AGENTBOARD: AN ANALYTICAL EVALUATION BOARD OF MULTI-TURN LLM AGENTS

Chang Ma[*♠] **Junlei Zhang**[*◇△] **Zhihao Zhu**[*♡] **Cheng Yang**[*♣]
Yujiu Yang[♣] **Yaohui Jin**[♡] **Zhenzhong Lan**[△] **Lingpeng Kong**[♠] **Junxian He**[★]
[♠]The University of Hong Kong [◇]Zhejiang University [♡]Shanghai Jiao Tong University
[♣]Tsinghua University [△] School of Engineering, Westlake University
[★]The Hong Kong University of Science and Technology
llmagentboard@gmail.com

ABSTRACT

Evaluating Large Language Models (LLMs) as general-purpose agents is essential for understanding their capabilities and facilitating their integration into practical applications. However, the evaluation process presents substantial challenges. A primary obstacle is the benchmarking of agent performance across diverse scenarios within a unified framework, especially in maintaining partially-observable environments and ensuring multi-round interactions. Moreover, current evaluation frameworks mostly focus on the final success rate, revealing few insights during the process and failing to provide a deep understanding of the model abilities. To address these challenges, we introduce AGENTBOARD, a pioneering comprehensive benchmark and accompanied open-source evaluation framework tailored to analytical evaluation of LLM agents. AGENTBOARD offers a fine-grained progress rate metric that captures incremental advancements as well as a comprehensive evaluation toolkit that features easy assessment of agents for multi-faceted analysis. This not only sheds light on the capabilities and limitations of LLM agents but also propels the interpretability of their performance to the forefront. Ultimately, AGENTBOARD serves as a step towards demystifying agent behaviors and accelerating the development of stronger LLM agents.[1]

1 INTRODUCTION

General-purpose agents that can autonomously perceive and act in various environments are considered significant milestones in Artificial Intelligence (Russell & Norvig, 2005). Recent advancements in large language models (OpenAI, 2023; Touvron et al., 2023) have demonstrated emergent agent abilities that enable them to understand diverse environments and perform step-by-step planning through multi-round interactions. (Yao et al., 2023; Song et al., 2023). These advanced abilities contribute to the potential of LLMs to act as generalist agents for real-world problem-solving.

A comprehensive evaluation of LLM agents is crucial for the progression of this emerging field. To start, *task diversity* is necessary to cover various agent tasks such as embodied, web, and tool agents. Additionally, *multi-round interaction* is critical to mimic realistic scenarios, in contrast to the single-round tasks commonly adopted in existing benchmarks (Xu et al., 2023a; Lin & Chen, 2023; Qin et al., 2023a). Furthermore, evaluating agents in *partially-observable* environments, where they must actively explore to understand their surroundings, is essential for practical assessments. This differs from the "synthetic" agent tasks (Wang et al., 2023b) derived from conventional benchmarks in fully-observable environments, such as MMLU (Lanham et al., 2023) and MATH (Hendrycks et al., 2021). However, existing agent benchmarks often fail to satisfy all of these criteria.

Moreover, the inherent complexity in agent tasks, characterized by multi-round interactions, decision-making based on long context, and the achievement of various subgoals, distinguishes them significantly from other language tasks. Due to this complexity, there is a pressing need to

[*]Equal Contribution. Individual contributions are listed in Appendix A. Work done during visit to HKUST.
[1]Code and data are available at https://github.com/hkust-nlp/AgentBoard

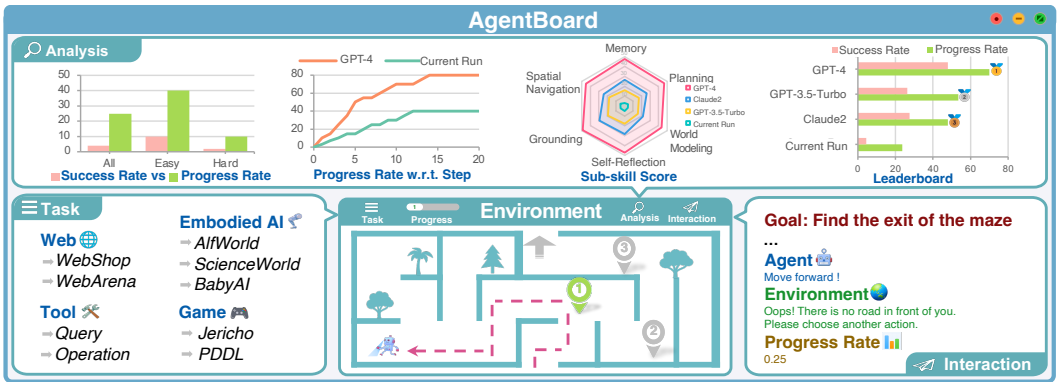


Figure 1: The illustrative overview of AGENTBOARD. AGENTBOARD consists of a 9 diverse tasks. Agents interact in multi-rounds with partially-observable environments to achieve each subgoal. Furthermore, AGENTBOARD provides an open-source analytical evaluation framework, as shown in the figure.

delve into the details and gain a deeper understanding of how models function during the process. Nonetheless, most current evaluations predominantly rely on the final success rate as their metric, which provides limited insights into these intricate processes (Liu et al., 2023a; Wang et al., 2023b; Yao et al., 2023; Liu et al., 2023b; Mialon et al., 2023). This simplified evaluation is particularly inadequate in challenging environments where most models demonstrate nearly zero success rates, consequently blurring finer distinctions and obscuring underlying mechanisms (Liu et al., 2023a).

To address these issues, we introduce AGENTBOARD, a benchmark designed for multi-turn LLM agents, complemented by an analytical evaluation board for detailed model assessment beyond final success rates. AGENTBOARD encompasses a diverse set of 9 unique tasks and 1013 exemplary environments, covering a range from embodied AI and game agents to web and tool agents. Each environment, whether newly created or adapted from pre-existing ones, is carefully crafted and authenticated by humans to ensure multi-round and partially observable characteristics in a unified manner. Notably, we have defined or manually annotated subgoals for each data sample, introducing a unified *progress rate* metric to track the agents' detailed advancements. As we will demonstrate in §5.2, this metric uncovers significant progress made by models that would otherwise appear trivial due to negligible differences in success rates.

Along with the benchmark, we develop the AGENTBOARD evaluation framework as an open-source toolkit that features an analytical web panel to examine various dimensions of agent abilities through interactive visualization. The toolkit offers a unified interface, providing users with easy access and effortless customization options. As partially shown in Figure 1, the AGENTBOARD panel currently supports analysis and visualization on fine-grained progress rates tracking, performance breakdown for hard and easy examples, detailed performance across various sub-skills, long-range interaction assessment, grounding accuracy, and trajectory. This detailed evaluation is crucial for acknowledging the progress of LLM agents and for guiding the development of more robust LLM agent models. The comparison between AGENTBOARD and previous works is shown in Table 1.

We assess a wide series of proprietary and open-weight LLM agents through AGENTBOARD and derive a clear perspective on current LLM models as agents: (1) GPT-4 outperforms all other models by exhibiting extensive proficiency across a range of tasks and distinct agentic abilities, while open-weight code-LLMs are catching up with commercial models, with DeepSeek LLM (DeepSeek-AI et al., 2024) and Lemur (Xu et al., 2023b) taking the lead; (2) Strong LLM agents are characterized by their capability for *multi-turn* interaction with the environment, an ability that is notably lacking in most open-weight models; (3) Current proprietary models typically demonstrate comprehensive agentic abilities, while open-weight LLMs show varying deficiencies in grounding, world modeling, and self-reflection. Through AGENTBOARD, we highlight the importance of analytic evaluation of LLM agents. We expect that the detailed assessments from AGENTBOARD and the open-source AGENTBOARD toolkit will help push further advancements of LLM agents.

2 RELATED WORK

LLM as Agent Traditional Reinforcement Learning provides general solutions for decision-making but lacks sample efficiency and generalization (Pourchot & Sigaud, 2019). The emergent reasoning

Benchmarks	Task Diversity	Multi-round Interaction	Partially-Observable Environments	Fine-grained Progress Metrics	Analytical Evaluation
AgentBench (Liu et al., 2023a)	✔	✗‡	✔	✗	✗
GAIA (Mialon et al., 2023)	✗§	✔	✔	✗	✗
MINT (Wang et al., 2023b)	✔	✔	✗†	✗	✗
API-Bank (Li et al., 2023)	✗	✔	✔	✗	✗
ToolEval (Qin et al., 2023b)	✗	✔	✔	✗	✗
LLM-Eval (Lin & Chen, 2023)	✔	✗	✗	✗	✗
AGENTBOARD	✔	✔	✔	✔	✔

Table 1: AGENTBOARD differs from other LLM benchmarks by providing a comprehensive framework that fully integrates all four guiding principles within its evaluation system. ‡Notably, AgentBench entails both single and multi-round tasks, with only the former differentiating open-sourced models. §The GAIA benchmark focuses solely on question answering tasks that involve interacting with the real world. †MINT benchmark primarily includes fully-observable environments tasks derived from conventional evaluations such as HumanEval and GSM8K.

and instruction-following abilities of LLMs (Wei et al., 2022) enable them to become proficient agents, excelling in zero-shot generalization (Yao et al., 2023; Richards, 2023; Wang et al., 2023a). The central method for employing LLMs as agents is to prompt them with task instructions and environmental context, enabling them to produce actionable responses (Richards, 2023; Xie et al., 2023). Other methods involve specialized training to repurpose LLMs into adept agents (Xu et al., 2023b; Reed et al., 2022; Driess et al., 2023). We benchmark both general (OpenAI, 2023; Touvron et al., 2023; Chiang et al., 2023) and agent LLMs (Xu et al., 2023b) to study LLMs as agents. Meanwhile, several work addresses dimensional aspects of agentic abilities, with a focus on the ability to ground goals to executable actions (Gu et al., 2022; Ahn et al., 2022), the ability to model the world (LeCun, 2022), the ability to perform step-by-step planning (Song et al., 2023), and self-reflection ability (Madaan et al., 2023; Wang et al., 2023b). Examining various agentic skills is crucial to fully understand the advantages and limitations of LLMs as agents.

Evaluating LLM in Decision Making Problems Several benchmarks and toolkits for LLM agents have been established, focusing on various tasks such as web-browsing, games, and tool use (Yao et al., 2022; Zhou et al., 2023; Shridhar et al., 2021; Qin et al., 2023a; Wang et al., 2023a; Ye et al., 2024; Kinniment et al., 2023). A few other benchmarks provide a proof-of-concept study on specific LLM features, with Wang et al. (2023b) focusing on model interaction ability, and Liu et al. (2023b) examining agent structures. Recent works by Liu et al. (2023a); Wu et al. (2023); Mialon et al. (2023) present a generalist challenge for LLM agents, please refer to Table 1 for a comparison. Note that recent progress in multimodal LLMs has spurred research into multimodal LLM agents (Zheng et al., 2024; Yang et al., 2023). Our study focuses exclusively on text-based environments to assess LLM agent abilities via textual reasoning and actions in-depth.

3 AGENTBOARD – OVERVIEW

AGENTBOARD is designed around the core principles of uniformity and user-friendliness. Beyond a text-based agent evaluation benchmark, we develop AGENTBOARD aiming for a readily accessible, open-source evaluation framework that facilitates diverse analysis and easy customization for different models, agents, and environments, all within a unified format. This commitment to uniformity manifests in three key areas:

- *Interface Uniformity*, where we present a consistent interface implementation across various environments, model deployments, and prompt types;
- *Observation and Action Space Uniformity*, ensuring datasets are uniformly constructed for text-based interactions with unified metrics;
- *Analysis Uniformity*, which expands metrics beyond mere success rate and scores to include detailed analyses. As illustrated in Figure 1, such a comprehensive evaluation includes assessment on (1) fine-grained progress rates tracking different agents, (2) grounding accuracy, (3) performance breakdown for hard and easy examples, (4) long-range interactions, (5) analyses of performance across various sub-skills, and (6) trajectory with friendly visualization. We elaborate these analyses in our experiments at §5. Additionally, AGENTBOARD provides an web interface through Wandb dashboard that offers interactive visualizations of these analyses during evaluation[2] and a case study in §6.

[2]An example of the panel is public here.

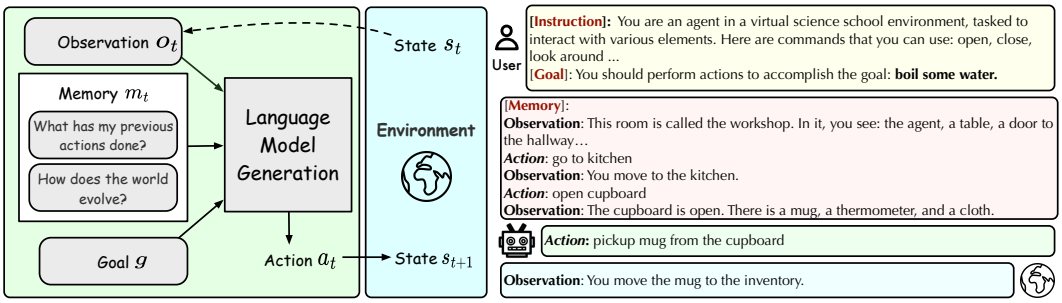


Figure 2: (Left) A structural overview of the reflex agent, which iteratively interacts with the environment and makes next step predictions based on the goal and history. (Right) An example of a prompt that queries the LLM to behave as our reflex agent.

3.1 PRELIMINARIES

An LLM agent receives textual world descriptions, chooses a text action, and gets feedback detailing state changes and any action errors. Interaction with these environments can be modeled as a special case of Partially Observable Markov Decision Processes (POMDPs) defined by tuple $\langle g, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T} \rangle$, with goal g, state space $\mathcal{S}$, valid actions space $\mathcal{A}$, observation space (including environment feedback) $\mathcal{O}$, transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. An agent with policy π makes prediction at time step t based on goal g and memory $m_t = \{o_j, a_j, o_{j+1}, a_{j+1}, \ldots o_t\}, 0 \leq j < t$, which is a sequence of actions and observations. This trajectory of the agent $\tau = [s_0, a_0, s_1, a_1, \ldots s_t]$ is formulated by policy and environmental state transitions, such as

$$p_\pi(\tau) = p(s_0) \prod_{t=0}^{T} \pi(a_t|g, s_t, m_t) \mathcal{T}(s_{t+1}|s_t, a_t) \tag{1}$$

The agent's structure is detailed in §3.2, and interactive environment designs in Section 4.

3.2 A UNIFIED REFLEX AGENT

AGENTBOARD unifies all tasks around a general framework where the agent receives observations o_t and performs actions a_t, causing deterministic state transitions $\mathcal{T} : (s_t, a_t) \rightarrow s_{t+1}$ based on real-world dynamics. A feedback function f is also defined in the environment to derive feedback from each interaction $o_t = f(s_t, a_t)$. This feedback includes: (1) list all valid actions when the agent uses help actions such as *check valid actions*; (2) execute valid action a_t and return a description of the changed stete s_{t+1}; (3) issue an error when the agent performs an action outside of the action space.

As shown in Figure 2, our agent makes decisions based on its memory of past perceptions, similar to how humans learn from experience and adapt. The implementation of the reflex agent assessed in this paper adopts an act-only prompting strategy in line with recent studies (Liu et al., 2023b; Zhou et al., 2023; Xu et al., 2023a), detailed in the right part of Figure 2, while other prompting strategy can be easily incorporated into our open-source framework. The prompt template and content are kept consistent across various LLMs, with occasional minor adjustments to fit specific model requirements. Also, LLM agents tend to struggle with limited context lengths in long interactions, failing to retain full history. Following the *"sliding window"* method from LangChain (Chase, 2022), we focus on recent, more impactful interactions (Puterman, 1990) within context constraints. This differs from previous practices that stop the agent when context limits are surpassed (Liu et al., 2023a; Wang et al., 2023b), allowing for extended, intricate interactions in our approach.

3.3 FINE-GRAINED PROGRESS RATE

Recent studies highlight the predominant use of success rate as the main metric for agent evaluation, which fails to capture the nuances of partial task completion by language model agents (Liu et al., 2023a; Li et al., 2023). This approach does not differentiate between near-complete tasks and minimal task execution, treating both as equivalent failures. Alternative metrics like reward scores are available but lack standardization, complicating cross-environment comparisons (Chevalier-

Task	Goal & Trajectory
ALF	**Goal: put a clean egg in microwave.** Step 02 **Action:** open fridge 1 **Observation:** You open the fridge 1. The fridge 1 is open. In it, you see a apple 2, a egg 1, a lettuce 1, a pan 2, a plate 1, and a tomato 1. Progress Rate: 0.00 → 0.25
JC	**Goal: Get out of the house. Then escape the city without getting caught via driving.** Step 29 **Action:** take a shower **Observation:** You step into the shower, turn on the water, and within a few moments you feel like a new man. But no time to dawdle - you hop back out again and dry off in record time . . . Progress Rate: 0.43 → 0.57
WA	**Goal: Display the list of issues in the kkroening/ffmpeg-python repository that have labels related to questions** Step 05 **Action:** click [5398] **Observation:** Tab 0 (current): Issues · Karl Kroening / ffmpeg-python · GitLab [6573] RootWebArea 'Issues · Karl Kroening / ffmpeg-python · GitLab' focused: True [6620] link . . . Progress Rate: 0.25 → 0.50
TO	**Goal: In "Sheet17", calculate and complete the "Profit" of the products in the table based on the sales information of the products. And then, sort the table in descending order by "Profit".** Step 07 **Action:** update_cell_by_formula with Action Input: {"operator": "PRODUCT", " start_position": "C8", "end_position": "D8", "result_position": "E8"} **Observation:** [['Product', 'Category', 'Price', 'Sold out', . . . Progress Rate: 0.47 → 0.49

Table 2: Examples of goals for the 4 task categories in AGENTBOARD, along with a sampled step of the trajectory and progress rate. The trajectory is generated by GPT-4. Some lengthy observations are omitted with ". . ." for brevity. The task name in the table uses an abbreviation, the full name can be found in §4.

Boisvert et al., 2019; Wang et al., 2022). We introduce a *progress rate* metric to accurately reflect language model agents' goal attainment at various stages in a unified manner.

In each round of interaction, a progress rate, denoted as r_t, is assigned to evaluate the agent's advancement towards the goal state g. As the agent moves through the states $\mathbf{s_t} = [s_0, \ldots, s_t]$, we assess its progress using a matching score $f(\cdot, g) \rightarrow [0, 1]$ that quantifies the similarity between the current state and the goal state. The initial value of r_t is set to 0, indicating no progress. The progress rate r_t reflects the highest matching score achieved, reaching 1 when the task is completed. The progress rate is formulated as below:

$$r_t = \begin{cases} r_t^{\text{match}} = \max\limits_{i, 0 \leq i \leq t} f(s_i, g), & \text{if } f(\cdot, g) \text{ is continuous} \\ r_t^{\text{subgoal}} = \max\limits_{i, 0 \leq i \leq t} \left(\dfrac{1}{K} \sum\limits_{k=1}^{K} f(s_i, g_k) \right), & \text{otherwise} \end{cases} \tag{2}$$

The function $f(\cdot, g)$ measures state similarity in tasks, such as comparing table states in manipulation activities. It works well for tasks with direct state comparisons but is less effective for tasks with ambiguous intermediate states, where progress is hard to measure. We mitigate this by introducing a discrete matching score to assess how closely intermediate states align with defined subgoals. We begin by decomposing the overall goal g into a sequence of subgoals $\mathbf{g} = [g_1, \ldots, g_K]$, where each subgoal precedes the next. We manually edit 5% of problems for a simpler setup where each final goal aligns with a unique subgoal sequence. Note that we only maintain a unique subgoal sequence, yet this allows a diverse set of trajectories, e.g. take detours when accomplishing the task. As an example, if the goal of the task is "clean an egg and put it in microwave". The necessary subgoals would be "open the fridge" → "taking an egg from the fridge" → "clean the egg with sinkbasin" → "put the egg in the microwave". Each subgoal g_i is associated with a labeled state that indicates its completion. To evaluate the match between an agent state and a subgoal, we employ a regular-expression-based matching function denoted as $f(\cdot, g_i) \rightarrow \{0, 1\}$ and the progress rate as r_t^{subgoal} in Equation 2.

4 AGENTBOARD – TASK COMPOSITION

In this section, we introduce the tasks in AGENTBOARD, the adaptations we made to accommodate our design principles, as well as the annotation process of subgoals. AGENTBOARD is composed of four types of environments: embodied, game, web, and tool. The examples of goals and trajectories are illustrated in Table 2 and the summary of task statistics is available in Table 10. We briefly describe these tasks next, while we leave more details on these environments in Appendix G and H.

4.1 EMBODIED ENVIRONMENTS

AlfWorld (ALF) (Shridhar et al., 2021) are Household tasks that require models to explore rooms and use commonsense reasoning to perform tasks, such as "put a pencil on the desk". To calculate the progress rate, we defined subgoals based on necessary observations to finish a task and the success flag provided by environments.

ScienceWorld (SW) (Wang et al., 2022) is a challenging interactive text environment testing scientific commonsense with tasks like "measure the melting point of the orange juice". The current subgoals provided by SW do not accurately reflect a language model's performance due to their sparsity and uneven weighting, as further explained in Appendix H.2. To rectify this, we re-annotate the subgoals to create a more evenly distributed reward system ($r_t^{subgoal}$). We also imposed restrictions on tool usage and task completion rooms to ensure these subgoals are essential for accomplishing the final goals.

BabyAI (BA) (Chevalier-Boisvert et al., 2019) is an interactive 20x20 grid environment where agents navigate and interact with objects within a limited sight range. The original setup uses image-based observations and tensor-based actions like "0: move left". We adapted it to include a textual action space and descriptive textual observations. Furthermore, we introduced a subgoal-based progress rate to enhance reward frequency over the original scoring system.

4.2 GAME ENVIRONMENTS

Jericho (JC) (Hausknecht et al., 2020) is a collection of text-based game environments that evaluate agents to perform adventures in fictional worlds. This task is unique in that it requires strong world modeling ability as agents could only gain information about the magic world through exploration and interaction. The original games are quite long (need 50-300 steps to finish), which is not suitable for LLM agents with fixed context length. To solve this issue, we rewrite the goal of each adventure to restrict the games to be finished within 15 subgoals.

PDDL (PL) (Vallati et al., 2015) is a set of strategic games defined with Planning Domain Definition Language (PDDL). We selected 4 representative games, *Gripper, Barman, Blocksworld, Tyreworld* to benchmark LLM agents in diverse scenarios. We adapted the environment implementation (Silver & Chitnis, 2020) written in PDDL expressions to provide text-based observations for agents, enabling Large Language Models (LLMs) to interact using natural language. We measure progress using a matching score, r_t^{match}, which assesses similarity between the current and goal states.

4.3 WEB-BASED ENVIRONMENTS

WebShop (WS) (Yao et al., 2022) is a network-based simulation environment for e-commerce experiences. We measure the distance of the current state to the final goal as the progress rate and expand the product scoring rules from Yao et al. (2022) to derive the score (H.6).

WebArena (WA) (Zhou et al., 2023) is a real web environment. To obtain the progress rate, we revised the existing method for calculating the final score (Zhou et al., 2023) and continuously computed the progress rate at each step, fusing the URL matching score with the content matching score, as detailed in Appendix H.7.

4.4 TOOL ENVIRONMENTS

Tool-Query (TQ) consists of three sub-environments: Weather, Movie and Academia Environment. Tools involved primarily serve the purpose of querying information about weather, movie and computer science academia. To compute the progress rate r_t^{subgoal}, the authors manually identify golden actions for each goal. Outputs returned by executing these actions are then processed as subgoals.

Tool-Operation (TO) includes two sub-environments: Todo and Sheet. They use tools to access and modify information. The progress rate in the Todo Environment is measured using r_t^{subgoal}, similar to Tool-Query Environments. In the Sheet Environment, progress is evaluated using r_t^{match}, which is based on the match between the current and the manually annotated golden spreadsheet.

Model	Embodied AI			Game		Web		Tool		Avg.
	ALF	SW	BA	JC	PL	WS	WA	TQ	TO	
GPT-4	**65.5/43.3**	**78.8/52.2**	**70.7/56.2**	**52.4/35.0**	**81.2/61.7**	**76.5/39.0**	**39.4/15.1**	**85.1/68.3**	**80.8/60.0**	**70.0/47.9**
Claude2	34.1/24.6	32.0/11.1	48.1/37.5	20.4/ 0.0	61.4/40.0	74.6/37.8	36.4/ 8.6	73.5/48.3	59.6/27.5	48.9/26.2
GPT-3.5-Turbo	35.6/17.2	31.9/18.9	51.7/39.3	19.9/ 5.0	25.0/ 5.0	76.4/35.1	25.5/ 4.6	69.4/45.0	37.2/ 7.5	41.4/19.7
DeepSeek-67b	34.5/20.9	36.1/10.0	31.7/22.3	13.7/ 0.0	22.0/ 6.7	72.7/31.9	23.9/ 5.7	71.4/40.0	40.5/17.5	38.5/17.2
Text-Davinci-003	18.8/ 9.0	28.9/ 7.8	17.5/14.3	28.6/10.0	31.7/11.7	72.3/29.5	16.2/ 2.5	65.0/38.3	56.2/22.5	37.2/16.2
GPT-3.5-Turbo-16k	25.2/ 4.5	2.2/ 0.0	45.1/33.9	16.1/ 0.0	22.6/ 3.3	73.8/27.9	23.7/ 6.1	59.1/31.7	39.6/15.0	34.2/13.6
Lemur-70b	10.8/ 0.7	33.4/ 5.6	19.4/ 9.8	10.1/ 0.0	9.7/ 3.3	71.8/11.6	12.2/ 3.3	72.0/28.3	37.7/12.5	30.8/ 8.3
CodeLlama-34b	11.3/ 3.0	3.5/ 0.0	19.9/13.4	15.5/ 0.0	18.5/ 3.3	71.7/23.5	21.2/ 4.1	60.0/13.3	48.8/ 7.5	30.0/ 7.6
CodeLlama-13b	13.4/ 2.2	9.6/ 2.2	22.2/17.0	0.0/ 0.0	9.3/ 1.7	65.5/25.9	17.7/ 3.7	52.5/25.0	41.8/12.5	25.8/10.0
Llama2-70b	13.2/ 3.0	2.6/ 0.0	30.0/19.6	7.8/ 0.0	8.1/ 1.7	53.6/13.1	11.6/ 3.3	48.3/ 0.0	38.6/ 0.0	23.8/ 4.5
Mistral-7b	9.8/ 0.0	15.8/ 2.2	20.1/14.3	11.0/ 0.0	4.7/ 0.0	68.2/13.9	13.2/ 1.3	51.0/ 3.3	27.2/ 0.0	24.6/ 3.9
Vicuna-13b-16k	11.0/ 1.5	14.1/ 2.2	14.3/ 5.4	15.2/ 0.0	7.2/ 1.7	73.3/21.9	11.3/ 2.9	34.3/ 3.3	26.9/ 0.0	23.1/ 4.3
Llama2-13b	7.8/ 0.0	1.1/ 0.0	18.1/ 6.2	3.2/ 0.0	4.1/ 0.0	63.5/10.8	7.9/ 2.0	35.1/ 0.0	29.3/ 0.0	18.9/ 2.1

Table 3: Performance of different LLMs across various tasks. The models are sorted in descending order, in terms of progress rates. "A/B" indicates "progress rate" and "success rate" metrics. We benchmark the chat version of the open-weight models, please refer to Appendix E for specific version of the models.

4.5 Annotation Validation

We conducted manual validation of our labeled subgoals through a custom interactive UI and multiple rounds of checks. Initially, annotators self-reviewed their work, followed by an external review by two annotators. Discrepancies based on progress rates were flagged for re-annotation. The low error rate reported in Table 9 attests to the high quality of the final annotations. Detailed procedures are outlined in Appendix F.

5 Experiments

We conduct a comprehensive evaluation of popular large language models, including proprietary and open-weight models. Firstly, we report the success rate and progress rate of these agents. Then, we perform detailed analysis of the performance of agents and measure the various abilities of LLM agents, as part of the AgentBoard evaluation automatically supported by our open-source toolkit.

5.1 Evaluation Setup

We implement the agent as described in §3.2. We use a one-shot in-context example in our prompt, in addition to task instructions. For the detailed prompt, please refer to Appendix J. We benchmark a series of strong proprietary and open-weight models. For open-weight models, we assess the corresponding chat version of them. Please refer to Appendix E for details.

5.2 Main Results

Progress Rate is more informative and discriminative than success rate. The success rate and progress rate across various tasks and categories are presented in Table 3. Regarding the overall performance, the progress rate serves as a more effective differentiator between models. For example, `Llama2-13b` and `Mistral-7b` exhibit similarly negligible success rates (2.1% and 3.9%, respectively), but their progress rates differ significantly: 18.9% for `Llama2-13b` and 24.6% for `Mistral-7b`. This disparity suggests that `Mistral-7b` generally outperforms `Llama2-13b`. For models with substantial differences in success rates, such as `Text-Davinci-003` outperforming `Llama2-70b` by 11.7% in success rate, `Text-Davinci-003` leads the progress rate by 13.4% as well, which indicates the consistency in performance disparity between significantly different models. Investigating the agent performance on specific tasks, progress rate is often able to differentiate models that have similar success rates – for instance, on the Embodied AI and Game categories, the success rates of most of the open-weight models are similarly low, while they are able to make meaningfully different progresses. Also, the success rate can be influenced by specific characteristics of agents, for example, an agent like `CodeLlama-34b` often fails to generate the action "finish" when performing tool-using tasks, leading to a higher progress rate and lower success rate compared to `CodeLlama-13b`. In contrast, progress rate is less susceptible to these agent-specific features as it reflects the overall ability of the agent at each step.

Proprietary models outperform the open-weight ones. Unsurprisingly, the performance of proprietary LLMs significantly surpasses that of open-weight models. Notably, `GPT-4` outperforms

Model	Embodied AI			Game		Web		Tool		Avg.
	ALF	SW	BA	JC	PL	WS	WA	TQ	TO	
GPT-4	**82.6**	22.8	**80.3**	**100.0**	**93.0**	**98.3**	**97.6**	**97.5**	**98.5**	**85.6**
Claude2	57.4	11.2	61.6	98.2	71.2	95.9	83.9	93.7	92.3	73.9
GPT-3.5-Turbo	59.2	18.7	62.4	99.8	66.0	90.2	91.3	97.7	91.8	75.2
Deepseek-67b	43.6	12.6	65.4	99.8	62.7	95.4	55.7	93.1	80.5	67.6
Text-Davinci-003	27.3	17.2	15.9	97.9	72.6	97.4	23.7	95.2	82.5	58.9
GPT-3.5-Turbo-16k	57.4	9.2	73.3	**100.0**	77.6	96.6	81.3	98.0	92.2	76.2
Lemur-70b	15.7	**47.8**	44.6	97.7	31.0	84.0	82.8	96.5	88.4	65.4
CodeLlama-34b	8.4	0.4	28.0	97.3	43.6	98.3	96.8	**97.5**	82.8	61.5
CodeLlama-13b	15.8	9.0	34.6	99.7	17.8	78.0	82.1	90.5	79.2	56.3
Llama2-70b	20.8	3.0	42.3	96.7	30.6	59.3	93.6	90.6	70.4	56.4
Mistral-7b	12.1	7.9	33.9	96.2	18.1	83.5	37.5	69.7	35.1	43.8
Vicuna-13b-16k	17.2	24.1	74.5	**100.0**	59.2	94.1	58.7	97.9	92.2	68.7
Llama2-13b	8.9	8.6	37.4	99.5	56.7	73.5	79.2	86.9	74.1	58.3

Table 4: Grounding accuracy (%) on different categories of tasks.

Model	Metric	Embodied AI		Game		Web		Tool		Avg.	
		Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
GPT-4	Progress	90.6	57.4 $_{\downarrow 33.2}$	70.3	62.6 $_{\downarrow 7.7}$	60.8	55.1 $_{\downarrow 5.7}$	89.3	78.5 $_{\downarrow 10.8}$	79.2	62.7 $_{\downarrow 16.5}$
	Success	85.0	24.9 $_{\downarrow 60.1}$	54.2	43.3 $_{\downarrow 10.9}$	32.2	21.8 $_{\downarrow 10.4}$	81.1	52.1 $_{\downarrow 29.0}$	65.6	34.4 $_{\downarrow 31.2}$
GPT-3.5-Turbo	Progress	48.8	31.0 $_{\downarrow 17.8}$	31.4	10.5 $_{\downarrow 20.9}$	49.7	50.4 $_{\uparrow 0.7}$	58.3	48.8 $_{\downarrow 9.5}$	47.2	34.7 $_{\downarrow 12.5}$
	Success	39.9	11.2 $_{\downarrow 28.7}$	9.2	0.0 $_{\downarrow 9.2}$	25.0	14.4 $_{\downarrow 10.6}$	40.6	14.5 $_{\downarrow 26.1}$	29.9	10.1 $_{\downarrow 19.8}$
DeepSeek-67b	Progress	35.8	29.1 $_{\downarrow 6.7}$	28.8	3.8 $_{\downarrow 25.0}$	50.3	45.5 $_{\downarrow 4.8}$	61.0	52.1 $_{\downarrow 8.9}$	43.1	32.2 $_{\downarrow 10.9}$
	Success	26.5	7.9 $_{\downarrow 18.6}$	5.6	2.1 $_{\downarrow 3.5}$	22.0	16.6 $_{\downarrow 5.4}$	40.1	20.1 $_{\downarrow 20.0}$	23.9	11.3 $_{\downarrow 12.6}$
Lemur-70b	Progress	26.0	15.0 $_{\downarrow 11.0}$	16.0	2.1 $_{\downarrow 13.9}$	46.1	39.1 $_{\downarrow 7.0}$	59.5	51.1 $_{\downarrow 8.4}$	35.7	25.5 $_{\downarrow 10.2}$
	Success	9.2	0.3 $_{\downarrow 8.9}$	2.8	0.0 $_{\downarrow 2.8}$	10.7	7.1 $_{\downarrow 3.6}$	31.4	11.8 $_{\downarrow 19.6}$	13.0	4.3 $_{\downarrow 8.7}$
CodeLlama-34b	Progress	13.8	6.6 $_{\downarrow 7.2}$	28.0	3.5 $_{\downarrow 24.5}$	48.3	44.5 $_{\downarrow 3.8}$	66.2	45.7 $_{\downarrow 20.5}$	36.3	23.0 $_{\downarrow 13.3}$
	Success	7.2	0.9 $_{\downarrow 6.3}$	2.8	0.0 $_{\downarrow 2.8}$	19.6	8.7 $_{\downarrow 10.9}$	19.2	3.6 $_{\downarrow 15.6}$	11.6	3.0 $_{\downarrow 8.6}$
Llama2-70b	Progress	13.6	11.0 $_{\downarrow 2.6}$	13.4	1.2 $_{\downarrow 12.2}$	38.4	27.4 $_{\downarrow 11.0}$	45.9	41.8 $_{\downarrow 4.1}$	26.2	19.3 $_{\downarrow 6.9}$
	Success	8.5	1.2 $_{\downarrow 7.3}$	1.4	0.0 $_{\downarrow 1.4}$	12.4	3.9 $_{\downarrow 8.5}$	0.0	0.0 $_{\rightarrow 0.0}$	5.9	1.3 $_{\downarrow 4.6}$

Table 5: Progress Rate and Success Rate for easy and hard cases. All models show distinct drop for hard cases.

other LLMs by a substantial margin with an average progress rate of 70.0%, particularly in the categories of Games and Embodied AI, where the success rates of open-weight LLMs are nearly zero. Surprisingly, `GPT-3.5-Turbo-16k` did not perform better than `GPT-3.5-Turbo`, suggesting that longer context length does not necessarily provide additional benefits in our benchmark.

Strong code skills help agent tasks, with `DeepSeek-67b` leading the open-weight models. In the realm of open-weight LLMs, `DeepSeek-67b` demonstrates relatively superior performance, surpassing `Text-Davinci-003` and is comparable with `GPT-3.5-Turbo`. Code LLMs also show distinct advantage over other open-weight models: `CodeLlama-34b` outperforms `Llama2-70b` by 6.2% in terms of progress rate, despite being significantly smaller. `Lemur-70b`, which is continual pretrained on code, also significantly surpasses `Llama2-70b`. This suggests that incorporating a greater volume of code in training data may enhance performance in agent tasks. However, all open-weight models exhibit weak performance in the Games category, which demands robust planning abilities, as evidenced by nearly zero success rates across the board. In the Tool category, while the success rates are low, the progress rates are comparatively higher, which implies that open-weight models are effective in utilizing tools but less proficient in summarizing information returned by these tools and delivering the final results.

5.3 Analytical Evaluation in AgentBoard

AgentBoard provides various analytical evaluations for in-depth understanding of agents. In this section, we'll use this framework to analyze benchmarked models, with all analyses supported by our toolkit via interactive visualizations on the wandb web panel.

Grounding accuracy. Errors in grounding valid actions demonstrate a key limitation in the model's capacity to follow instructions and produce actions in the correct format, as noted by Zheng et al. (2024). We report the grounding accuracy, the percentage of valid actions, in Table 4. While `Text-Davinci-003` and `DeepSeek-67b` show lower grounding accuracy than `GPT-3.5-Turbo-16K`, they are notably stronger in main results, indicating their advantage in other abilities. `Text-Davinci-003` is notably weak in grounding accuracy, with only 58.9% on average, but its performance in main results is not far away from `GPT-3.5-Turbo`. This suggests that the model has limited capability in utilizing tools but is proficient in planning and other sub-skills. Open-weight models generally have lower grounding accuracy than proprietary

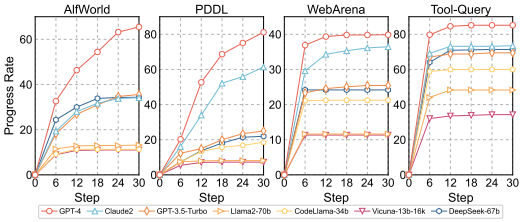


Figure 3: Long-range interaction analysis. Specifically, we report the progress rate w.r.t. step of AlfWorld, PDDL, WebArena and Tool-Query.

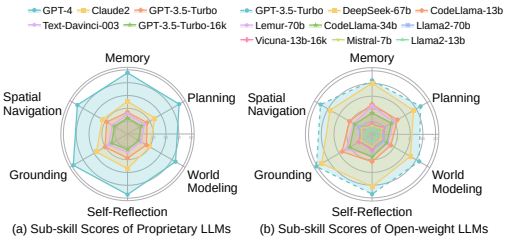


Figure 4: The Sub-skill scores of different LLMs.

ones. Interestingly, `Vicuna-13b-16k`, despite lower main results, achieves a grounding score of 68.7%, comparable to `DeepSeek-67b` and `Claude2`. This underlines why instruction tuning alone couldn't enhance agentic abilities, as found in previous work (Wang et al., 2023b). While tuning improves models' ability to follow instructions, it doesn't necessarily boost overall performance.

Performance breakdown for hard and easy examples. For each task, we divide environments into "hard" or "easy" based on the number of subgoals/conditions to meet, as shown in Table 10. The outcomes are presented in Table 5. Unsurprisingly, all models show a significant performance drop on hard examples, consistent with Dziri et al. (2023)'s findings that even robust LLMs like `GPT-4` struggle with task compositionality. The performance on hard examples, which aligns to real-world settings with multiple subgoals, could be more crucial than average metrics.

Long-Range Interaction. One important characteristic of LLM agents is their ability to engage in multi-round interactions. We analyze how these models proceed across long-range interactions by calculating the progress rate relative to interaction steps, as depicted in Figure 3. We observe that proprietary models like `GPT-4`, `Claude2`, and `GPT-3.5-Turbo` consistently make progress across 30 steps in Alfworld and PDDL tasks, unlike in WebArena and Tool where they quickly peak and then stagnate. This could be because Embodied AI and Games tasks often require more steps, and later stages of Webarena are challenging, causing performance saturation after a few turns.

Sub-skill analysis. We aim to assess LLMs across several facets: *memory* that measures incorporating long-range information in context, *planning* that assesses decomposing complex goals into manageable sub-goals, *world modeling* which tests knowledge necessary for task completion, *self-reflection* that captures the ability to use environmental feedback, *grounding* that focuses on competency in generating valid actions, and *spatial navigation* that represents efficiency in moving to a target location. We develop a sub-skill scoring system based on Table 7, which assigns weighted values to tasks reflecting their sub-skill demands. As depicted in Figure 4, `GPT-4` surpasses other LLMs across all sub-skills, notably outshining proprietary LLMs with lower weights.

Exploration Behavior. Analysis on the exploration behavior are available in Appendix B.

6 Visualization Panel for LLM Agent Analysis: A Case Study

We use Weights&Bias for our visualization panel with task boards for individual task analysis (§3 and §5). As shown in Appendix Figure 6, `GPT-4` is the "Current Run" compared with six other models. The summary board reveals that `GPT-4` outperforms all baselines in overall metrics and all six sub-skills. Its weakest performances are in the Jericho and WebArena tasks, with details on their task boards. `GPT-4` scores lower metrics in Jericho, as it struggles with more complex challenges.

7 Conclusion

In this work, we introduce AgentBoard as a benchmark for evaluating generalist LLM agents. In addition to being a benchmark, AgentBoard offers an open-source, analytical evaluation framework that facilitates easy customization, unified metrics, and comprehensive analysis from diverse aspects, in addition to an interactive visualization web panel. Such analytical evaluation is equipped with an interactive visualization web panel, allowing users to efficiently explore the evaluation and gain a deeper understanding of the agents of interest. Overall, AgentBoard aims to facilitate detailed evaluation and understanding of LLM agents, driving further advancements in the field.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *ArXiv preprint*, abs/2204.01691, 2022. URL https://arxiv.org/abs/2204.01691.

Anthropic. Introducing claude, 2023. URL https://www.anthropic.com/index/introducing-claude.

Harrison Chase. Langchain, 2022. URL https://github.com/langchain-ai/langchain.

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. URL https://openreview.net/forum?id=rJeXCo0cYX.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2023.

DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, A. X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. DeepSeek LLM: Scaling open-source language models with longtermism, 2024.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *ArXiv preprint*, abs/2306.06070, 2023. URL https://arxiv.org/abs/2306.06070.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *ArXiv preprint*, abs/2303.03378, 2023. URL https://arxiv.org/abs/2303.03378.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. Faith and fate: Limits of transformers on compositionality. *ArXiv preprint*, abs/2305.18654, 2023. URL https://arxiv.org/abs/2305.18654.

Yu Gu, Xiang Deng, and Yu Su. Don't generate, discriminate: A proposal for grounding language models to real-world environments. *ArXiv preprint*, abs/2212.09736, 2022. URL https://arxiv.org/abs/2212.09736.

Matthew J. Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7903–7910. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6297.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *ArXiv preprint*, abs/2103.03874, 2021. URL https://arxiv.org/abs/2103.03874.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *ArXiv preprint*, abs/2310.06825, 2023. URL https://arxiv.org/abs/2310.06825.

Megan Kinniment, Lucas Jun Koba Sato, Haoxing Du, Brian Goodrich, Max Hasin, Lawrence Chan, Luke Harold Miles, Tao R Lin, Hjalmar Wijk, Joel Burget, et al. Evaluating language-model agents on realistic autonomous tasks. *ArXiv preprint*, abs/2312.11671, 2023. URL https://arxiv.org/abs/2312.11671.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Tamera Lanham, Anna Chen, Ansh Radhakrishnan, Benoit Steiner, Carson Denison, Danny Hernandez, Dustin Li, Esin Durmus, Evan Hubinger, Jackson Kernion, et al. Measuring faithfulness in chain-of-thought reasoning. *ArXiv preprint*, abs/2307.13702, 2023. URL https://arxiv.org/abs/2307.13702.

Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

Minghao Li, Feifan Song, Bowen Yu, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A benchmark for tool-augmented llms. *ArXiv preprint*, abs/2304.08244, 2023. URL https://arxiv.org/abs/2304.08244.

Yen-Ting Lin and Yun-Nung Chen. Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models. *ArXiv preprint*, abs/2305.13711, 2023. URL https://arxiv.org/abs/2305.13711.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *ArXiv preprint*, abs/2308.03688, 2023a. URL https://arxiv.org/abs/2308.03688.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *ArXiv preprint*, abs/2308.05960, 2023b. URL https://arxiv.org/abs/2308.05960.

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

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. *ArXiv preprint*, abs/2311.12983, 2023. URL https://arxiv.org/abs/2311.12983.

OpenAI. Introducing chatgpt, 2022. URL https://openai.com/blog/chatgpt.

OpenAI. Gpt-4 technical report. *arXiv*, pp. 2303–08774, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Aloïs Pourchot and Olivier Sigaud. CEM-RL: combining evolutionary and gradient-based methods for policy search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=BkeU5j0ctQ.

Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *ArXiv preprint*, abs/2304.08354, 2023a. URL https://arxiv.org/abs/2304.08354.

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

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *ArXiv preprint*, abs/2205.06175, 2022. URL https://arxiv.org/abs/2205.06175.

Toran Bruce Richards. Significant-gravitas/autogpt: An experimental open-source attempt to make gpt-4 fully autonomous., 2023. URL https://github.com/Significant-Gravitas/AutoGPT.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *ArXiv preprint*, abs/2308.12950, 2023. URL https://arxiv.org/abs/2308.12950.

Stuart Russell and Peter Norvig. Ai a modern approach. *Learning*, 2(3):4, 2005.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 2017a. URL http://proceedings.mlr.press/v70/shi17a.html.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 2017b. URL http://proceedings.mlr.press/v70/shi17a.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Tom Silver and Rohan Chitnis. Pddlgym: Gym environments from pddl problems. *ArXiv preprint*, abs/2002.06432, 2020. URL https://arxiv.org/abs/2002.06432.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971, 2023. URL https://arxiv.org/abs/2302.13971.

Mauro Vallati, Lukas Chrpa, Marek Grześ, Thomas Leo McCluskey, Mark Roberts, Scott Sanner, et al. The 2014 international planning competition: Progress and trends. *Ai Magazine*, 36(3): 90–98, 2015.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *ArXiv preprint*, abs/2305.16291, 2023a. URL https://arxiv.org/abs/2305.16291.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.emnlp-main.775.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *ArXiv preprint*, abs/2309.10691, 2023b. URL https://arxiv.org/abs/2309.10691.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *ArXiv preprint*, abs/2206.07682, 2022. URL https://arxiv.org/abs/2206.07682.

Yue Wu, Xuan Tang, Tom M Mitchell, and Yuanzhi Li. Smartplay: A benchmark for llms as intelligent agents. *ArXiv preprint*, abs/2310.01557, 2023. URL https://arxiv.org/abs/2310.01557.

Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. Openagents: An open platform for language agents in the wild. *ArXiv preprint*, abs/2310.10634, 2023. URL https://arxiv.org/abs/2310.10634.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *ArXiv preprint*, abs/2305.16504, 2023a. URL https://arxiv.org/abs/2305.16504.

Yiheng Xu, Hongjin Su, Chen Xing, Boyu Mi, Qian Liu, Weijia Shi, Binyuan Hui, Fan Zhou, Yitao Liu, Tianbao Xie, et al. Lemur: Harmonizing natural language and code for language agents. *ArXiv preprint*, abs/2310.06830, 2023b. URL https://arxiv.org/abs/2310.06830.

Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *ArXiv preprint*, abs/2312.13771, 2023. URL https://arxiv.org/abs/2312.13771.

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

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*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Qi Zhang, Tao Gui, et al. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios. *ArXiv preprint*, abs/2401.00741, 2024. URL https://arxiv.org/abs/2401.00741.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. *ArXiv preprint*, abs/2401.01614, 2024. URL https://arxiv.org/abs/2401.01614.

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

APPENDIX

A AUTHOR CONTRIBUTIONS

Code Implementation Chang Ma implemented the code base for AgentBoard framework. The code for different tasks is implemented by respective person in charge: Junlei Zhang (Alfworld, Scienceworld), Chang Ma (BabyAI, Jericho and PDDL), Zhihao Zhu (WebShop and WebArena), Cheng Yang (Tool-Query and Tool-Operation). The website was implemented by Zhihao Zhu and the visualization panel was implemented by Chang Ma. The code of Alfworld, ScienceWorld, PDDLGym, WebShop, WebArena and Mint sped up the implementation.

Task Unification Junlei Zhang, Chang Ma, Cheng Yang, Zhihao Zhu implemented the tasks into environmental interaction format, provided labels for respective tasks, adapted the metrics, and verified the performances. Chang Ma, Junlei Zhang, Junxian He additionally verified the tasks to be unified.

Paper writing Chang Ma and Junxian He finished introduction and methodology sections of the paper. Junlei Zhang and Chang Ma wrote the experiments section. Cheng Yang provided all the visualizations shown in the paper. Cheng Yang, Zhihao Zhu added results and analysis for their corresponding parts. Junxian He carefully reviewed and revised the paper and gave feedback for multiple rounds. Other authors help proofread and provide feedbacks.

Experiments Chang Ma and Junlei Zhang co-lead the evaluation of the models. Zhihao Zhu conducted all the evaluations on web tasks. Cheng Yang conducted evaluation on tool tasks for several models and conducted experiments on analysis and visualization.

Data Collection and Human Annotation Data for task examples and progress rate annotation for each task is collected and annotated with one person in charge, and verfied by at least two others: Junlei Zhang led data collection and annotation for ScienceWorld, Alfworld; Chang Ma led data collection and annotation for BabyAI, Jericho and PDDL; Zhihao Zhu led data collection and annotation for WebShop, WebArena and Sheet task in Tool-Operation; Cheng Yang led data collection and annotation for Tool-Query and Tool-Operation. Junlei Zhang led data validation for BabyAI and Tool-Query; Chang Ma led data validation for ScienceWorld and WebShop; Zhihao Zhu led data validation for Jericho, PDDL and Tool-Operation; Cheng Yang led data validation for ScienceWorld and WebArena.

Junxian He is the main advisor of this project.

B EXPLORATION BEHAVIOR ANALYSIS

We examine the exploration behavior of models in various environments, as illustrated in Table 6. The ability of agents to explore plays a significant role in their performance in partially-observable environments, as diverse exploration trajectories enable agents to acquire all the necessary information. We compare the number of locations explored by models, including rooms in BabyAI, containers in AlfWorld, and places in Jericho. This metric reflects the models' exploration capabilities. Most models are unable to explore the minimum number of locations necessary to complete the goal. `GPT-3.5-Turbo` demonstrates performance comparable to `GPT-4` in this score. Among the open-weight models, `Llama2-70b` and `CodeLlama-34b` show similar performance and both outperform `Vicuna-13b-16k`, consistent with progress rate and success rate. Detailed analysis on exploration behavior is not currently implemented in our AGENTBOARD framework since it is feasible only for some environments.

C SUB-SKILL TABLE

Table 7 shows the criteria for sub-skill scoring and sub-skill scores for each task in AGENTBOARD.

Tasks	Minimum	GPT-4	GPT-3.5-Turbo	Llama2-70b	CodeLlama-34b	Vicuna-13b-16k
Babyai - UnlocktoUnlock	3	1	1.25	1	1.25	1
Babyai - FindObjs5	3	2	3.5	2	2	1
Babyai - Keycorridor	3	3	2.5	1.5	1.75	1.25
Alfworld	3	5.625	5.125	1.75	0.125	1
Jericho - Zork1	5	5	5	1	5	1
Jericho - Zork2	6	6	2	1	3	3
Jericho - Zork3	11	6	4	3	2	1

Table 6: Comparison of the number of locations(room in babyai, containers in alfworld, and places in Jericho) explored by models. The minimum column states the least number of locations need to explore on average in order to finish the tasks.

	AlfWorld	ScienceWorld	BabyAI	Jericho	PDDL	WebShop	WebArena	Tool-Query	Tool-Operation
Memory 1. Could finish tasks within 2k tokens 2. Could finish task within 4k tokens 3. Otherwise	1	2	1	1	2	1	3	2	3
Planning 1. ≤ 3 subgoals on average 2. ≤ 5 subgoals on average 3. Otherwise	1	2	2	3	3	2	3	2	2
World Modeling 1. Requires no additional knowledge other than instruction 2. Requires knowledge of the environment from exploration 3. Requires commonsense knowledge in addition to knowledge from environment	3	3	2	3	1	1	3	1	1
Self-Reflection 1. Detailed feedback and error message with instruction for the next step. 2. Not very detailed feedback and error message 3. No error message, e.g. "no change in state"	3	2	2	1	3	2	2	1	1
Grounding 1. No specific action format is required, could recognize similar actions 2. Action format is required 3. Action format hard to follow	2	3	2	1	3	3	3	3	3
Spatial Navigation 0. No spatial navigation 1. 2D navigation	1	1	1	1	0	0	1	0	0

Table 7: The sub-skill scores associated with each task in AGENTBOARD.

D VISUALIZATION PANEL

The visualization panel supported by AGENTBOARD is shown in Figure 5. We provide a detailed explanation of panel features and usage tutorial in WandB blog[3].

We provide a case study in Figure 6 to show example usage of AGENTBOARD.

E DETAILS OF EVALUATED LLMS

E.1 EVALUATION SETUP

We use greedy decoding strategy and set temperature to zero for better replicacy, and all LLMs are implemented with vLLM (Kwon et al., 2023) architecture, which has 10× acceleration over huggingface inference. During prompting, we keep the most recent interaction histories within the maximum context length of the model. For models with different versions of checkpoints, we choose the version with best instruction following ability, with chat SFT and alignment. The following are the specific models we assess in the experiments.

E.2 DETAILS OF MODELS

We list our evaluated models in Table 8.

[3]This blog will be released after the review period

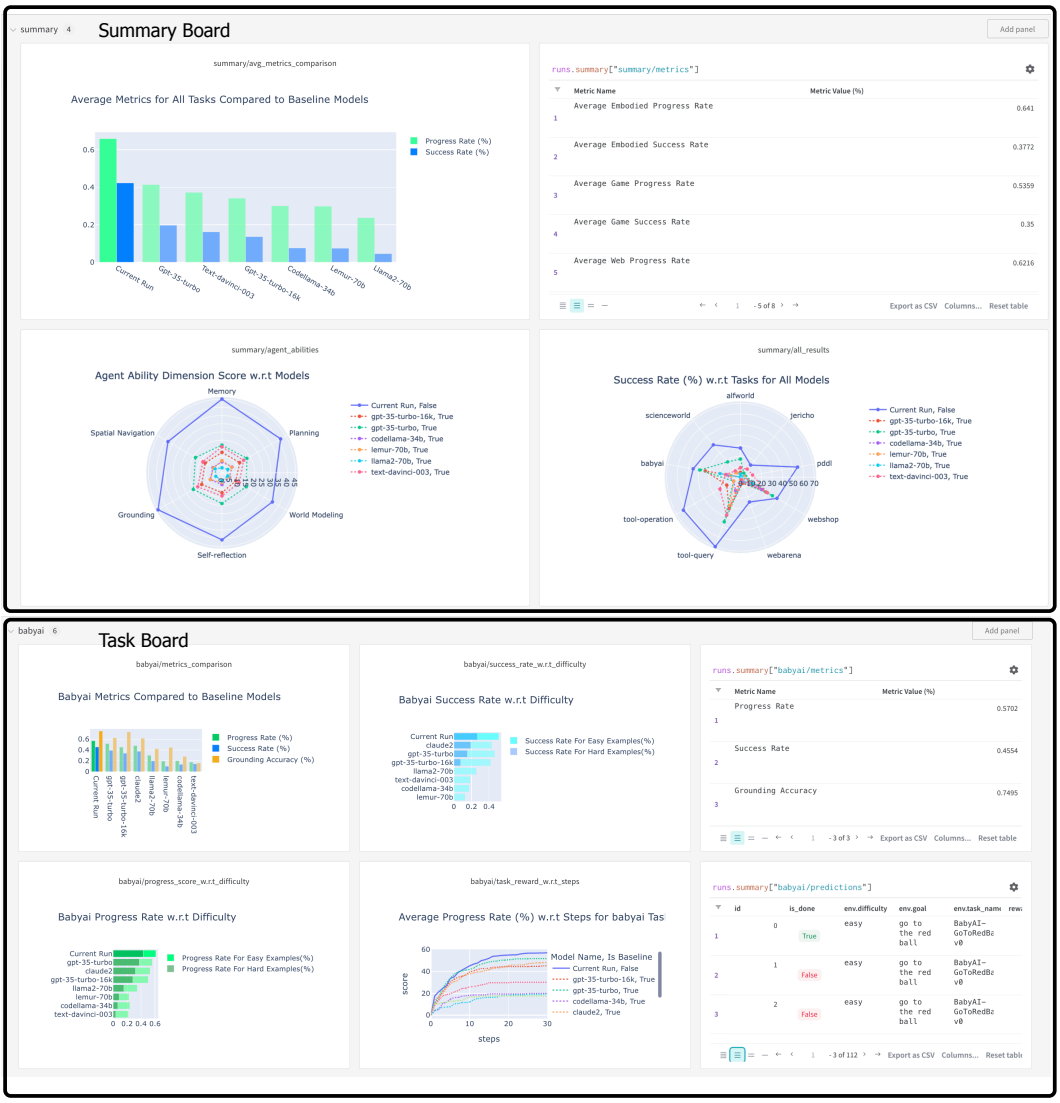


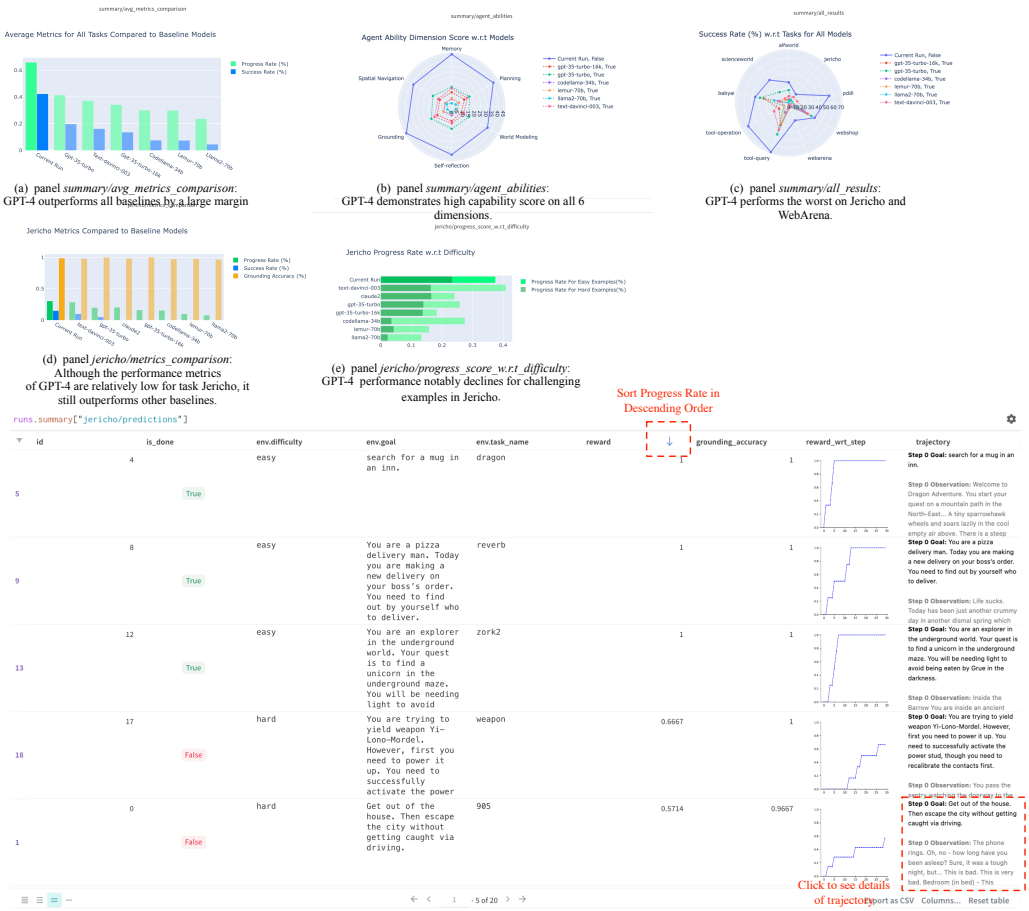
Figure 5: Visualization Panel based on WandB, composed of a summary board with all metrics and a task board for each task.

Model Name	Model Code/API
GPT-4 (OpenAI, 2023)	Azure api: `gpt-4` (version: 2023-05-15)
GPT-3.5-Turbo (OpenAI, 2022)	Azure api: `gpt-35-turbo`
GPT-3.5-Turbo-16k (OpenAI, 2022)	Azure api:`gpt-35-turbo-16k`
Claude2 (Anthropic, 2023)	Anthropic api: `claude-2` (version: 2023-06-01)
Text-Davinci-003 (Ouyang et al., 2022)	Azure api: `text-davinci-003`
Mistral-7b (Jiang et al., 2023)	`mistralai/Mistral-7B-v0.1`
CodeLlama-13b (Roziere et al., 2023)	`codellama/CodeLlama-13b-Instruct-hf`
CodeLlama-34b (Roziere et al., 2023)	`codellama/CodeLlama-34b-Instruct-hf`
Llama2-13b (Touvron et al., 2023)	`meta-llama/CodeLlama13b-chat-hf`
Llama2-70b (Touvron et al., 2023)	`meta-llama/Llama-2-70b-chat-hf`
Vicuna-13b-16k (Chiang et al., 2023)	`lmsys/vicuna-13b-v1.5-16k`
Lemur-70b (Xu et al., 2023b)	`OpenLemur/lemur-70b-chat-v1`
DeepSeek-67b (DeepSeek-AI et al., 2024)	`deepseek-ai/deepseek-llm-67b-chat`

Table 8: Model code/API of our evaluated models.

F DATA QUALITY CONTROL

To ensure the quality of labeled sub-goals, we conducted three rounds of data verification for each labeled sub-goal. We developed an interactive interface through which inspectors complete tasks and



(a) panel *summary/avg_metrics_comparison*: GPT-4 outperforms all baselines by a large margin

(b) panel *summary/agent_abilities*: GPT-4 demonstrates high capability score on all 6 dimensions.

(c) panel *summary/all_results*: GPT-4 performs the worst on Jericho and WebArena.

(d) panel *jericho/metrics_comparison*: Although the performance metrics of GPT-4 are relatively low for task Jericho, it still outperforms other baselines.

(e) panel *jericho/progress_score_w.r.t_difficulty*: GPT-4 performance notably declines for challenging examples in Jericho.

(f) panel *jericho/predictions*: After sorting progress rate in descending order, we can see that the best performing problems are all easy ones. We can take a closed look at "trajectory" to understand why hard problems could not be solved by GPT-4.

Figure 6: A case study for GPT-4 based on Panels from AGENTBOARD.

Task	Proportion of environments with annotation errors
AlfWorld	10.0%
ScienceWorld	0%
Babyai	4.2%
Jericho	25%
PDDL	5%
WebShop	0%
WebArena	0%
Tool-Query	0%
Tool-Operation	0%

Table 9: The proportion of environments with annotation errors in the second round of data checking. Environments identified with errors are subsequently analyzed to determine the underlying causes, and any environments exhibiting similar errors are amended collectively.

observe the reward scores obtained at each step. If the inspector deems the reward score assigned during interaction with an environment to be unreasonable, additional annotators will engage in a discussion to determine if modifications to the labeled sub-goals are necessary.

The first round of verification is a self-check. Each annotator is required to carefully review the labeled tasks in every environment they are responsible for. The second round involves a sampled inspection by two annotators for each task. They examine a sample of 5-10 items from different sub-tasks within the task and document the proportion of issues identified, as presented in Table 9.

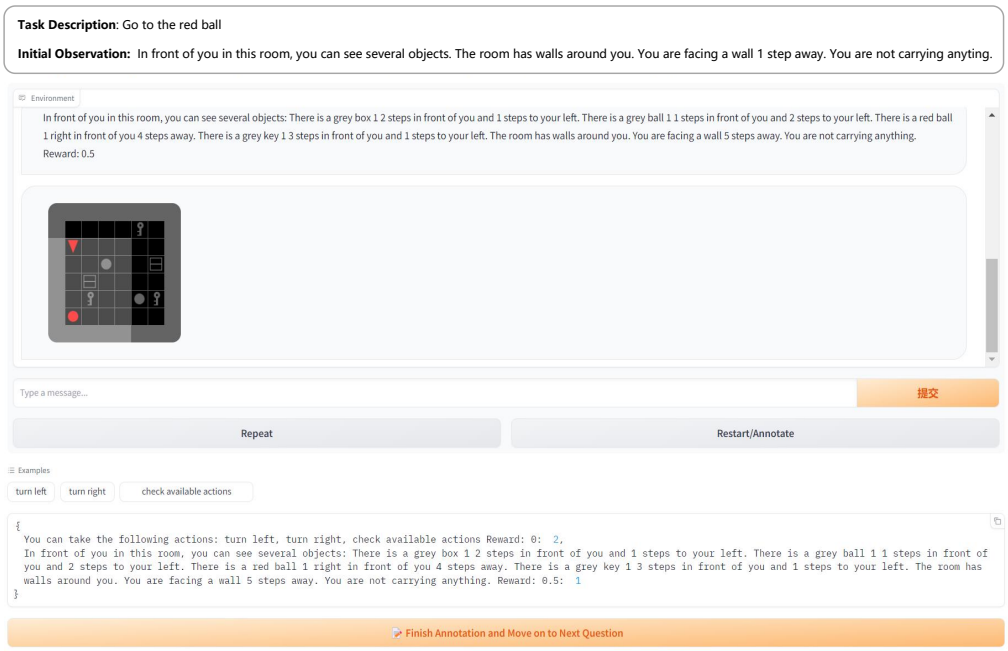


Figure 7: An illustration of our sub-goal checking interface. We develop an interactive interface for annotators to checking sub-goals. Firstly, annotators play and pass the game with the interface. The reward score for each step will be given based on the labeled score. If the annotators are dissatisfied with the reward, annotators will record them and the corresponded environment will be discussed by more annotators and annotated again."

Table 10: Statistics of 9 environments in AGENTBOARD. "subgoal" and "match" means 2 different implementations of progress rate r^{subgoal} and r^{match} respectively. †Note that Sheet Environments in Tool-Operation are evaluated with r^{match}, while other sub-tasks are evaluated with r^{subgoal}. ‡For tasks without subgoal label, we state the average number of constraints to satisfy in the goal state, which is essentially the complexity of the problems. For context length, we report the number of tokens generated with Llama2 tokenizer. † We divide problems into hard/easy based on the number of subgoals – problems with a larger number of subgoals than cutoff are viewed as hard.

	Embodied AI			Game		Web		Tool	
	ALF	**SW**	**BA**	**JC**	**PL**	**WS**	**WA**	**TQ**	**TO**
# Environment	134	90	112	20	60	251	245	60	40
# Turns	6	15	10	20	20	3	25	5	6
Action Space	13	21	8	150	8	2	12	15	16
# Avg. Subgoals‡	3	5	4	6	6	4	6	5	5
Hard/Easy Cutoff†	3	3	3	4	6	1	4	4	4
Context Length	900	2800	1800	1500	2700	1200	15000	2100	4300
Progress Rate	subgoal	subgoal	subgoal	subgoal	match	match	match	subgoal	subgoal/match †

The third round is conducted by an annotator who is well-acquainted with the various tasks, who then performs a sampled review of all tasks.

G DETAILS OF ENVIRONMENTS

G.1 DETAILS OF EMBODIED ENVIRONMENTS

AlfWorld (ALF) (Shridhar et al., 2021) are Household tasks that require models to explore rooms and use commonsense reasoning to perform tasks, Within AGENTBOARD, we evaluate a

model's ability to perform tasks in physical household settings, such as "put a pencil on the desk". AlfWorld is categorized into six types, comprising a total of 134 environments.

ScienceWorld (SW) (Wang et al., 2022) is a complex interactive text environment that poses a significant challenge to agents' scientific commonsense. This environment requires agents to navigate through 8 distinct functional rooms (e.g., workshop, kitchen) and utilize the tools to complete tasks such as "measure the melting point of the orange juice". To address these issues, we re-annotate subgoals to calculate $r_t^{subgoal}$, where Specifically, we incorporate necessary observations as part of the subgoals. the rewards for these subgoals are uniform and distributed evenly throughout the task. To ensure that our annotated subgoals are necessary for achieving final goals, we restrict the use of tools and designated task completion rooms in the task descriptions. We show more details of we annotated subgoals in the Appendix H.2

BabyAI (BA) (Chevalier-Boisvert et al., 2019) is an interactive environment where agents navigate and manipulate objects in a 20x20 grid space. The agent can only see objects within a limited sight and cannot perceive objects in remote rooms. The original implementation represents observations as images and only allows for tensor-based low-level actions such as "0: move left". "1: move right", and "2: move forward". To enable text-based input and output for LLM agents, We modified it by mapping the original actions to a textual action space and providing textual descriptions of visual observations, as shown in Table 2. For each step, the environment returns a text description of the current observation, such as "There is a red ball 1 step to your right and 1 step ahead of you. There is a wall 2 steps ahead." We also introduced high-level actions, such as "go to red ball 1" and "toggle and go through green locked door 1", to expand the action space and enrich the semantic complexity of the environment. Additionally, we implemented a new subgoal-based progress rate for the environments to increase the density of rewards compared to the original reward scores. Unlike the previous reward score in BabyAI which awards a point only after a new object is found or pickup, requiring many steps to see progress in reward score, our new approach increases density of the rewards, requiring fewer steps to achieve them. We re-annotated subgoals and calculate with the equation of r_t^{subgoal}. Subgoals are re-annotated to update the progress rate whenever the agent makes progress, such as navigating to another room, finding a red ball, and picking it up in the problem "pickup a red ball".

G.2 Details of Game Environments

Evaluating LLM agents as strategic game playing agents demands strong planning ability of agents. We choose three tasks that are all demanding in planning and making strategies.

Jericho (JC) (Hausknecht et al., 2020) is a collection of text-based game environments that evaluate agents to perform adventures in fictional worlds. This task is unique in that it requires strong world modeling ability as agents could only gain information about the magic world through exploration and interaction. For example, for the task that requires the agent to perform actions with magic, it cannot reason with pre-trained commonsense knowledge and must perform exploration to understand the rules of the magic world. The original games are quite long (need 50-300 steps to finish), which is not suitable for LLM agents with fixed context length. To solve this issue, we rewrite the goal of each adventure to restrict the games to be finished within 15 subgoals. For example, *zork1* game requires the player to enter a dungeon and explore the dungeon to find a bar. We rewrite the goal as "You need to find your way into a secret passage where the entrance is in the living room of the house." and the agent only needs to find the entrance to the dungeon, which can be finished in 8 steps. We use the r_t^{subgoal} as progress rate metrics, and we meticulously annotate the subgoals for each problem. Each subgoal characterize that the agent has solved a small problem, e.g. "find the entrance to the house" $\rightarrow$ "enter the house" $\rightarrow$ "find the living room" $\rightarrow$ "discover a trap door" $\rightarrow$ "find the entrance to dungeon".

PDDL (PL) (Vallati et al., 2015), short for Planning Domain Definition Language, is a set of strategic games defined with PDDL symbolic language. We selected 4 representative game domains, *Gripper, Barman, Blocksworld, Tyreworld* to benchmark LLM agents in diverse scenarios, where the agent needs to move balls across rooms, make cocktails, rearrange blocks and pump up and install new tyres to cars. This task is difficult as it requires multiple rounds of planned actions to

finish a single subgoal and agents need to plan strategically to avoid repetitive steps. For example, in *Barman*, the player is given a menu, and is required to make a few cocktails with a few containers and ingredients. The agent could use a strategy of trying to use different containers each time to avoid repetitive cleaning and save steps. While the commonly-used environment implementation (Silver & Chitnis, 2020) requires the agent to interact with an environment with PDDL expressions, e.g. `clean-shaker(hand1, hand2, shaker)` and provides observations as set of predicates `ontable(shaker1) ∧ empty(shaker1)`. we write parser rules to offer a text-based observation to agents that allows LLMs to interact with natural language to be consistent with other tasks. e.g."Shaker1 is on the table. Shaker1 is empty" and enable the agents to interact with the environment with simple text commands, e.g. "clean-shaker shaker1 with hand1 while hand2 is empty." We curate 10-20 problems for each of the four domains by ourselves, ensuring the problems are multi-round and diverse. We use the r_t^{match} as progress rate metric, where the matching score compares the similarity between the properties of current state and the goal state. e.g. for the goal state "Block a is on block b. Block b is on the table", if at current state "Block a is on the table. Block b is on the table", then the matching score is 0.5. The agent will receive a 100% progress rate only if all conditions of the goal state are satisfied.

G.3 Details of Web-based Environments

Evaluating LLM's capability as a generalist agent in web-based scenarios has become pivotal (Shi et al., 2017a; Deng et al., 2023). Web agent is expected to navigate the network efficiently and perform diverse tasks amidst highly dynamic, intricate, and multi-turn interactions. Based on the task categorization, we've pinpointed two tasks of high recognition and quality: the specific network task, WebShop (Yao et al., 2022), and the general network task, WebArena (Zhou et al., 2023). The latter permits unrestricted access to any supported webpage.

WebShop (WS) (Yao et al., 2022) is a network-based simulation environment for e-commerce experiences, featuring a website with 1.18 million actual products, each with distinct labels and attributes. In this environment, the agent is allowed to interact with the system through 'search[QUERY]' or 'click[ELEMENT]' actions to purchase products matching the instructions. This process necessitates that the model possesses reasoning and grounding abilities. Based on the original implementation method (Yao et al., 2022; Shinn et al., 2023), we have improved the error feedback, including refining the observation for exceeding page limits and interacting with wrong objects. These enhancements contribute to the effective operation of the entire environment and the rationality of multi-step reasoning processes. As there are no sub-goals in the environment, to obtain a continuous progress rate, we expanded the calculation rules from (Yao et al., 2022), calculating the score at different web pages (stages). To measure the distance of the current state to the final goal as the progress rate, we expanded the product scoring rules from Yao et al. (2022) to derive the score at different web pages. Please refer to Appendix H.6 for details.

WebArena (WA) (Zhou et al., 2023) is a real web environment containing four applications: online shopping, discussion forums, collaborative development, and business content management. It supports 11 different web browsing actions. such as click (element), new tab, goto (URL), etc., and offers additional tools like maps and wikis. The observation space consists of structured web content (the accessibility tree [4]). Completing tasks in this highly realistic environment requires the agent to possess strong memory, high-level planning, common sense, and reasoning abilities. Compared to other datasets (Deng et al., 2023; Shi et al., 2017b), WebArena offers multi-round and continuous web browsing interaction simulation. We filtered 245 instances from the original dataset for two main sub-tasks: Site Navigation and Contact & Config, each annotated with the target URLs or required content. To obtain the progress rate, we revised the existing method for calculating the final score (Zhou et al., 2023) and continuously computed the progress rate at each step, fusing the URL matching score with the content matching score, derived from the current URL and target URL, with the content matching score calculated based on the detected required content, as detailed in Appendix H.7.

[4] https://developer.mozilla.org/en-US/docs/Glossary/Accessibility_tree

G.4 DETAILS OF TOOL ENVIRONMENTS

In AGENTBOARD, a tool contains a variety of functions, accessed by agents via function calling. These functions are the actions that LLM agents can take in tool environments. Drawing upon open datasets and APIs, we have developed a suite of five distinct tools, each encapsulated in its own environment. Tool Environments are categorized into two groups: Tool-Query Environments and Tool-Operation Environments, representing two general usage scenarios. Tool-Query Environments include Weather Environment, Movie Environment and Academia Environment. Tool-Operation Environments include Todo Environment and Sheet Environment.

G.4.1 TOOL-QUERY ENVIRONMENTS

Weather Environment Weather Environment enables LLM agents to use the weather tool to retrieve past, present and future weather data, encompassing temperature, precipitation and air quality across various locales. We use Python codes to integrate Open-Meteo API[5], implement the requisite functions and subsequently develop a weather tool.

Movie Environments Movie Environment grants LLM agents to use the movie tool to access cinematic data, encompassing film details, personnel and production companies. We incorporate the API and data from The Movie Database[6], implement the necessary functions, and thus establish the movie tool.

Academia Environment Academia Environment equips LLM agents the academia tool to query information related to computer science research, including academic papers and author information. In its development, we harness data from the Citation Network Dataset[7], craft the relevant functions, and subsequently construct the academia tool.

G.4.2 TOOL-OPERATION ENVIRONMENTS

Todo Environment Todo Environment facilitates LLM agents in querying and amending personal agenda data through the todo tool. We implement the todo tool based on the Todoist API[8].

Sheet Environment Sheet Environment allows LLM agents to use the sheet tool to access and modify spreadsheet data. We build our sheet tool upon the Google Sheets API[9].

H DETAILS OF PROGRESS RATE METRICS

H.1 ALFWORLD

We identify and annotate the necessary subgoals using regular expressions. For instance, for the task "put a pencil on the desk", we annotate one necessary observation as "You pick up the pencil +". This expression would match observations like "You pick up the pencil 1". When the goal of an environment is achieved, the environment emits a task success flag. Specifically, for each environment, we labeled N-1 necessary subgoals as N-1 subgoals. The final success flag combined with the N-1 annotated subgoals constitutes the set of N subgoals.

H.2 SCIENCEWORLD

We compare our modified task descriptions and subgoals with the original ones in Table 11. In the original scheme, subgoals are categorized as "sequential subgoals" and "unordered and optional subgoals". For the former, achieving sequential subgoals alone is sufficient to receive full rewards (100 points). However, under the "unordered and optional subgoals", each completed task is only

[5] https://open-meteo.com/
[6] https://www.themoviedb.org/
[7] https://www.aminer.org/citation
[8] https://todoist.com/
[9] https://www.google.com/sheets/about/

	Original Labels	Ours
Task Description	Your task is to freeze orange juice. First, focus on the substance. Then, take actions that will cause it to change its state of matter.	Your task is to freeze orange juice **in the kitchen**. **The objects you can use are a metal pot, a freezer, a thermometer, and a fridge.** Take actions that will cause it to change its state of matter to a solid state. **Finally, examine its altered state. You should wait and monitor the temperature of the water until it changes its state.**
Subgoals	**Sequential Subgoals:** 1. focus on substance 2. substance is in a liquid state 3. substance is in a solid state **Unordered and Optional Subgoals:** 1. be in same location as orange juice 2. have substance alone in a single container 3. have object in cooler (fridge) 4. have object in cooler (freezer) 5. cool object by at least 5C	**Necessary Observations:** 1. You move to the kitchen. 2. The freezer is now open. 3. The fridge is now open. 4. the thermometer measures a temperature of (-?[0-9] | – ?[1-9][0-9] | – ?[1-9][0-9]2) degrees celsius 5. solid orange juice

Table 11: Comparison between the original task description and subgoals of ScienceWorld and our labeled subgoals(Best viewed in color).

awarded low point (e.g. 1 point). These tasks are also important and necessary for accomplishing the given task. For instance, the "optional subgoals" outlined in Table 11, such as "be in the same location as the orange juice" and "have the substance alone in a single container" are necessary for the task and can help to evaluate a model's navigation and common sense abilities. It is inappropriate to assign such tasks a low score. Furthermore, the uneven distribution of "Sequential Subgoals" throughout the entire task process can lead to a disproportionately low score, which does not accurately reflect the model's progress. For example, if the model fails to complete the initial subgoals within the "Sequential Subgoals" category, which could be considerably distant from the start state, it can only achieve a very low score. This scoring method does not align with our motivation, which is to ensure that the progress rate adequately reflects the model's performance. Therefore, we have re-annotated the subgoals. Specifically, we label necessary observations as part of the subgoals.

In the original task descriptions, the possibility of multiple necessary tools being present in multiple rooms (e.g., a thermometer) creates multiple viable gold paths for task completion. Consequently, a single state may exhibit different progress levels across various gold paths. This disparity makes it challenging to assign a definitive progress rate to any given state. Therefore, in our task descriptions, we have restricted the locations and tools used for tasks to ensure the uniqueness of our goal paths and the necessity of observations. For the necessary observations, our initial observation is more close to the initial state and but still challenging.

we design an interactive UI framework (Figure 7). We ask one graduate student to interact with the environment and record the necessary observations to achieve the given goal. As a result, we revise the task descriptions to include sufficient information for achieving the subgoals and to ensure the gold path is unique. .

H.3 BABYAI

The origin implementation of babyai provides a reward score. Different from the original reward, our progress rate is more dense and the agent does not need to accomplish many steps before getting a increase in score. Here we compare the difference between our progress rate and the original reward score, as shown in Table 12. We can see from this case that our progress rate better measures intermediate progress for agents.

The progress rate is labelled via an interactive UI framework (Figure 7). A graduate student interact with the environments and record the observations corresponding to subgoals needed to finish the problem.

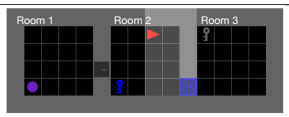
Problem: Unlock to Unlock	Steps with Score Increase (Original)	Steps with Score Increase (Ours)
	1. Pickup purple ball	1. Pickup blue key; 2.Enter *room 3*; 3. Pickup grey key; 4. Enter *room 2*; 5. Enter *room 1*; 6. Pickup purple ball

Table 12: Comparison between our progress rate for BabyAI and original reward score.

H.4 JERICHO

The original Jericho games are free-exploration text-based games, where the player is not given a tangent goal but allowed to explore around the environment as adventureres. For uniformity with other tasks, we first write a new goal for each problem, and we carefully select the goal so that the game could be accomplished within 15 subgoals. In contrast, the original environments requires around 50-300 interactions to get the maximum rewards. The annotation of goal and subgoals are also performed by a graduate student in the interactive UI framework.

H.5 PDDL

In the PDDL environment, each state is discribed by a conjunction of properties $p_1 \wedge p_2, \ldots, \wedge p_m$, each property is a simple predicate describing the property of an object, e.g. "Block a is on the table". Given the goal state $g_1 \wedge g_2, \ldots, \wedge g_n$ and any state $p_1 \wedge p_2, \ldots, \wedge p_m$, the matching score formula is defined as:

$$f = \frac{|\mathcal{G} \cap \mathcal{P}|}{|\mathcal{G}|}, \mathcal{G} = \{g_1, g_2, \ldots, g_n\}, \mathcal{P} = \{p_1, p_2, \ldots, p_m\} \tag{3}$$

The matching score is 1 if and only if the properties of goal state is satisfied in current state.

H.6 WEBSHOP

In the webshop environment, we expanded the product scoring rules from (Yao et al., 2022) to derive the score at different web pages. We can calculate the score of any product (the distance from the target product) using the original scoring formula as follows:

$$f = f_{\text{type}} \cdot \frac{|\mathcal{U}_{\text{att}} \cap \mathcal{Y}_{\text{att}}| + |\mathcal{U}_{\text{opt}} \cap \mathcal{Y}_{\text{opt}}| + \mathbf{1}[\text{y}_{\text{price}} \leq \text{u}_{\text{price}}]}{|\mathcal{U}_{\text{att}}| + |\mathcal{U}_{\text{opt}}| + 1}, \tag{4}$$

Each natural language instruction, denoted as $\text{u} \in \mathcal{U}$, encompasses a non-empty set of attributes, $\mathcal{U}_{\text{att}}$, a set of options, $\mathcal{U}_{\text{opt}}$, and a specified price, u_{price}. Meanwhile, $\mathcal{Y}$ represents the product chosen by the agent. The function $f_{\text{type}} =$ TextMatch $(\bar{y}, \bar{y}^*)$ is based on text matching heuristics to assign low reward when y and y^* have similar attributes and options but are obviously different types of products.

Typically, completing a web shopping task involves three continuous stages: search, product selection, and finalizing the product style before placing an order. Therefore, to measure the distance between the current state and the target state, we primarily calculate scores for three pages (states): search result page, product description page, and order confirmation page. On the search result page, we calculate the score of each product on the page and take the highest score as the score for this page. On the product description page, we compute the highest score for the product under various options as the page score. On the order confirmation page, the score of the finally selected product is considered as the score for that page. In our method, the progress rate is the average of the scores from these three pages

H.7 WEBARENA

In our method, we effectively utilize the annotation data, treating URLs as indicators of the web browsing trajectory and required contents as integral scoring points.The progress rate is formulated

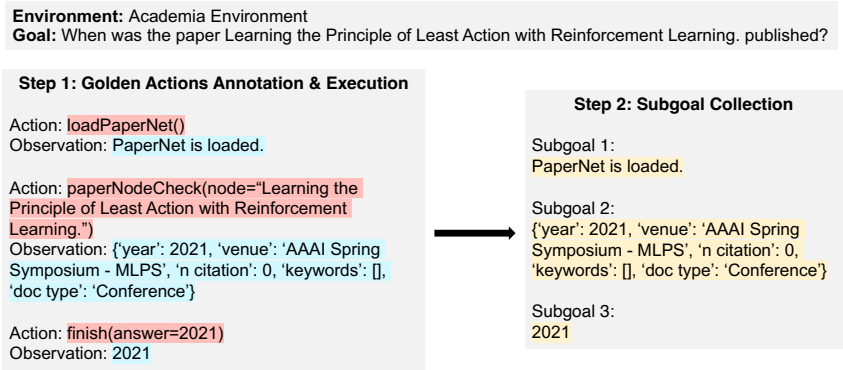


Figure 8: An illustration of the process of subgoal annotation for Academia Environment.

as follows:

$$r^{\text{match}} = \frac{n}{m+n}(r_d(r_q + r_p)) + \frac{m}{n+m} r_c \quad (n = 3; m = 0, 1, 2, \ldots) \tag{5}$$

Initially, we dissect the URL into its constituent elements: domain, query, and parameters by using `util.parse`. For domain verification, a binary value, r_d, is assigned, with a score of 1 indicating a correct domain match, and 0 otherwise. Subsequently, the matching score for the query, r_q, is determined through the application of the Longest Common Subsequence (LCS) algorithm, which assesses the similarity between the current and target queries based on their sequential nature. In contrast, the alignment between the current and target parameters is evaluated using the F1 score, denoted as r_p, which is particularly suited for unordered sets.

In parallel, the content matching score, r_c, emerges from the analysis of required content presence at each stage, calculated as the ratio of detected essential contents to the total required contents.

The overall progress rate integrates these two aspects, calculated as a weighted sum of the URL matching scores (incorporating domain, query, and parameter scores) and the content matching score. Here, n represents the number of target URL components, and m denotes the count of target required contents.

H.8 Tool-Query

In Tool-Query Environments, we employ r_t^{subgoal} as a metric to measure progress rate. Therefore, it is necessary to annotate subgoals for these envrionments. In Figure 8, we present an illustration of the process of subgoal annotation for Academia Environment. Specifically, when designing actions for these environments, we ensure that each action's functionality is indecomposable (i.e., the functionality and outcome of one action can not be achieved through other actions). This design choice results in a deterministic set of required golden actions to achieve our annotated goal. Furthermore, we ask human annotators to identify golden actions for each goal. Every output returned by executing golden actions is then processed as a subgoal.

H.9 Tool-Operation

For Todo Environment, we adopt r_t^{subgoal} as progress rate metric. Subgoals are annotated following the same process as Tool-Query Environments. In Sheet Environment, progress rate is assessed with r_t^{match}. Specifically, we first ask human annotators to annotate the golden spreadsheet for each goal. During the evaluation process, we calculate the matching score after each interaction round. The matching score is determined by the proportion of cells in current spreadsheet that align [10] with the golden spreadsheet.

[10] A cell is aligned if and only if its value is the same as the cell in the same position on the golden spreadsheet.

Model	Device/API	Inference Architecture	Inference Speed	Total-time
GPT-4	azure API	-	1.5s/round	5.5h
GPT-3.5-Turbo	azure API	-	1s/round	3h
DeepSpeed-67b	8*V100	vLLm	5s/round	18.5h
Llama2-70b	8*V100	vLLm	8s/round	28h
Llama2-70b	4*A100	vLLm	4s/round	13.5h

Table 13: Inference Time Estimation

I RUNTIME ESTIMATION

The evaluation runtime for a language model depends on the device/API, model, and inference architecture used. In the case of open-source LLMs, the vllm inference speed is approximately 10 times faster than the huggingface pipeline. We show some time cost in Table 13.

J PROMPT DETAILS

As shown in Figure 9, we use a unified prompt template for different tasks in AGENTBOARD. Basically, a prompt consists of 5 parts. {System Prompt} represents the system prompt for the LLM, such as "You are a helpful AI agent". {Instruction} mainly consists of task descriptions and action definitions. {Examples} represents in-context learning examples. {Goal} is the current goal that needs to be accomplished, and {Trajectory} is the interaction history between the LLM agent and the environment.

For different tasks, the contents of these five parts are different. Prompt details for Embodied AI tasks are shown in Figure 10, 11 and 12. Prompt details for Game tasks are shown in Figure 13 and 14. Prompt details for Web tasks are shown in Figure 15 and 16. Prompt details for Tool tasks are shown in Figure 17 and 18, respectively.

Unified Prompt Template

{System Prompt}
{Instruction}
Here are examples:
{Examples}
{Goal}
{Trajectory}

Figure 9: The unified prompt template in AGENTBOARD. {text} in blue font represents placeholders, which varies according to different tasks.

Prompt Details for AlfWorld

System Prompt
You are a helpful assistant. Generate your next step of action after Action. Action must not be empty. e.g. Action: put down cup.

Instruction
Your task is to interact with a virtual household simulator to accomplish a specific task. With each interaction, you will receive an observation. Your role is to decide on an action based on the observation. Please ensure that any objects ('obj') and receptacles ('recep') you mention in your response are present in the observation provided.
Here are the available actions you can take:
- take obj from recep
- put obj in/on recep

- open recep
- close recep
- toggle obj/recep
- clean obj using recep
- cool obj using recep
- heat obj using recep
- inventory
- examine recep/obj
- go to recep

Examples
Your task is to: look at statue under the desklamp.
You are in the middle of a room. Looking quickly around you, you see a coffeetable 1, a diningtable 1, a drawer 4, a drawer 3, a drawer 2, a drawer 1, a dresser 1, a garbagecan 1, a sidetable 2, a sidetable 1, and a sofa 1.
Action: go to dresser 1
Observation: On the dresser 1, you see a cellphone 3, a newspaper 2, a statue 1, and a television 1.
Action: take statue 1 from dresser 1
Observation: You pick up the statue 1 from the dresser 1.
Action: go to sidetable 1
Observation: On the sidetable 1, you see nothing.
Action: go to sidetable 2
Observation: On the sidetable 2, you see a desklamp 3, a newspaper 1, and a statue 2.
Action: use desklamp 3
Observation: You turn on the desklamp 3.

Figure 10: Prompt details for AlfWorld. The provided example is changed based on the type of the specific environment instance.

Prompt Details for ScienceWorld

System Prompt
You are a helpful agent that interacts with the virtual science school environment to solve the given task. Generate your next step of action after Action. Action must not be empty. e.g. Action: put down cup.

Instruction
You are an agent in a virtual science school environment, tasked to interact with various elements. Here are the commands you can use:

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

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

- **Device Operations**:
- activate OBJ / deactivate OBJ: Toggle a device.
- use OBJ [on OBJ]: Utilize a device or item.

- **Movement**:
- go to LOC: Relocate.

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

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

Where:
- OBJ: Object
- LOC: Location
- [DURATION]: Specified time

Examples
Task Description: Your task is to boil water. For compounds without a boiling point, combusting the substance is also acceptable. First, focus on the substance. Then, take actions that will cause it to change its state of matter.

ACTION: look around
OBSERVATION: This room is called the hallway. In it, you see:
a picture
a substance called air
the agent
You also see:
A door to the green house (that is open)
A door to the living room (that is open)
A door to the art studio (that is open)
A door to the kitchen (that is open)
A door to the bedroom (that is open)
A door to the workshop (that is open)

ACTION: open door to kitchen
OBSERVATION: The door is already open.

ACTION: go to kitchen
OBSERVATION: You move to the kitchen.
...

Figure 11: Prompt Details for ScienceWorld.

Prompt Details for BabyAI

System Prompt
You are an exploration master that wants to finish every goal you are given. Generate your next step of action after Action. Action must not be empty. e.g. Action: put down cup.

Instruction
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.

Examples
Observation: 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.
Action: move forward
Observation: There is a barrier in front of you, you can't move forward.
Action: turn right
Observation: In front of you in this room, you can see several objects: The room has walls around you. You are facing a wall 2 steps away. You are not carrying anything. Action: move forward
Observation: 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.
Action: turn left
Observation: 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.
Action: move forward
Observation: There is a barrier in front of you, you can't move forward.
Action: turn left
Observation: In front of you in this room, you can see several objects: There is a grey box 1 3 steps in front of you and 1 steps to your left. There is a grey ball 1 2 steps in front of you and 2 steps to your left. There is a red ball 1 right in front of you 5 steps away. There is a grey key 1 4 steps in front of you and 1 steps to your left. The room has walls around you. You are facing a wall 6 steps away. You are not carrying anything.
Action: move forward
Observation: In front of you in this room, you can see several objects: There is a grey box 1 2 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 left. There is a red ball 1 right in front of you 4 steps away. There is a grey key 1 3 steps in front of you and 1 steps to your left. The room has walls around you. You are facing a wall 5 steps away. You are not carrying anything.
...

Figure 12: Prompt details for BabyAI.

Prompt Details for Jericho

System Prompt
You are a game master in fictional text games. Generate your next step of action after Action. Action must not be empty. e.g. Action: put down cup.

Instruction
You are in a fictional game environment and you need to accomplish goals by performing actions. Each action is a simple phrase. Here are the actions you can do:
- Inventory: check things you are carrying

- Look: check your surroundings
- Examine place/object: check the details of something
- Take object: pickup object
- Put down object: leave a object at your current place.
- Drop object
- Check valid actions: Check actions you can use
- South: go south
- North: go north
- East: go east
- West: go west
- Up: go up
- Down: go down

Other available actions could be determined through check valid actions.

Examples

Goal: You are the warrior Link that needs to save the princess from the castle.
Observation: You are at the path leading to the castle. The castle is to your north. There is a barrel in front of you.
Action: Examine barrel
Observation: There is a gleaming elvish sword.
Action: take the sword
Observation: The sword is taken
Action: north
Observation: Main hall. The castle is dimly lit, with lava flowing in many places. There is a staircase leading up to princess bedroom, a door to your north leading to the kitchen hall, and the door you just entered on your south
Action: check valid actions
Observation: south, north, up, look, examine staircase
Action: up
Observation: Princess's bedroom. Princess is sleeping in bed.
Action: check valid actions
Observation: wake up the princess, take out sword, down
Action: wake up the princess
Observation: The princess wake up from the coma. Thank you my knight, she says. The task is finished.

Figure 13: Prompt details for Jericho.

Prompt Details for PDDL

System Prompt

You are a master in planning. Generate your next step of action after Action. Action must not be empty. e.g. Action: put down cup.

Instruction

The robot has four actions: pickup, putdown, stack, and unstack. The domain assumes a world where there are a set of blocks that can be stacked on top of each other, an arm that can hold one block at a time, and a table where blocks can be placed.
The actions defined in this domain include:
pickup `<block>`: allows the arm to pick up a block from the table if it is clear and the arm is empty. After the pickup action, the arm will be holding the block, and the block will no longer be on the table or clear.
putdown `<block>`: allows the arm to put down a block on the table if it is holding a block. After the putdown action, the arm will be empty, and the block will be on the table and clear.
stack `<block> <block>`: allows the arm to stack a block on top of another block if the arm is holding the top block and the bottom block is clear. After the stack action, the arm

will be empty, the top block will be on top of the bottom block, and the bottom block will no longer be clear.
unstack <block> <block>: allows the arm to unstack a block from on top of another block if the arm is empty and the top block is clear. After the unstack action, the arm will be holding the top block, the top block will no longer be on top of the bottom block, and the bottom block will be clear.

Examples
Goal: The goal is to satisfy the following conditions: b1 is on b2., b2 is on b3. Observation: b1 is on the table. b2 is on the table. B3 is on the table. Robot arm is empty. The b1 is clear. The b2 is clear. The b3 is clear.
Action: pickup b2
Observation: b1 is on the table. B2 is on the table. The b1 is clear. The b3 is clear. You are holding b2.
Action: stack b2 b3
Observation: b1 is on the table. b1 is on b2. B3 is on the table. Robot arm is empty. The b1 is clear. The b2 is clear.
Action: pickup b2.
Observation: The action is not valid and therefore takes no effect. Please remember to satisfy the restriction of actions. You can also check valid actions.
Action: check valid actions.
Observation: valid actions are: pickup b2, unstack b1 b2.
Action: pickup b1
Observation: b2 is on b3. B3 is on the table. Robot arm is empty. The b2 is clear. You are holding b1.
Action: stack b1 b2
Observation: b1 is on b2. b2 is on b3. B3 is on the table. Robot arm is empty. The b1 is clear. The goal is satisfied.

Figure 14: Prompt details for PDDL. The provided instruction/example are changed based on the type of the specific environment instance.

Prompt Details for WebShop

System Prompt
You are a helpful virtual webshop assistant that interacts with the simulated website to solve a task.

Instruction
You are now the virtual webshop assistant, navigating a website to locate and purchase items based on given commands. Our interaction will follow this structure:

Your Actions: You will preface each of your actions with "Action: ".
Website's Response: The website will provide feedback starting with "Observation: ".

[click]something: Engage with specific buttons or links.
[search]something: Seek specific data on the website. Use this only if a [Search] button appears in the observation.
Note: If you wish to search and there's no [Search] button, click the [Back to Search] button instead.

Observation Format: The website will showcase its content. Elements within square brackets (like [Buy Now]) indicate clickable buttons or links.

Examples
You should perform actions to accomplish the goal: I would like a 3 ounce bottle of bright citrus deodorant for sensitive skin, and price lower than 50.00 dollars
Action: reset[]
Observation: WEB PAGE: {Let us begin shopping
[Search]}

Action: search[3 ounce bright citrus deodorant sensitive skin]
Observation: WEB PAGE: {
[Back to Search]
Page 1 (Total results: 15)
[Next >]
[B078GWRC1J]
Bright Citrus Deodorant by Earth Mama — Natural and Safe for Sensitive Skin, Pregnancy and Breastfeeding, Contains Organic Calendula 3-Ounce
$10.99}

Action: click[B078GWRC1J]
Observation: WEB PAGE: {
[Back to Search]
[<Prev]
scent [assorted scents][bright citrus][calming lavender][ginger fresh][simply non-scents]
size [3 ounce (pack of 1)]}

Action: click[bright citrus]
Observation: You have clicked bright citrus.

Action: click[3 ounce (pack of 1)]
Observation: You have clicked 3 ounce (pack of 1).

Action: click[Buy Now]
Observation: You have bought 3 ounce (pack of 1).

Figure 15: Prompt details for WebShop.

Prompt Details for WebArena

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.

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

The useful websites and corresponding URL you can navigate:
'reddit': "http://reddit.com"
'online shop': "http://onestopmarket.com"
'e-commerce platform': "http://luma.com/admin"
'gitlab': "http://gitlab.com"

‘wikipedia’: “http://wikipedia.org”
‘map’: “http://openstreetmap.org”

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 [directio n= down |up]’: Scroll the page up or down.

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

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

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

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

Examples
You should perform actions to accomplish the goal: Add a white desk to my wish list
Observation:
WINDOWED PAGE:{
Tab 0 (current): One Stop Market
[1] RootWebArea ‘One Stop Market’ focused: True
    [1254] link ‘My Wish List’
    [2427] StaticText ‘Search’
    [1585] combobox ‘Search’ autocomplete: both hasPopup: listbox required: False expanded: False
    [2430] link ‘Advanced Search’
    [1588] button ‘Search’ disabled: True
}
URL: http://onestopmarket.com

Action: @type [1585] [white desk] [press_enter_after=1]@
Observation:
WINDOWED PAGE:{

Tab 0 (current): Search results for: 'white desk'
[2635] RootWebArea "Search results for: 'white desk'" focused: True
    [3869] link 'My Wish List'
    [4827] StaticText 'Search'
    [4072] combobox 'Search' autocomplete: both hasPopup: listbox required: False expanded: False
        [5027] StaticText 'white desk'
    [4830] link 'Advanced Search'
    [4075] button 'Search' disabled: True
    [3729] main ''
        [3842] heading "Search results for: 'white desk'"
        [3907] StaticText 'Items 1-12 of 38823'
        [4781] link 'Image'
            [4833] img 'Image'
        [4783] link 'Image'
            [4849] img 'Image'
}
URL: http://onestopmarket.com/catalogsearch/result/?q=white+desk
...

Figure 16: Prompt details for WebArena. The provided example is changed based on the type of the specific environment instance.

Prompt Details for Academia

System Prompt
You can use actions to help people solve problems.

- - - -

- 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

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]”

Examples
Goal: When was the paper Learning the Principle of Least Action with Reinforcement Learning. published?

Action: loadPaperNet with Action Input: {}
Observation: PaperNet is loaded.
Action: paperNodeCheck with Action Input: {“node”:“Learning the Principle of Least Action with Reinforcement Learning.”}
Observation: {‘year’: 2021, ‘venue’: ‘AAAI Spring Symposium - MLPS’, ‘n_citation’: 0, ‘keywords’: [], ‘doc_type’: ‘Conference’}
Action: finish with Action Input: {“answer”: “2021”}
Observation: 2021

Figure 17: Prompt Details for Academia in Tool-Query Environments.

Prompt Details for Todo

System Prompt
You can use actions to help people solve problems.

Instruction
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 Todoist 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 Todoist 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. Incluing the Place, Tips, etc.

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

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

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

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

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

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

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

Examples
Goal: Is Prepare for history quiz a task of School project? Please answer yes or no.

Action: get_projects with Action Input: {}

Observation: [{‘id’: ‘12345’, ‘order’: 0, ‘color’: ‘charcoal’, ‘name’: ‘School’, ‘is_favorite’: false}]
Action: get_tasks with Action Input: {“project_id”: “12345”}
Observation: [{‘id’: ‘123451’, ‘order’: 0, ‘content’: ‘Prepare for history quiz’, ‘is_completed’: false, ‘priority’: 1, ‘due_date’: ‘2030-10-10’}, {‘id’: ‘123452’, ‘order’: 1, ‘content’: ‘Prepare for math quiz’, ‘is_completed’: false, ‘priority’: 1, ‘due_date’: ‘2030-11-10’}]
Action: finish with Action Input: {“answer”: “yes”}
Observation: yes

Figure 18: Prompt Details for Todo in Tool-Operation Environments.