> <p>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"]</p> <p>Thought 3: I need to pick up the green box to complete my goal.</p> <p>Action 3: pickup green box 1</p> <p>Reward: 1.0</p>

Figure 6: 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.

Table 8: Prompt details for WebShop.

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 9: Prompt details for ALFWorld.

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 10: Prompt details for TextCraft.

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 11: Prompt details for SciWorld.

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 12: Prompt details for BabyAI.

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 13: Prompt details for MAZE.

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 14: Prompt details for Wordle.

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 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 15: Prompt details for BIRD.

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

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.

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.
-

Table 17: Prompt details for WebArena (Part 2/2).

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

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_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.

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.

Table 19: Prompt details for Weather (Part 2/4).

<p>Name: <code>get_current_snow(latitude, longitude, current_date)</code> 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.</p>
<p>Name: <code>get_current_temp(latitude, longitude, current_date)</code> 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.</p>
<p>Name: <code>get_latitude_longitude(name)</code> 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.</p>
<p>Name: <code>get_elevation(latitude, longitude)</code> 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.</p>
<p>Name: <code>get_temp_forecast(latitude, longitude, start_date, end_date)</code> 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.</p>
<p>Name: <code>get_rain_forecast(latitude, longitude, start_date, end_date)</code> 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.</p>

Table 20: Prompt details for Weather (Part 3/4).

<p>Name: <code>get_current_rain(latitude, longitude, current_date)</code> 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.</p>
<p>Name: <code>get_distance(latitude1, longitude1, latitude2, longitude2)</code> 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.</p>
<p>Name: <code>get_historical_air_quality_index(latitude, longitude, start_date, end_date)</code> 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.</p>
<p>Name: <code>get_current_air_quality_index(latitude, longitude, current_date)</code> 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.</p>
<p>Name: <code>get_air_quality_level(air_quality_index)</code> 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'.</p>
<p>Name: <code>check_valid_actions()</code> Description: Get supported actions for current tool. Returns: - actions (Type: array): Supported actions for current tool.</p>
<p>Name: <code>finish(answer)</code> Description: Return an answer and finish the task Parameters: - answer (Type: ['string', 'number', 'array']): The answer to be returned</p>

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

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 22: Prompt details for TODOList (Part 1/2).

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_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.

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.
-

Table 23: Prompt details for TODOList (Part 2/2).

<p>Name: <code>get_task_duration(task_id)</code> Description: Get the duration of a specific task in the TodoList account. Parameters: - <code>task_id</code> (Type: string) Returns: - <code>id</code> (Type: string) - <code>content</code> (Type: string): Name of the task. - <code>duration</code> (Type: string): Duration of the task in the format of 'amount(unit)'.</p>
<p>Name: <code>complete_task(task_id)</code> Description: Mark a task as completed Parameters: - <code>task_id</code> (Type: string) Returns: information of the completed task</p>
<p>Name: <code>update_task(task_id, due_date)</code> Description: Update a task Parameters: - <code>task_id</code> (Type: string) - <code>due_date</code> (Type: string) Returns: Information of the updated task</p>
<p>Name: <code>delete_task(task_id)</code> Description: Delete a specific task from the TodoList account. Parameters: - <code>task_id</code> (Type: string): Unique identifier of the task to delete. Returns: Information of the deleted task.</p>
<p>Name: <code>check_valid_actions()</code> Description: Get supported actions for current tool. Returns: Supported actions for current tool.</p>
<p>Name: <code>finish(answer)</code> Description: Call this action, when find the answer for the current task or complete essential operations. Parameters: - <code>answer</code> (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'</p>
<p>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]"</p>
<p>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.</p>

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

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.

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
-

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

<p>Name: <code>get_movie_keywords(movie_id)</code> Description: Get the keywords associated with a movie by ID Parameters: - <code>movie_id</code> (Type: string): The ID of the movie. Returns: - <code>keywords</code> : The keywords associated with the movie.</p>
<p>Name: <code>get_search_person(person_name)</code> Description: Search for a person by name. Parameters: - <code>person_name</code> (Type: string): The name of the person to search for. Returns: - <code>id</code> : The ID of the found person. - <code>name</code> : The name of the person.</p>
<p>Name: <code>get_person_details(person_id)</code> Description: Get detailed information about a person by ID Parameters: - <code>person_id</code> (Type: string): The ID of the person. Returns: - <code>biography</code> : The biography of the person. - <code>birthday</code> : The birthday of the person. - <code>place_of_birth</code> : The place of birth of the person.</p>
<p>Name: <code>get_person_cast(person_id)</code> Description: Retrieve the top 10 movie cast roles of a person by their ID Parameters: - <code>person_id</code> (Type: string): The ID of the person. Returns: - <code>cast</code> : A list of movies where the person has acted, limited to top 10</p>
<p>Name: <code>get_person_crew(person_id)</code> Description: Retrieve the top 10 movie crew roles of a person by their ID Parameters: - <code>person_id</code> (Type: string): The ID of the person. Returns: - <code>crew</code> : A list of movies where the person has participated as crew, limited to top 10</p>
<p>Name: <code>get_person_external_ids(person_id)</code> Description: Get the external ids for a person by ID Parameters: - <code>person_id</code> (Type: string): The ID of the person. Returns: - <code>imdb_id</code> : The IMDB id of the person. - <code>facebook_id</code> : The Facebook id of the person. - <code>instagram_id</code> : The Instagram id of the person. - <code>twitter_id</code> : The Twitter id of the person.</p>
<p>Name: <code>get_movie_alternative_titles(movie_id)</code> Description: Get the alternative titles for a movie by ID Parameters: - <code>movie_id</code> (Type: string): The ID of the movie. Returns: - <code>titles</code> : The alternative titles of the movie. - <code>id</code> : The ID of the movie.</p>

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

<p>Name: <code>get_movie_translation(movie_id)</code> Description: Get the description translation for a movie by ID Parameters: - <code>movie_id</code> (Type: string): The ID of the movie. Returns: - <code>translations</code> : The description translation of the movie. - <code>id</code> : The ID of the movie. Name: <code>check_valid_actions()</code> Description: Get supported actions for current tool. Returns: - <code>actions</code> (Type: array): Supported actions for current tool.</p>
<p>Name: <code>finish(answer)</code> Description: Return an answer and finish the task Parameters: - <code>answer</code> (Type: ['string', 'number', 'array']): The answer to be returned</p>
<p>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]"</p>
<p>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.</p>

Table 27: Prompt details for Academia (Part 1/2).

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: 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, neighbours are cited papers of the paper. In authorNet, neighbours 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 published 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 28: Prompt details for Academia (Part 2/2).

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

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

<p>Name: delete_batch_data(dimension, index_list)</p> <p>Description: Delete a batch of data in the sheet</p> <p>Parameters:</p> <ul style="list-style-type: none"> - 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. <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.
<p>Name: update_cell(position, value)</p> <p>Description: Update the value of the cell</p> <p>Parameters:</p> <ul style="list-style-type: none"> - position (Type: string): A1 notation of the cell position. - value: The value to set. <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.
<p>Name: update_cell_by_formula(start_position, end_position, position_list, result_position, operator)</p> <p>Description: Update the value of the target cell by applying formulas on some specified cells.</p> <p>Note: Either specify position_list or start_position and end_position.</p> <p>Parameters:</p> <ul style="list-style-type: none"> - 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']. <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.
<p>Name: update_range(start_position, end_position, values_list)</p> <p>Description: Update a range of the cells from a list</p> <p>Parameters:</p> <ul style="list-style-type: none"> - 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 <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.
<p>Name: sort_sheet_by_col(col_num, order)</p> <p>Description: Sorts the current sheet using given sort orders</p> <p>Parameters:</p> <ul style="list-style-type: none"> - col_num (Type: integer): The index of the sort column. - order (Type: string): The sort order. Possible values are 'asc' or 'des'. <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.
<p>Name: merge_cells(start_position, end_position)</p> <p>Description: Merge cells in sheet</p> <p>Parameters:</p> <ul style="list-style-type: none"> - start_position (Type: string): Starting cell position(top left) in A1 annotation. - end_position (Type: string): Ending cell position(bottom right) in A1 annotation. <p>Returns:</p> <ul style="list-style-type: none"> - result (Type: object): The updated worksheet data or an error message.

Table 31: Prompt details for Sheet (Part 3/4).

<p>Name: <code>update_note(position, content)</code> 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.</p>
<p>Name: <code>get_all_values()</code> Description: Display all cell values in current sheet Returns: - result (Type: array[array[Any]]): Return all cell values or an error message.</p>
<p>Name: <code>get_range_values(start_position, end_position)</code> 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.</p>
<p>Name: <code>get_cell_value(position)</code> 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.</p>
<p>Name: <code>get_value_by_formula(start_position, end_position, position_list, operator)</code> 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.</p>
<p>Name: <code>filter_cells(query, in_row, in_column)</code> 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.</p>
<p>Name: <code>get_note(position)</code> 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.</p>

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

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.