

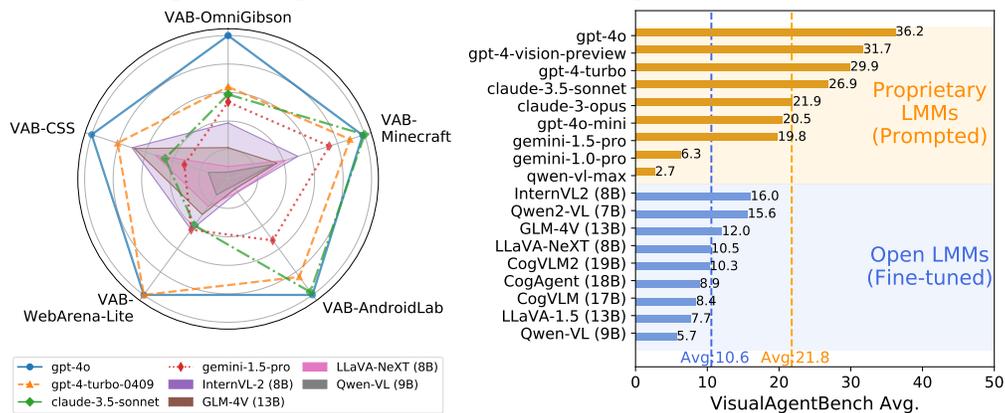
VISUALAGENTBENCH: TOWARDS LARGE MULTI-MODAL MODELS AS VISUAL AGENTS

Anonymous authors

Paper under double-blind review

ABSTRACT

Large Multimodal Models (LMMs) have ushered in a new era in artificial intelligence, merging capabilities in both language and vision to form highly capable visual agents that are postulated to excel across a myriad of tasks. However, existing benchmarks fail to sufficiently challenge or showcase the full potential of LMMs as agents in complex, real-world environments. To address this gap, we introduce VisualAgentBench (VAB), a comprehensive and unified benchmark specifically designed to train and evaluate LMMs as visual agents across diverse scenarios in one standard setting, including Embodied, Graphical User Interface, and Visual Design, with tasks formulated to probe the depth of LMMs’ understanding and interaction capabilities. Through rigorous testing across 9 proprietary LMM APIs and 9 open models (18 in total), we demonstrate the considerable yet still developing visual agent capabilities of these models. Additionally, VAB explores the synthesizing of visual agent trajectory data through hybrid methods including Program-based Solvers, LMM Agent Bootstrapping, and Human Demonstrations, offering insights into obstacles, solutions, and trade-offs one may meet in developing open LMM agents. Our work not only aims to benchmark existing models but also provides an instrumental playground for future development into visual agents. Code, train, and test data will be available at <https://anonymous.4open.science/r/VisualAgentBench-AFA0/>.



(a) Typical LMMs’ VAB performance (relative) against the best in each environment. (b) Average VAB Success Rates of tested LMMs across 5 environments. Dashed lines for two LMM types’ average.

Figure 1: Overview of Proprietary and Open LMMs on VISUALAGENTBENCH. After Behavior Cloning (BC) on trajectories, Open LMMs demonstrate potential to serve as visual agents.

1 INTRODUCTION

Recent advancements in Foundation Models, particularly Large Language Models (LLMs) (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023; Zeng et al., 2022) and Large Multimodal Models (LMMs) (Liu et al., 2024c; OpenAI, 2023; 2024a; Anthropic, 2024), have showcased their profound capabilities in understanding and processing vast amounts of world knowledge, factual information, and common sense reasoning. Notably, these models have demonstrated potential as

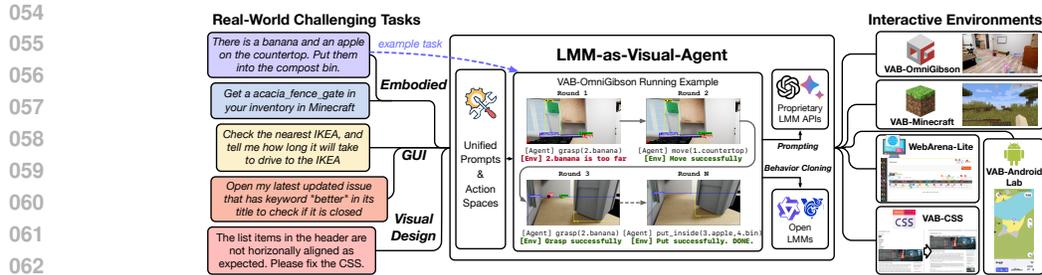


Figure 2: VISUALAGENTBENCH is the first systematic benchmark for training and evaluating LMM-as-Visual-Agent with both proprietary and open LMMs across a diverse set of practical challenges. Based on created tasks, we unify the benchmarking of both proprietary LMM APIs via prompting and open LMMs via behavior cloning training in interactive environments.

intelligent agents (Searle, 1970; Maes, 1994; Wooldridge & Jennings, 1995), addressing a broad spectrum of real-world challenges (Liu et al., 2023a). LMMs, in particular, enhance the capabilities of these agents by integrating visual inputs, thereby expanding the scope of intelligent agent applications.

However, the setup for LMM-as-Visual-Agent remains underdeveloped. Most existing evaluations on LMMs focus on traditional vision tasks (Singh et al., 2019; Lu et al., 2022; Liu et al., 2023b; Kazemzadeh et al., 2014), or on performance in standardized human exams (Yue et al., 2023; Lu et al., 2023). They rarely measure the models’ higher-level planning abilities or specific strengths as visual agents. In contrast, the role of LLMs as agents in text environments has been extensively explored and validated as a reliable measure of their capabilities (Yao et al., 2023; Liu et al., 2023a).

Recent benchmarks for multimodal agents, while valuable, do not adequately address the comprehensive multitask evaluation required for LMM-as-Visual-Agent. These benchmarks often limit their focus to single environments such as Household (Shridhar et al., 2020a;b), Gaming (Fan et al., 2022; Wu et al., 2023), Web (Deng et al., 2024; Zhou et al., 2023; Koh et al., 2024), or Desktop (Xie et al., 2024; Kapoor et al., 2024). The narrow scope and varied settings prevents a holistic assessment of LMMs’ multitask agent capabilities. Furthermore, the prevalent prompting-only evaluation in existing benchmarks does not suffice for open LMMs (Liu et al., 2024c; Bai et al., 2023), which typically show limited instruction-following capabilities so far, thus hindering a comprehensive evaluation.

To bridge this gap, we introduce VISUALAGENTBENCH (VAB)—the first systematic benchmark to multitask train and evaluate visual agents across a diverse array of realistic vision-centric tasks. We present three representative scenarios and develop five distinct datasets for this study: **Embodied** (VAB-OmniGibson, VAB-Minecraft), **Graphical User Interface (GUI)** (VAB-AndroidLab (Anonymous, 2024), VAB-WebArena-lite (Zhou et al., 2023)), and **Visual Design** (VAB-CSS), enabling comprehensive testing and development of agents that can navigate complex spaces, interact with digital interfaces, and understand aesthetic and functional aspects of visual design. This diversity not only challenges the agents’ capabilities across different settings but also enhances their adaptability and utility in practical applications, paving the way for more robust and versatile visual agents.

Contributions. Our main contributions in VAB are as follow: Standardized Environments, LMM-oriented Test Data Creation, Train Data Synthesis, Extensive Benchmarking, and Analytical Insights.

- **Standardized Environments.** VAB standardizes the interfaces, prompting, and data formats to facilitate a consistent evaluation of LMM agents across environments. The efforts include adapting previously LMM-irrelevant environment (OmniGibson) and creating new ones (VAB-AndroidLab, VAB-CSS). Each task is assessed through interactive evaluation, where LMMs engage directly with the environment, and their performance is measured by specific judge functions. The feature substantially distinguishes VAB from many other benchmarks (Deng et al., 2024; Rawles et al., 2024; Kapoor et al., 2024) based on offline human-annotated trajectories.
- **LMM-oriented Test Data Creation.** Test set queries and judge functions are newly created for all environments except for WebArena. To acquire massive tasks, we develop a “Prototype-Instantiation” method to evolve valid task instructions and judge functions grounded on environments. We refer to task categories and prototypes in some previous study Li et al. (2023); Zhu et al. (2023) which unsatisfies the practical use for evaluating LMM agents or has no public data.
- **Train Data Synthesis.** VAB strives to offer the first multitask multi-environment trajectory train set for developing LMM agents, containing 4,482 high-quality training trajectories spanning 5

environments. It explores and investigate how to synthesize multimodal agent task instructions and trajectory data via hybrid strategies of *Program-based Solvers*, *LMM Agent Bootstrapping*, and *Human Demonstrations*. Thereby, it also for the first time enables a holistic agent evaluation that includes many open LMMs with valid scores for meaningful comparison.

- **Extensive Benchmarking & Analysis.** Our extensive testing over 18 LMMs, including 9 proprietary LMM APIs and 9 open LMMs, demonstrates the impressive progress of LMM-as-Visual-Agent. Top proprietary LMMs, such as `gpt-4o`, are solving 36.2% of challenging problems with mere prompting. Behavior cloning (BC) on the VAB train set remarkably enhances the capabilities of open LMMs as visual agents, with most surpassing the performance of weaker proprietary LMMs and approaching close towards the strong `gemini-1.5-pro`.
- **Analytical Insights.** VAB provides deep insights into the general status quo and detailed dimensions of grounding and planning for LMM agents. For grounding, we quantitatively analyze the use of object labels, set-of-marks, and visual difference ability. For planning, we study the actual impact of Chain-of-Thought and error recovering ability for LMM agents.

2 VISUALAGENTBENCH: TASKS AND ENVIRONMENTS

In this section, we will first introduce the problem definition of LMM-as-Visual-Agent, and then the detailed description of each environment and dataset.

LMM-as-Visual-Agent. An agentic problem could be generally formulated as a Partially Observable Markov Decision Process (POMDP) problem, where \mathcal{S} denotes the state space, \mathcal{A} denotes the action space, \mathcal{T} denotes the transition function, \mathcal{R} refers to the reward function, \mathcal{I} refers to the instruction space, and \mathcal{O} refers to the observation space. Compared to LLM-as-Agent (Liu et al., 2023a), the observation space \mathcal{O} must incorporate visual inputs (e.g., images or videos) in LMM-as-Visual-Agent, significantly extending the application scope but also casting a substantial challenge for LMMs to reconcile their multimodal understanding and high-level reasoning.

Overview of VAB. In VAB, we carefully select the most representative and promising tasks that could be enabled by LMM-based agents. These tasks generally fall into three categories: embodied agents, including household and game environments; GUI agents, covering mobile and web apps; and visual design agents, focusing on frontend CSS debugging (Figure 2). They span diverse domains and feature unique challenges, providing an ideal testbed for a comprehensive evaluation of LMM-based agents. When constructing VAB, we strictly follow the principles outlined in Appendix A.1. Our efforts focus on addressing gaps in evaluating LMM-based agents while leveraging existing resources to avoid redundancy, ensuring all our work is meaningful and avoids reinventing the wheel. For 4 out of 5 tasks, we collect new data from scratch. For web agents, we adapt and clean WebArena (Zhou et al., 2023) as our test set, as it is already suitable for LMM-based evaluation. For household agents, we use the OmniGibson environment from Behavior-1k (Li et al., 2023) and create new tasks based on high-level actions we defined, which are crucial for evaluating LMM-based agents and absent in existing datasets. We similarly construct our tasks in Minecraft using the MineRL environment¹ with our self-defined high-level actions. Finally, for our mobile app and CSS debugging tasks, we create new interactive environments due to the lack of suitable resources in the literature and collect data based on these environments. An overview of VAB is shown in Table 2.

2.1 EMBODIED AGENT

Embodied agents have been a central topic in AI, naturally involving multimodal sensory data, including language and vision signals. The multimodal capabilities of LMMs could enable new possibilities for embodied agents.

VAB-OmniGibson. One of the most actively researched environments in embodied AI is the household environment due to its complexity and range of everyday tasks (Huang et al., 2022; Song et al., 2023; Shridhar et al., 2020a). We build the household environment for embodied agents using OmniGibson, a high-fidelity simulator based on Nvidia Omniverse that features diverse scenes and realistic physical effects.² An example activity in VAB-OmniGibson would be “Put all 8 plates from

¹<https://minerl.readthedocs.io>

²<https://www.nvidia.com/en-us/omniverse/>

162 *the countertops into the cabinet in the kitchen*”, where agents should accomplish the tasks using
 163 provided high-level actions (e.g., “grasp”, “put_inside”). We adopt the task *success rate* as the
 164 evaluation metric. (Cf. Appendix B).

165 **VAB-Minecraft.** Minecraft has become a popular open-world environment for developing generalist
 166 embodied agents due to its diverse tasks (e.g., survival, harvesting, crafting, combat, and creative
 167 tasks), varied environments, and interactive mobs, necessitating generalized agent abilities (Fan et al.,
 168 2022; Lifshitz et al., 2024). In VAB-Minecraft, the agent is expected to accomplish a wide range
 169 of tasks, including item collection and killing hostile mobs. An example task in VAB-Minecraft
 170 would be “*Get a fishing rod in your inventory*”, and the LMM agent need to interact with the game
 171 environment using provided scripts (e.g., “craft”, “smelt”) or calling a low-level controller Steve-
 172 1 (Lifshitz et al., 2024) with prompt. We adopt the task *success rate* as metric. (Cf. Appendix C)

174 2.2 GUI AGENT

175 GUI is another typical scenario where LMM agents may excel. Compared to embodied environments,
 176 GUI environments are more information-intensive and require a good understanding of UI elements
 177 and layouts. We provide two interactive and reproducible GUI environments, Mobile (i.e., Android)
 178 and WebArena, to evaluate LMM GUI agents in a practical manner.

179 **VAB-AndroidLab (Anonymous, 2024).** Automated agents on Android GUI are instrumental.
 180 Although pioneer works like (Burns et al., 2022; Rawles et al., 2024) have explored training and
 181 evaluating these agents, they typically use Step Success Rate evaluated offline. Recent works (Yang
 182 et al., 2023b; Wang et al., 2024a) leverage LMMs as Android GUI agents but lack reproducible
 183 executive evaluation frameworks. We address this by creating tasks for LMM agents to perform
 184 human-like actions (e.g., Tap, Swipe) on smartphones using Android Virtual Device (AVD). For
 185 example, “*Find a hotpot restaurant nearby and make a reservation for me tonight.*” Agents must
 186 understand the Android GUI and make decisions based on screen observations. (Cf. Appendix D)

187 **VAB-WebArena-Lite (Zhou et al., 2023).** Web browsing is an ideal testbed for evaluating LMMs
 188 as GUI agents. Previous works (Shi et al., 2017; Liu et al., 2018; Deng et al., 2024; Yao et al.,
 189 2022) mainly focus on offline evaluation. We adopt WebArena (Zhou et al., 2023), a benchmark for
 190 text-based web GUI agents with 812 tasks across 5 websites. LMMs perform tasks based on user
 191 instructions, such as finding and summarizing customer reviews on OneStopShop. We use HTML
 192 SoM (Koh et al., 2024) to annotate operable HTML elements, enabling LMMs to generate actions
 193 via `playwright` . WebArena-Lite is a subset of 165 tasks, refined and adapted for multimodal
 194 evaluation, removing cross-website tasks and fixing implausible conditions. (Cf. Appendix E)

196 2.3 VISUAL DESIGN AGENT

197 Visual design tasks demand a nuanced understanding of visual signals, which text-only LLMs cannot
 198 handle with any easy augmentation, unlike embodied or GUI agent tasks that can rely on external
 199 object detectors (Song et al., 2023) or textual representations like accessibility trees (Xie et al., 2024).

200 **VAB-CSS.** We create a new task to evaluate LMMs on web frontend design, focusing on CSS style
 201 adjustments. Fixing CSS styles is a labor-intensive task that often requires engineers to iteratively
 202 adjust an element through trial and error. Such a task inherently entails fine-grained visual grounding
 203 and reasoning across a series of rendering outcomes resulting from iterative CSS edits. In VAB-CSS,
 204 the agent iteratively edits the CSS style using provided tools until it thinks the rendering matches
 205 a given target design. We adopt *success rate (SR)* as the metric, which evaluates whether the final
 206 rendering matches the target design. (Cf. Appendix F)

208 3 METHODOLOGY FOR VAB DATA COLLECTION

209 For agent tasks, it is known to be very challenging to design practical and verifiable task instances;
 210 let alone creating high-quality training trajectories on top of them later. In constructing VAB, we
 211 not only aim to deliver a high-quality agent benchmark but also endeavor to develop a systematic
 212 methodology for the problem of LMM-as-Visual-Agent data curation. For task instance collection,
 213 we follow a two-stage paradigm (*prototyping* and *instantiation*) for each new task instance to ensure
 214 data quality and executability. Additionally, we harness a suite of hybrid strategies to collect training
 215

Table 1: Recommendation levels for 3 strategies used in curating VAB’s agent-tuning trajectory data on different dimensions. (Cf. Section 3.2 for detailed explanation on each dimension)

Strategy	Avg. Cost	Adaptability	Versatility	Flexibility	Adoption
Program-based Solvers	★★★	★★★	★★★	★★★	VAB-OmniGibson, VAB-WebArena-Lite
LMM Agent Bootstrapping	★★★	★★★	★★★	★★★	VAB-Minecraft, VAB-AndroidLab, VAB-CSS
Human Demonstrations	★★	★★	★★★	★★★	VAB-AndroidLab

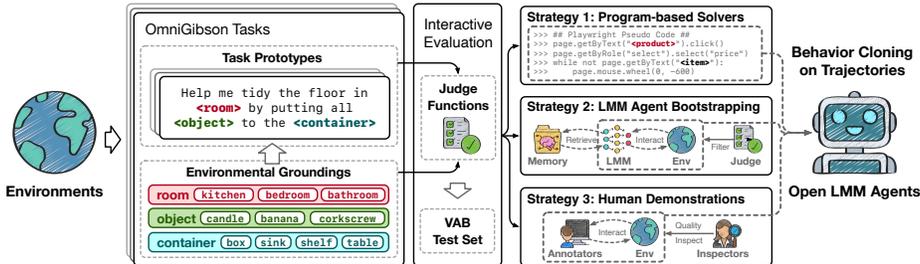


Figure 3: Data collection process in VAB. We follow a principled two-stage paradigm to collect task instances and then adopt various methods to further collect training trajectories for them.

trajectories that can be used to tune open LMMs into better visual foundation agents. Our rigorous data collection process in VAB is crucial for presenting a high-quality resource for LMM-based agents (Figure 3). The statistics of different tasks in VAB are shown in Table 3.

3.1 TASK INSTANCE COLLECTION: PROTOTYPING AND INSTANTIATION

Curating meaningful and testable task instances for LMM agent tasks can be difficult. On one hand, they should be diverse and useful to cover real-world applications. On the other hand, they should be grounded to environments carefully to ensure feasibility and practicality. As a result, we collect all our task instances in a two-stage paradigm:

- **Prototyping:** We gather many task prototypes representing high-level goals based on the functionality provided by the environment. Related items are temporarily set to placeholders.
- **Instantiation:** Task prototypes are grounded to concrete items and conditions collected from the environment. Judging functions are thereby set up by instantiated tasks. Instructions are then rephrased by LLMs to enhance expression diversity.

Following the mentioned principles, we develop diverse task sets for all VAB environments. For VAB-OmniGibbon, we source 89 prototypes of general household activities, instantiating them in various scenes to create 992 instances, with 181 reserved for testing. In VAB-Minecraft, we focus on object-collecting tasks, generating 628 instances from our defined prototypes, allocating 116 for testing. For VAB-AndroidLab, we select 8 Android applications and create 119 test instructions, while developing approximately 70 task prototypes from 18 common apps for training. In WebArena-Lite, we refine 165 test samples and generate 1186 training instances from 40 task prototypes across various web applications. For VAB-CSS, we design tasks around CSS rule corruptions, creating 1210 instances with 165 for testing, each annotated with natural language descriptions of visual changes. This comprehensive approach ensures a rich and varied task environment across our selected domains. For more details, please refer to Appendix A.3.

3.2 TRAINING TRAJECTORY COLLECTION: 3-LEVELED STRATEGIES

Recently, there has been a rise in benchmarks for interactively evaluating LLM or LMM agents (Liu et al., 2023a; Zhou et al., 2023; Xie et al., 2024). Despite showcasing the substantial potential of LLM and LMM as agents, they usually only provide the test set and thus fail to facilitate the improving of open LLMs and LMMs on agent tasks. In light of the challenge, in VAB we are devoted to offering a preliminary behavior cloning (BC) (Nakano et al., 2021; Zeng et al., 2023) setup for training open LMM agents. In VAB, we summarize our trajectory collecting into 3-leveled strategies:

1. **Program-based Solvers:** Trajectories are collected by prototype-specific programs written by human experts (e.g., Playwright scripts for automating web browsing tasks).

Table 2: Comparison between VAB and related benchmarks. VAB is the first comprehensive multi-domain agent benchmark offering interactive environments, supporting multimodal agent evaluation, and providing a large and diverse set of training trajectories for visual agent tuning. “#Test Ins.” refers to the number of test instances; “#Train Traj.” refers to the number of training trajectories for SFT, “RL” means no training trajectory is available and only a reinforcement learning setup is provided; “#Avg. Turns” refers to the average number of turns per training trajectory.

	Category	#Env.	#Test Ins.	#Train Traj.	#Avg. Turns	Multimodal	Interactive Env.
ALFWorld Shridhar et al. (2020b)	Household	1	134	6,374	7.54	✗	✓
Alfred Shridhar et al. (2020a)	Household	1	1,529	6,574	7.51	✓	✓
Behavior-1K Li et al. (2023)	Household	1	1,000	RL	-	✓	✓
MineDojo Fan et al. (2022)	Game	1	3,141	RL	-	✓	✓
SmartPlay Wu et al. (2023)	Game	6	20	-	-	✗	✓
Mind2Web Deng et al. (2024)	Web	1	1,341	1,009	7.71	✓	✗
WebArena Zhou et al. (2023)	Web	1	812	-	-	✓	✓
VisualWebArena Koh et al. (2024)	Web	1	910	-	-	✓	✓
META-GUI Sun et al. (2022)	Mobile	1	483	3,692	7.64	✓	✗
OSWorld Xie et al. (2024)	Desktop	1	369	-	-	✓	✓
OmniACT Kapoor et al. (2024)	Desktop & Web	2	9,802	-	-	✓	✗
AgentBench Liu et al. (2023a)	Multi-domain	8	1,091	-	-	✗	✓
VISUALAGENTBENCH	Multi-domain	5	746	4,482	11.22	✓	✓

Table 3: Statistics of all datasets in VAB.

	VAB-OmniGibson	VAB-Minecraft	VAB-AndroidLab	VAB-WebArena-Lite	VAB-CSS
#Action Space	20	6	7	12	4
#Test Instance	181	116	119	165	165
#Train Trajectory	872	382	1,213	1,186	829
#Train Step	20,153	5,197	10,175	9,522	5,250
#Max Round Limit	100	100	25	20	10

- LMM Agent Bootstrapping:** Trajectories are collected by prompted LMM agents (e.g., `gpt-4o`), with optional memory augmentations (Wang et al., 2023c). For instance, in Minecraft we allow agent to access memories for solving easier sub-goals (e.g., how to collect a stick) when constructing trajectories for more complex goals (e.g., how to collect a hammer).
- Human Demonstrations:** Trajectories are annotated by human experts. It is necessary for scenarios where humans are indispensable (e.g., mobile apps require logged-in human accounts).

These strategies are quite different from each other and present their own unique advantages in certain environments. We summarize their recommendation levels on 4 dimensions (Cf. Table 1):

- **Average Cost:** The most important dimension. Program-based solvers are most cost-effective, followed by human demonstrations. LMM bootstrapping is currently the most expensive due to proprietary API costs, but this may decrease as open LMMs improve.
- **Adaptability:** It indicates how easy we can implement a strategy to an environment. LMM bootstrapping is highly adaptable with good prompts. Program-based solvers require moderate implementation time. Human demonstrations need training and may face accessibility issues.
- **Versatility:** It refers to how versatile tasks a strategy could deal with. Human annotators can handle the widest range of tasks, followed by LMM agents. Program-based solvers are limited to predefined prototypes.
- **Flexibility:** It denotes the trial and error process in the solution trajectories. LMM bootstrapping naturally incorporates trial-and-error processes. Program-based solvers struggle with this, while human annotators are often discouraged from it for quality control reasons.

Considering all mentioned dimensions and their trade-offs, we adopt a hybrid set of strategies for each of the 5 environments in VAB as shown in Table 1. In brief, we employ diverse strategies tailored to each domain’s unique characteristics. For VAB-OmniGibson, we utilize program-based solvers due to the platform’s high hardware requirements and the need for cost-effective, adaptable solutions. In VAB-Minecraft, we opt for LMM agent bootstrapping to handle the game’s inherent randomness and exploration requirements. For VAB-AndroidLab, we primarily rely on human demonstrations, supplemented by LMM agent bootstrapping for offline apps, to address the challenges of XML legibility and app-specific login requirements. In VAB-WebArena-Lite, we choose program-based solvers, leveraging the mature `Playwright` automation tool and addressing the difficulties faced by

Table 4: Main results on VISUALAGENTBENCH. The metric reported is success rate (SR), which measures the proportion of successful tasks in all evaluated tasks. Open LMMs are evaluated using multitask fine-tuning rather than direct prompting, as they were unable to effectively follow system prompts from VAB in our preliminary trials. For # Params of open LMMs, we report the sizes of their language and vision part respectively.

Type	Model	#Params	AVG	Embodied		GUI		Visual Design
				OmniGibson	Minecraft	AndroidLab	WebArena-Lite	CSS
Proprietary LMMs (Prompting)	gpt-4o-2024-05-13	N/A	36.2	41.4	<u>55.2</u>	31.9	18.2	34.5
	gpt-4-vision-preview	N/A	<u>31.7</u>	36.5	47.4	26.9	<u>18.8</u>	<u>29.1</u>
	gpt-4-turbo-0409	N/A	29.9	26.5	50.0	26.9	18.2	27.9
	claude-3.5-sonnet	N/A	26.9	24.3	56.0	<u>31.1</u>	7.2	15.8
	claude-3-opus	N/A	21.9	14.9	51.7	15.1	7.9	20.0
	gpt-4o-mini-2024-07-18	N/A	20.5	12.2	30.2	22.7	20.6	17.0
	gemini-1.5-pro	N/A	19.8	22.1	41.4	16.8	7.9	10.9
	gemini-1.0-pro	N/A	6.3	4.4	11.2	11.8	4.2	0.0
	qwen-vl-max	N/A	2.7	0.0	6.0	2.5	3.0	1.8
	Open LMMs (Fine-tuning)	InternVL-2 (Chen et al., 2024)	7B + 0.3B	16.0	16.0	28.4	3.4	7.9
Qwen2-VL (Wang et al., 2024b)		7B + 0.3B	15.6	13.8	24.1	5.9	6.7	27.3
GLM-4V (GLM et al., 2024)		9B + 4B	12.0	8.8	19.8	2.5	5.5	23.6
LLaVA-NeXT (Liu et al., 2024b)		8B + 0.3B	10.5	3.3	23.3	3.4	4.2	18.2
CogVLM2 (Hong et al., 2024)		8B + 12B	10.3	3.3	25.9	1.7	3.0	17.6
CogAgent (Hong et al., 2023)		7B + 11B	8.9	6.6	20.7	2.5	0.6	13.9
CogVLM (Wang et al., 2023b)		7B + 10B	8.4	3.3	19.8	4.2	4.2	10.3
LLaVA-1.5 (Liu et al., 2024a)		13B + 1B	7.7	1.7	25.9	0.8	2.4	7.9
Qwen-VL (Bai et al., 2023)	7B + 2B	5.7	1.7	18.1	1.7	2.4	4.8	

human annotators with unfamiliar interfaces. For VAB-CSS, we implement LMM agent bootstrapping to accommodate the iterative nature of CSS debugging, using gpt-4o for initial trajectories and providing hints to improve success rates. For more details, please refer to Appendix A.4.

4 BASELINE EXPERIMENT

4.1 SETUP

Baselines. We evaluate on both proprietary LMM APIs and selected open LMMs. For proprietary LMMs, we include models from OpenAI GPT (OpenAI, 2024a; 2023; 2024b), Anthropic Claude (Anthropic, 2024), Google Gemini (Reid et al., 2024; Team et al., 2023), and Qwen-VL-Max (Bai et al., 2023). For open LMMs, we select nine state-of-the-art models as representative fine-tuning baselines in VAB: InternVL-2 (Chen et al., 2024), Qwen2-VL (Wang et al., 2024b), GLM-4V (GLM et al., 2024), CogVLM2 (Wang et al., 2023b), CogAgent (Hong et al., 2023), CogVLM (Wang et al., 2023b), LLaVA-NeXT (Liu et al., 2024b), LLaVA-1.5 (Liu et al., 2024a), Qwen-VL (Bai et al., 2023).

Prompting. We format LMM-as-Visual-Agent as two roles (i.e., `user` and `assistant`) interacting in multiple rounds. The task description, action spaces, few-shot demonstrations, and important notices for each environment are formatted as the `system` prompt at the beginning of the conversation. Task instruction is given in the first `user` round. Environmental observations and feedback are passed via `user` in later rounds. Considering current LMM APIs’ poorer support of multi-image and outrageous cost when interaction rounds soar up, in Embodied and GUI agents we only offer the vision input of the latest `user` round (following (Koh et al., 2024)) while reserving history text contents. An exception is the CSS agent in Visual Design. In this case, comparing differences in visual inputs is essential, and the interaction rounds are typically fewer than 10. Therefore, we retain all image inputs in the conversation history for this task.

Training for Open LMMs. We generally follow the prompting format of proprietary LMM APIs in each environment to organize our training trajectories, and make several minor modifications. In the system prompt we remove the few-shot demonstrations as we would fine-tune models. In addition, during fine-tuning, since open LMMs perform poorly on multi-image input (especially for CogVLM and CogAgent, whose expert architecture disallows simple implementation of multi-image input), we only use the vision input of the latest `user` turn, and concatenate histories together using role tokens (i.e., “<|user|>”) and linebreaks. For CSS agent where multi-image input is necessary, we

concatenate history images vertically into one as the input. To benchmark the potential of LMMs to serve as visual foundation agents, we conduct multitask fine-tuning over the dataset aggregation of all environments. To optimize performance, all LMMs undergo full-parameter fine-tuning, with a batch size of 64 and 5k training steps. Other hyperparameters are configured using the default ones provided by the model’s original repository or the third-party’s integrated training framework. For data composition, we uniformly combine all training samples except for VAB-CSS, which we duplicate an additional 2 times as the preliminary experiments show that the task requires more extensive training for open LMMs to adapt to the screenshot concatenation format.

4.2 MAIN RESULTS

Table 4 shows the main results on VAB, including both prompting proprietary LMMs and fine-tuned open LMMs. We have several important observations on the status quo of LMM-as-Visual-Agent.

VAB is challenging for existing LMMs. We observe that existing LMMs face significant challenges when evaluated on VAB. The majority of proprietary LMMs, with mere prompting, achieve an overall success rate above 20%, demonstrating their multimodal understanding and reasoning abilities. The most capable LMM, `gpt-4o`, achieves an overall success rate of 36.2%. However, these performances are still far from satisfactory and not yet qualified for direct deployment. Notably, despite its superiority on existing benchmarks, `claude-3.5-sonnet` still falls significantly behind `gpt-4o`. Additionally, we present the first systematic evaluation of `gpt-4o-mini` on agent tasks, which reveals that its performance is considerably inferior to `gpt-4o` but comparable to `claude-3-opus` and `gemini-1.5-pro`.

Trajectory SFT can improve LMM agents. For open LMMs, we find they can rarely follow the system prompt’s instruction without fine-tuning in preliminary trials, resulting in 0% success rates. After training on VAB, open LMMs present significant improvements. The strongest one, `InternVL-2`, even outperforms `gemini-1.0-pro` on all evaluated environments and `claude-3-opus` on CSS agent task. These results suggest that learning from trajectories would be a promising direction for us to build visual foundation agents.

Gaps between top proprietary and open LMMs are huge but likely to be narrowed. Despite the improvement from training, the gap between proprietary and tested open LMMs is much wider than expected. While many of them have claimed competitive performance to `gpt-4-vision-preview` on traditional vision benchmarks such as image captioning, VQA, and so on, their fundamental ability to serve as practical visual foundation agents is far from comparable even after fine-tuning on VAB datasets. It also demonstrates that VAB could serve as an ideal testbed for benchmarking the practical performance of LMMs. With larger backbone LLMs (which are insufficiently tested in this work due to limitations of our computing resources) and more high-quality trajectory data, it is likely that open LMMs will be comparable or even outperform more proprietary LMMs.

5 ANALYSIS

Multimodal agent tasks encompass two critical challenges: *visual grounding* and *planning*. We conduct fine-grained analyses to gain deeper insights into performance in these two aspects and offer valuable perspectives for the future development of visual foundation agents based on LMMs.

5.1 VISUAL GROUNDING ANALYSIS

Visual grounding refers to the ability to associate language concepts with content in visual perception (Fukui et al., 2016; Zheng et al., 2024), which is crucial for LMM-as-Visual-Agent. We look into 3 typical design choices in VAB related to visual grounding to show its current status and challenges.

The use of object labels in embodied environment. Despite the strong image caption and object recognizing ability of LMMs, they do not seem to play well in the context of an embodied agent task. In VAB-OmniGibson, we compare the LMM-as-Visual-Agent performance with and without object labels annotated in the vision input. The result in Figure 4 shows that LMM agents significantly underperform without object labels. It indicates that notwithstanding LMMs’ strong performance on downstream benchmarks, they can still struggle in the same task in the context of LMM-as-Visual-Agent.

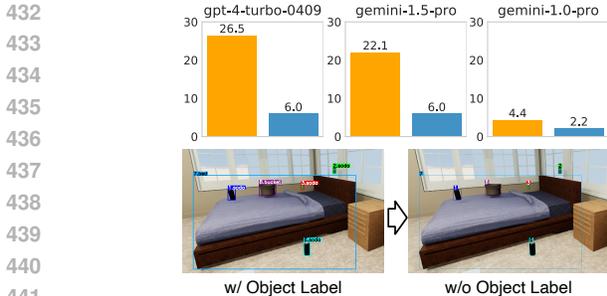


Figure 4: Compare w/ and w/o Object Labels.

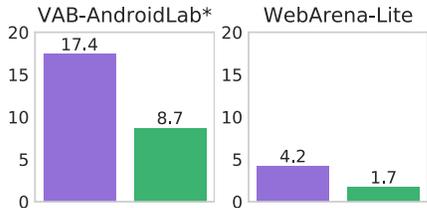


Figure 5: Compare SoM and REC in GUI agent tasks, trained on CogVLM2. VAB-AndroidLab* here is an earlier version different from the one in Table 4.

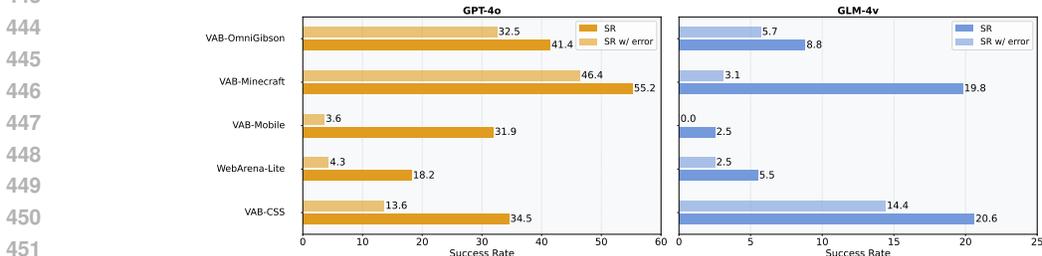


Figure 6: Comparison of overall success rates and success rates when incorrect actions are present in trajectories for various tasks.

The use of Set-of-Marks (SoM) in GUI environment. For GUI tasks, we also augment the image input with SoM by default because it is difficult to elicit accurate bounding box coordinates from the LMM, which is essentially a referring expression comprehension (REC) task (Qiao et al., 2020). With our training trajectories, we can evaluate whether LMMs can effectively perform visual grounding by directly outputting a bounding box without relying on external SoM signals. Specifically, we fine-tune CogVLM2 with and without SoM. The results in Figure 5 show that CogVLM2 struggles to learn to directly output a bounding box, and SoM plays an instrumental role in visual grounding.

Visual difference grounding. Our new frontend design task offers a unique opportunity to examine a specific type of visual grounding: *visual difference grounding*. Unlike traditional visual grounding with a single scene involving associating a language concept to a static region or object in the image, visual grounding in VAB-CSS requires the LMM to properly ground the “layout difference” (Cf. Appendix F.5) to the different areas of two images through comparison. All our current results on VAB-CSS in Table 4 are based on a relatively lenient setting. Instead of requiring the LMM to directly perceive the difference between two screenshots, we provide a language description that explicitly states the adjustments to make to match the two input images (see an example in Appendix F.2).

5.2 PERFORMANCE ON PLANNING

The role of thought in ReAct. ReAct (Yao et al., 2023) is one of the most commonly used frameworks for language agents. The central concept emphasizes the importance of integrating the agent’s reasoning and actions by intertwining the output with both thought and action components. However, in our study, we find that the thought step may not be essential. When using gpt-4o and claude-3.5-sonnet as the backbone of the agents, directly outputting an action field can yield comparable or even superior performance compared to using the ReAct framework (see Table 6).

Recovering from errors during planning. In real-world applications, agents require the error recovery ability to dynamically adjust their actions and plans based on environmental feedback. To understand it in LMMs, we analyze two representative models: gpt-4o, the most powerful model currently available, and glm-4v, a prominent open LMM. Their performance, illustrated in Figure 6, reveals that gpt-4o exhibits robust error recovery across most tasks, with GUI tasks being an exception due to their often irreversible nature. Importantly, we find that incorporating error recovery scenarios in training data significantly enhances the performance of fine-tuned open LMMs, as evidenced by results from VAB-OmniGibson and VAB-CSS (Cf. Appendix B.1 and Appendix F.2 for details about error recovery of training trajectories).

Table 5: LMM performances drop drastically on VAB-CSS when the language description is removed.

	gpt-4o-2024-05-13	gpt-4-turbo-0409	gpt-4-vision-preview
w/ NL	34.5	27.9	29.1
w/o NL	24.2 $\downarrow 10.3\%$	1.9 $\downarrow 26.1\%$	2.4 $\downarrow 26.7\%$

Table 6: ReAct (w/ CoT) has varied (potentially positive or negative) impacts on visual agent tasks.

Model	Prompting	VAB-Minecraft	VAB-AndroidLab	VAB-CSS
gpt-4o	w/ Thought	55.2	30.4	34.5
	w/o Thought	48.3 $\downarrow 6.9\%$	31.9 $\uparrow 1.5\%$	38.2 $\uparrow 3.7\%$
claude-3.5-sonnet	w/ Thought	56.0	29.0	15.8
	w/o Thought	55.2 $\downarrow 0.8\%$	31.1 $\uparrow 2.1\%$	17.6 $\uparrow 1.8\%$

6 RELATED WORK

LMM-as-Visual-Agent. In pre-LMM era, most visual agents are built with task specific training (Shridhar et al., 2020a) and reinforcement learning (Kempka et al., 2016). With the rapid development of LMMs (OpenAI, 2024a; Reid et al., 2024; OpenAI, 2023; Bai et al., 2023; Anthropic, 2024; Team et al., 2023; GLM et al., 2024), the study of LMM-based visual agents begins to thrive. Leveraging the general capabilities of LMMs, these visual agents have the potential to perform complex tasks in various scenarios, including embodied and game tasks (Brohan et al., 2022; Yang et al., 2023a; Driess et al., 2023; Tan et al., 2024), GUI interaction (Zheng et al., 2024; Zhou et al., 2023; Koh et al., 2024; Xie et al., 2024; Kapoor et al., 2024; Yang et al., 2023b), and visual design tasks (Si et al., 2024; Laurençon et al., 2024). However, these complex scenarios pose several challenges for LMM-based visual agents: basic visual understanding and grounding (Zheng et al., 2024; Yue et al., 2023), vision-text information comprehension (Kil et al., 2024), instruction following, and long-term planning ability (Wu et al., 2023; Liu et al., 2023a). Most general-purpose LMMs still lack strong zero-shot capabilities, leading to different application paradigms when deploying LMMs as visual agents. While prompting methods offer great convenience, they may not achieve satisfactory performance in many areas (Zhou et al., 2023; Drouin et al., 2024). Consequently, task-specific training and alignment remain common practices in these applications (Lai et al., 2024). In response, VAB aims to establish a comprehensive benchmark for LMM-based visual agents, covering a wide range of typical applications. In the meantime, VAB seeks to provide an in-depth evaluation of both prompting and training approaches, ultimately fostering the development of LMM visual agents.

Benchmarking LMM-based visual agents. With the rapid development of LMM agents and their impressive performance in various scenarios (Xie et al., 2024; Kapoor et al., 2024; Yang et al., 2023b;a; Si et al., 2024; Mu et al., 2024), it has made the evaluation of LMM agent an urgent problem. In the GUI interaction domain, recent works have proposed static datasets (Deng et al., 2024; Rawles et al., 2024; Sun et al., 2022) and interactive environments (Zhou et al., 2023; Koh et al., 2024; Xie et al., 2024) to evaluate LMM agents in different applications, including web (Zhou et al., 2023; Koh et al., 2024; Deng et al., 2024), mobile phone (Rawles et al., 2024; Sun et al., 2022), and desktop (Xie et al., 2024). In the embodied domain, previous works have proposed various game environments (Guss et al., 2019; Fan et al., 2022) and household environments (Li et al., 2023), but few works have explored benchmarking LMM agents on these environments. Most existing benchmarks are designed for relatively narrow domains and lack a comprehensive evaluation across different applications of LMM agents. Additionally, many benchmarks focus solely on the prompting evaluation of LMM agents. VAB aims to provide a training set for open-source foundation LMMs, offering a new perspective on benchmarking these models and advancing their diverse applicability.

7 CONCLUSION

We present VisualAgentBench (VAB), a comprehensive benchmark for evaluating Large Multimodal Models as visual agents across diverse scenarios. Our testing of 18 LMM models reveals their developing capabilities in this domain. VAB also explores methods for synthesizing visual agent trajectory data, providing insights for future advancements.

REFERENCES

- 540
541
542 Anonymou. Developing and evaluating android agents in a reproducible environment. *Underwork*,
543 2024.
- 544 Anthropic. Introducing the next generation of claude, 2024. URL <https://www.anthropic.com/news/claude-3-family>.
545
- 546 Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang
547 Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities.
548 *arXiv preprint arXiv:2308.12966*, 2023.
- 549
550 Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn,
551 Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics
552 transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.
- 553 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal,
554 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel
555 Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler,
556 Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott
557 Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya
558 Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th
559 International Conference on Neural Information Processing Systems, NIPS’20*, Red Hook, NY,
560 USA, 2020. Curran Associates Inc. ISBN 9781713829546.
- 561 Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. A
562 dataset for interactive vision-language navigation with unknown command feasibility. In *European
563 Conference on Computer Vision*, pp. 312–328. Springer, 2022.
- 564 Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi
565 Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial
566 multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*, 2024.
- 567
568 Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam
569 Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm:
570 Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- 571 Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su.
572 Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing
573 Systems*, 36, 2024.
- 574
575 Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan
576 Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: an embodied multimodal
577 language model. In *Proceedings of the 40th International Conference on Machine Learning*, pp.
578 8469–8488, 2023.
- 579 Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty,
580 Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are
581 web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.
- 582 Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlikar, Yuncong Yang, Haoyi Zhu, Andrew Tang,
583 De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied
584 agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:
585 18343–18362, 2022.
- 586
587 Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach.
588 Multimodal compact bilinear pooling for visual question answering and visual grounding. In
589 *Conference on Empirical Methods in Natural Language Processing*, pp. 457–468. ACL, 2016.
- 590 Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund,
591 Daniel LK Yamins, James J DiCarlo, Josh McDermott, Antonio Torralba, et al. The threedworld
592 transport challenge: A visually guided task-and-motion planning benchmark towards physically
593 realistic embodied ai. In *2022 International conference on robotics and automation (ICRA)*, pp.
8847–8854. IEEE, 2022.

- 594 Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu
595 Feng, Hanlin Zhao, Hanyu Lai, et al. Chatglm: A family of large language models from glm-130b
596 to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024.
- 597 William H. Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codell, Manuela Veloso,
598 and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations, 2019.
- 600 Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazhen Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan
601 Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. *arXiv*
602 *preprint arXiv:2312.08914*, 2023.
- 603 Wenyi Hong, Weihang Wang, Ming Ding, Wenmeng Yu, Qingsong Lv, Yan Wang, Yean Cheng,
604 Shiyu Huang, Junhui Ji, Zhao Xue, et al. Cogvlm2: Visual language models for image and video
605 understanding. *arXiv preprint arXiv:2408.16500*, 2024.
- 607 Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot
608 planners: Extracting actionable knowledge for embodied agents. In *International Conference on*
609 *Machine Learning*, pp. 9118–9147. PMLR, 2022.
- 610 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R
611 Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth*
612 *International Conference on Learning Representations*, 2023.
- 613 Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh,
614 and Ruslan Salakhutdinov. Omniaact: A dataset and benchmark for enabling multimodal generalist
615 autonomous agents for desktop and web. *arXiv preprint arXiv:2402.17553*, 2024.
- 617 Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to
618 objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical*
619 *methods in natural language processing (EMNLP)*, pp. 787–798, 2014.
- 620 Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Viz-
621 doom: A doom-based ai research platform for visual reinforcement learning. *2016 IEEE*
622 *Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, 2016. URL <https://api.semanticscholar.org/CorpusID:430714>.
- 623 Jihyung Kil, Chan Hee Song, Boyuan Zheng, Xiang Deng, Yu Su, and Wei-Lun Chao. Dual-view
624 visual contextualization for web navigation. In *Proceedings of the IEEE/CVF Conference on*
625 *Computer Vision and Pattern Recognition (CVPR)*, pp. 14445–14454, June 2024.
- 626 Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham
627 Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating
628 multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.
- 629 Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen
630 Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autoweblm: Bootstrap and reinforce a large
631 language model-based web navigating agent, 2024.
- 632 Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into
633 html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.
- 634 Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-
635 Martín, Chen Wang, Gabriel Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A
636 benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference*
637 *on Robot Learning*, pp. 80–93. PMLR, 2023.
- 638 Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative
639 model for text-to-behavior in minecraft. *Advances in Neural Information Processing Systems*, 36,
640 2024.
- 641 Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement
642 learning on web interfaces using workflow-guided exploration. In *International Conference on*
643 *Learning Representations*, 2018.

- 648 Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction
649 tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*,
650 pp. 26296–26306, 2024a.
- 651 Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee.
652 Llava-next: Improved reasoning, ocr, and world knowledge, January 2024b. URL <https://llava-vl.github.io/blog/2024-01-30-llava-next/>.
- 653
654
- 655 Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in*
656 *neural information processing systems*, 36, 2024c.
- 657
- 658 Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding,
659 Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint*
660 *arXiv:2308.03688*, 2023a.
- 661 Yuliang Liu, Zhang Li, Hongliang Li, Wenwen Yu, Mingxin Huang, Dezhi Peng, Mingyu Liu,
662 Mingrui Chen, Chunyuan Li, Lianwen Jin, et al. On the hidden mystery of ocr in large multimodal
663 models. *arXiv preprint arXiv:2305.07895*, 2023b.
- 664
- 665 Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord,
666 Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for
667 science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521,
668 2022.
- 669 Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng,
670 Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning
671 of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.
- 672
- 673 Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data.
674 *arXiv preprint arXiv:2005.07648*, 2020.
- 675
- 676 Pattie Maes. Agents that reduce work and information overload. *Commun. ACM*, 37:30–40, 1994.
- 677
- 678 Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng
679 Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of
680 thought. *Advances in Neural Information Processing Systems*, 36, 2024.
- 681
- 682 Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher
683 Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted
684 question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
- 685
- 686 OpenAI. New models and developer products announced
687 at devday, 2023. URL [https://openai.com/index/
new-models-and-developer-products-announced-at-devday/](https://openai.com/index/new-models-and-developer-products-announced-at-devday/).
- 688
- 689 OpenAI. Hello gpt-4o, 2024a. URL <https://openai.com/index/hello-gpt-4o/>.
- 690
- 691 OpenAI. Gpt-4o mini: advancing cost-efficient intelligence, 2024b. URL [https://openai.
com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/](https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/).
- 692
- 693 Yanyuan Qiao, Chaorui Deng, and Qi Wu. Referring expression comprehension: A survey of methods
694 and datasets. *IEEE Transactions on Multimedia*, 23:4426–4440, 2020.
- 695
- 696 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the
697 wild: A large-scale dataset for android device control. *Advances in Neural Information Processing*
698 *Systems*, 36, 2024.
- 699
- 700 Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste
701 Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini
1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint*
arXiv:2403.05530, 2024.
- John R. Searle. Speech acts: An essay in the philosophy of language. *Language*, 46:217, 1970.

- 702 Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An
703 open-domain platform for web-based agents. In *International Conference on Machine Learning*,
704 pp. 3135–3144. PMLR, 2017.
- 705 Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi,
706 Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions
707 for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern
708 recognition*, pp. 10740–10749, 2020a.
- 709 Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew
710 Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. In
711 *International Conference on Learning Representations*, 2020b.
- 712 Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are
713 we from automating front-end engineering? *arXiv preprint arXiv:2403.03163*, 2024.
- 714 Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and
715 Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference
716 on computer vision and pattern recognition*, pp. 8317–8326, 2019.
- 717 Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su.
718 Llm-planner: Few-shot grounded planning for embodied agents with large language models. In
719 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2998–3009, 2023.
- 720 Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott
721 Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for
722 everyday household activities in virtual, interactive, and ecological environments. In *Conference
723 on robot learning*, pp. 477–490. PMLR, 2022.
- 724 Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. Meta-gui: Towards
725 multi-modal conversational agents on mobile gui. In *Proceedings of the 2022 Conference on
726 Empirical Methods in Natural Language Processing*, pp. 6699–6712, 2022.
- 727 Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng
728 Yue, Jiechuan Jiang, Yewen Li, Ruyi An, Molei Qin, Chuqiao Zong, Longtao Zheng, Yujie Wu,
729 Xiaoqiang Chai, Yifei Bi, Tianbao Xie, Pengjie Gu, Xiyun Li, Ceyao Zhang, Long Tian, Chaojie
730 Wang, Xinrun Wang, Börje F. Karlsson, Bo An, Shuicheng Yan, and Zongqing Lu. Cradle: Em-
731 powering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*,
732 2024.
- 733 Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu
734 Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable
735 multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- 736 Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay
737 Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation
738 and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- 739 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and
740 Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv
741 preprint arXiv:2305.16291*, 2023a.
- 742 Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao
743 Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv
744 preprint arXiv:2401.16158*, 2024a.
- 745 Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu,
746 Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model’s perception of the
747 world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024b.
- 748 Weihang Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang,
749 Lei Zhao, Xixuan Song, et al. Cogvlm: Visual expert for pretrained language models. *arXiv
750 preprint arXiv:2311.03079*, 2023b.

- 756 Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from
757 error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612,
758 2004.
- 759 Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng
760 He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-
761 augmented multimodal language models. *arXiv preprint arXiv:2311.05997*, 2023c.
- 762 Michael Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge
763 engineering review*, 10(2):115–152, 1995.
- 764 Yue Wu, Xuan Tang, Tom Mitchell, and Yuanzhi Li. Smartplay: A benchmark for llms as intelligent
765 agents. In *The Twelfth International Conference on Learning Representations*, 2023.
- 766 Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing
767 Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal
768 agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*,
769 2024.
- 770 Jingkang Yang, Yuhao Dong, Shuai Liu, Bo Li, Ziyue Wang, Chencheng Jiang, Haoran Tan, Jiamu
771 Kang, Yuanhan Zhang, Kaiyang Zhou, et al. Octopus: Embodied vision-language programmer
772 from environmental feedback. *arXiv preprint arXiv:2310.08588*, 2023a.
- 773 Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent:
774 Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*, 2023b.
- 775 Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable
776 real-world web interaction with grounded language agents. *Advances in Neural Information
777 Processing Systems*, 35:20744–20757, 2022.
- 778 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan
779 Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International
780 Conference on Learning Representations*, 2023.
- 781 Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens,
782 Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal
783 understanding and reasoning benchmark for expert agi. *arXiv preprint arXiv:2311.16502*, 2023.
- 784 Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu,
785 Wendi Zheng, Xiao Xia, et al. Glm-130b: An open bilingual pre-trained model. *arXiv preprint
786 arXiv:2210.02414*, 2022.
- 787 Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning:
788 Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.
- 789 Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web
790 agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024. URL
791 <https://openreview.net/forum?id=piecKJ2D1B>.
- 792 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
793 Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building
794 autonomous agents. In *The Twelfth International Conference on Learning Representations*, 2023.
- 795 Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li,
796 Lewei Lu, Xiaogang Wang, et al. Ghost in the minecraft: Generally capable agents for open-world
797 environments via large language models with text-based knowledge and memory. *arXiv preprint
798 arXiv:2305.17144*, 2023.
- 799
800
801
802
803
804
805
806
807
808
809

Part I

Appendix

Table of Contents

A Overview	17
A.1 Design Features of VAB	17
A.2 Details on Use of Visual Information in Each Environment	17
A.3 Details on Task Instance Collection	18
A.4 Details on Training Trajectory Collection	18
B VAB-OmniGibson	19
B.1 Detailed Description	19
B.2 Actions	20
B.3 Program-based Solver for Training Trajectory Collection	20
B.4 Prompt Example	20
C VAB-Minecraft	25
C.1 Detailed Description	25
C.2 Actions	25
C.3 Prompt Example	26
D VAB-AndroidLab	30
D.1 Detailed Description	30
D.2 Actions	30
D.3 Metrics	31
D.4 Prompt Example	31
E WebArena-Lite	32
E.1 Detailed Description	32
E.2 Actions	33
E.3 Metrics	34
E.4 Task Amendment	34
E.5 Prompt Example	36
F VAB-CSS	37
F.1 Detailed Description	37
F.2 Data Collection	38
F.3 Actions	38
F.4 Metrics	38
F.5 Prompt Example	39
G Case Studies	41
G.1 VAB-OmniGibson Cases	41
G.2 VAB-Minecraft Cases	44
G.3 VAB-AndroidLab Cases	48
G.4 WebArena-Lite Cases	50
G.5 VAB-CSS Cases	52
G.6 Error Recovery Cases	54

864 A OVERVIEW

865 A.1 DESIGN FEATURES OF VAB

866 Given that LMMs are still evolving rapidly, we adhere to several principles in our design of VAB to
867 accommodate the current capabilities and limitations of LMMs.

- 871 • **Vision-Centric:** VAB agent tasks are designed to primarily rely on visual inputs to solve problems.
872 While additional text inputs could be beneficial, VAB aims to evaluate how LMMs perform when
873 perceiving the environment as humans do in agent tasks. For example, while HTML is shown
874 useful for Web GUI Agent (Zhou et al., 2023; Deng et al., 2024), humans typically browse the
875 internet from screens without reading HTMLs.
- 876 • **High-Level Decision Making:** VAB focuses on evaluating LMMs’ high-level decision-making
877 abilities. Compared to prior smaller visual-language models that specifically target low-level
878 policies (Lynch & Sermanet, 2020; Brohan et al., 2022; Lifshitz et al., 2024), LMMs excel at high-
879 level planning and interacting (Driess et al., 2023) in text response thanks to their commonsense,
880 knowledge, and flexible instruction following with mere prompting. Therefore, in VAB, we
881 simplify the low-level control by providing convenient action interfaces, and ask tested LMMs to
882 concentrate on delivering high-level decision sequences in text.
- 883 • **Interactive Evaluation:** Evaluating LLMs or LMMs on real-world agent tasks is challenging, as
884 task goals can be achieved by various means. As a result, it becomes a mainstream practice to
885 evaluate in an interactive manner (Liu et al., 2023a; Zhou et al., 2023; Jimenez et al., 2023; Xie
886 et al., 2024). VAB also adheres to this principle.
- 887 • **Trajectories for Behavior Cloning:** Many previous execution-based agent benchmarks for
888 LLMs and LMMs, despite being realistic and challenging, often fail to provide effective training
889 sets for the community to use for improvement. LLMs and LMMs need behavior cloning training
890 on trajectories for better performance (Nakano et al., 2021; Zeng et al., 2023; Lai et al., 2024).
891 However, creating such datasets consisting of valid instructions, trajectories, and reward functions
892 is costly and requires annotators’ good understanding of the environment. In response to the
893 challenge, for each VAB environment we endeavor to deliver instructions created with a hybrid
894 set of strategies (Cf. Section 3.2). Experiments show that our constructed training sets can
effectively improve the performance of open LMMs on VAB.

895 Note that as the field advances, some of the above principles may become obsolete and irrelevant.
896 We will continuously update VAB to accommodate the progress of LMMs.

897 A.2 DETAILS ON USE OF VISUAL INFORMATION IN EACH ENVIRONMENT

898 Vision-centric design is a crucial feature for VAB with regard to planning and grounding. Here we
899 will give a brief look at the specific use of vision from both perspectives.

- 903 • **VAB-OmniGibson:** The agent perceives the simulated environment through egocentric vision.
904 It must interpret visual information to identify the affordances of objects within its view and
905 understand their spatial relationships (e.g., whether an object is reachable).
- 906 • **VAB-Minecraft:** Similar to VAB-OmniGibson, the agent must interpret the current scene in the
907 game environment to determine its next action. In addition, VAB-Minecraft includes moving
908 elements such as animals and monsters, which places greater demands on the agent’s visual
909 understanding to complete tasks.
- 910 • **VAB-AndroidLab:** The agent operates on the graphical interface of an Android system to
911 complete tasks without access to system APIs and relies solely on visual inputs. At each step, it
912 analyzes the current screenshot to predict an action—identifying interactive elements such as app
913 icons or buttons based solely on the screenshot. Executing the action generates a new screenshot
914 for the next iteration.
- 915 • **WebArena-Lite:** Similar to VAB-AndroidLab, WebArena-Lite is also a GUI-based environment.
916 In the original paper of WebArena (Zhou et al., 2023), they use HTML/Accessibility Tree as the
917 input, whereas in VAB, we focus on visual inputs and mainly focus on screenshots to predict
actions at each step.

- **VAB-CSS**: The agent must carefully adjust the CSS style file until the rendering matches the given screenshot. It needs to perceive fine-grained visual differences between two screenshots, which can often be minimal. This makes VAB-CSS a highly vision-intensive task.

A.3 DETAILS ON TASK INSTANCE COLLECTION

For VAB-OmniGibson, a prototype is a general household activity, such as recycling office papers. We source these prototypes either by sampling from Behavior-1K or by annotating them ourselves. Instantiating a prototype involves grounding it in a specific scene (e.g., specific rooms with office papers and recycling bins) generated in OmniGibson. To increase task diversity, we instantiate each prototype with multiple random scenes and various initializations of object positions in the room. In total, we collect 992 instances using 89 prototypes. We sample 181 out of them as our test set.

For VAB-Minecraft, we target high-level task prototypes related to object collecting and then instantiate them with game configurations using different world seeds or spawn points. We manually check to ensure that each high-level goal is achievable within its configuration. In total, we collect 628 task instances using high-quality prototypes defined by us, with 116 instances designated as the test set. Additionally, we sample 132 task prototypes from JARVIS-1, resulting in 596 task instances that could be leveraged to collect our training trajectories later.

For VAB-AndroidLab, we first select 8 typical Android applications, from system services to third-party applications (e.g., Maps, Music, etc.) that could be evaluated offline. We come up with 119 test instructions for them and prepare valid groundings in the AVD snapshot (e.g., an MP3 file to play in the Music APP). For the training task construction, we filter 18 commonly used APPs and summarize their major functions to around 70 task prototypes.

For WebArena-Lite, we filtered and cleaned 165 test samples from the original WebArena dataset and collected new task instances for web applications to use in training trajectory collection. Specifically, we summarize each website’s basic functions and valid items for synthetic queries, created 40 task prototypes, and fill them with valid and invalid items (e.g., `product categories`, `prices`) to generate specific instructions, resulting in 1,186 training task instances.

For VAB-CSS, a task prototype simply corresponds to one possible corruption of a CSS rule such as adding or altering a CSS property. To instantiate a task for a specific website, we randomly select a corruption that results in noticeable visual changes, determined by an SSIM (Wang et al., 2004) score below 0.8.³ In addition, we manually annotate each instance with a natural language description of the difference between the two images as an additional clue to the agent. In total, we collect 1,210 instances and use 165 to form the test set.

A.4 DETAILS ON TRAINING TRAJECTORY COLLECTION

For VAB-OmniGibson, we adopt the program-based solvers focusing on the cost and adaptability. OmniGibson has no friendly interface for humans to operate on, and requires high-end laptops with GPUs supporting ray tracing and large main memory (> 10 GB) to run. Thus it is unlikely for us to find a large number of qualified annotators to label for OmniGibson. LMM agent bootstrapping is fine but uneconomical, as the task usually takes more steps than others (i.e., up to 100). Program-based solvers, instead, are suitable for collecting massive high-quality trajectories in OmniGibson.

For VAB-Minecraft, we adopt LMM agent bootstrapping considering adaptability. Minecraft requires some flexible explorations (as environments are generated randomly), which is beyond the scope of program-based solvers. Humans need to be well-trained for some time on playing Minecraft before becoming qualified annotators. Since previous work has explored the usage of memory augmentation (Wang et al., 2023c) for improving LMM agents in Minecraft, it becomes practical to leverage the bootstrapping strategy by LMM APIs such as `gpt-4o` for creating training trajectories.

For VAB-AndroidLab, we primarily adopt human demonstrations, accompanied with some LMM Agent Bootstrapping considering the versatility and flexibility. As android XMLs are less legible and operable than HTMLs on web with existing automation tools, program-based solvers are not applicable. Additionally, for many apps require login and internet connection, human demonstration

³This is an empirical choice based on our own experience.

972 is the best solution. LMM agent bootstrapping is employed in some offline APPs such as system
973 settings to enhance trajectory flexibility.

974 For VAB-WebArena-Lite, we adopt program-based solvers due to cost and adaptability. On the
975 one hand, there have been a mature web automation tool `Playwright` that supports Python. On
976 the other hand, although WebArena (Zhou et al., 2023) is adopting some mirror websites for their
977 real-world counterparts, their interfaces could be vastly different (e.g., OpenStreetMap in WebArena
978 vs. Google Maps in real-world). Consequently, human annotators struggle to label demonstrations
979 on these websites efficiently in our preliminary trials. For LMM agents, they tend to perform too
980 poorly under mere prompting on WebArena (with success rate less than 20%) for efficient trajectory
981 construction.

982 For VAB-CSS, we adopt LMM agent bootstrapping, mostly owing to concerns on flexibility. A
983 critical challenge for the agent in debugging CSS styles is to iteratively adjust the CSS rules through
984 a trial and error process, which can be flexibly achieved using the LMM agent bootstrapping scheme.
985 In particular, we first use `gpt-4o` to collect trajectories that finally resolve the CSS issue. However,
986 `gpt-4o` can only achieve a success rate lower than 40%. To collect additional trajectories, we hint
987 the agent with the target CSS rule to edit, after 5 steps of trials, on tasks where the agent initially
988 fails.

990 B VAB-OMNIGIBSON

993 In this section, we provide additional details about VAB-OmniGibson that are not covered in the
994 main paper due to space limitations.

996 B.1 DETAILED DESCRIPTION

998 Current household datasets or benchmarks are not originally designed for LMMs, making them
999 less suitable for evaluating today’s LMMs. Behavior-1K (Li et al., 2023) offers an action space
1000 focused on low-level physical control over the robot (e.g., joint angles), while Alfred (Shridhar et al.,
1001 2020a) requires actions to output masks on images, which may not be practical for most LMMs.
1002 The ThreeDWorld Transport Challenge (Gan et al., 2022) provides high-level action APIs, but the
1003 simulator environment is less realistic and the tasks may not fully challenge LMMs. The recent
1004 work Octopus (Yang et al., 2023a) sets up household tasks for LMMs in the OmniGibson simulator.
1005 However, in this setting, vision input is less critical as the observed objects are also listed in text input
1006 for LMMs.

1007 In order to set up a realistic and challenging benchmark for testing LMMs’ embodied planning ability,
1008 we select the recent household simulator OmniGibson (Li et al., 2023) as the interactive environment,
1009 and build a pipeline for LMM to serve as a high-level planner on everyday household activities.
1010 An example of the task is shown in Fig. 7: The ego-centric image with annotated bounding boxes,
1011 high-level activity instruction and environment feedback are fed into the LMM, and it is tasked with
1012 reasoning over the current progress to decide on the next low-level action. It must interact with
1013 objects using the corresponding tags attached to the bounding boxes.

1014 **Test Set.** We select 45 activity instances from Behavior-1K (Li et al., 2023), and manually adapt
1015 some of them to ensure these activities are solvable within our provided action space and suitable for
1016 evaluating current LMMs’ embodied planning ability. We instantiate each activity in several scenes,
1017 resulting in a total of 181 test task instances. All the activity instructions are manually annotated by
1018 us.

1019 **Training Set.** We provide a set of successful trajectories using both program-based solving and
1020 LMM bootstrapping. We newly design 47 activities, each instantiated in several different scenes with
1021 various initializations of object positions, resulting in a total of 901 task instances. To solve these
1022 tasks, we develop a program-based solver that decomposes the long-horizon activities into subtasks
1023 and solves them sequentially. Running the program-based solver on the 901 training task instances
1024 yields 785 successful trajectories. Then we manually add a type of error recovery process (agent fails
1025 to place an object into a closed container, and then opens the container) into these trajectories, aiming
to enhance LMMs’ capability to rectify errors. Additionally, we select 464 training instances and

1026 utilize `gpt-4-vision-preview` to bootstrap 87 successful trajectories, resulting in a total of
 1027 872 training trajectories.

1028 **Metrics.** We adopt task success rate as the metric of VAB-OmniGibson. In Behavior-1K (Li
 1029 et al., 2023), each activity is defined in the form of BEHAVIOR Domain Definition Language
 1030 (BDDL) (Srivastava et al., 2022), which describes the concrete initial and goal conditions of a specific
 1031 activity. Only when all the goal conditions are met within the limit of 100 turns, the task is judged as
 1032 successfully completed.

1034 B.2 ACTIONS

1036 In VAB-OmniGibson, we provide the LMM agent with 20 low-level actions to interact with objects
 1037 and navigate the household environment. The actions marked with an asterisk (*) are adapted from
 1038 OmniGibson (Li et al., 2023), while the others are newly defined and implemented by us. With these
 1039 provided actions, the LMM agent is possible to solve all the testing instances.

- 1040 • **grasp**: Grasp a specific object into the robot’s hand.
- 1041 • **move**: Move towards a specific object.
- 1042 • **move_to_room**: Move to a specific room in the house.
- 1043 • **turn_left**: Turn the robot left 90 degrees.
- 1044 • **turn_right**: Turn the robot right 90 degrees.
- 1045 • **raise_camera**: Raise the camera of the robot to see higher objects.
- 1046 • **lower_camera**: Lower the camera of the robot to see lower objects.
- 1047 • **put_inside**: Place the object from the robot’s hand inside another object.
- 1048 • **put_on_top**: Place the object from the robot’s hand on top of another object.
- 1049 • **put_under**: Place the object from the robot’s hand under another object.
- 1050 • **put_next_to**: Place the object from the robot’s hand next to another object.
- 1051 • **get_fridge_view**: Obtain the view inside a nearby fridge.
- 1052 • **cook***: Cook a specific object.
- 1053 • **burn***: Burn a specific object.
- 1054 • **freeze***: Freeze a specific object.
- 1055 • **heat***: Heat a specific object.
- 1056 • **open***: Open a specific object.
- 1057 • **close***: Close a specific object.
- 1058 • **toggle_on***: Turn on a specific object.
- 1059 • **toggle_off***: Turn off a specific object.

1063 B.3 PROGRAM-BASED SOLVER FOR TRAINING TRAJECTORY COLLECTION

1065 **BDDL task goals.** Among activities of VAB-OmniGibson, each of the BDDL task goal can be
 1066 decomposed into a sequence of subgoals (e.g., a specific door should be open, or a specific bottle
 1067 should be on a specific countertop). All subgoals can be categorized into 2 types: identifying the
 1068 state of a specific object, or the positional relationship between two objects.

1069 **Method of program-based solver.** To achieve the BDDL task goal of a VAB-OmniGibson activity,
 1070 the program-based solver need to sequentially fulfill all the subgoals. For the first type of sub-
 1071 goal, the program-based solver can navigate (`move_to_room`, `move`, `turn_left`, `turn_right`,
 1072 `raise_camera`, `lower_camera`, `get_fridge_view`) to find the specific object; and then
 1073 move towards it (`move`) and change its state (`cook`, `burn`, `freeze`, `heat`, `open`, `close`,
 1074 `toggle_on`, `toggle_off`). For positional relationships, the solver should find and approach an ob-
 1075 ject, grasp it (`grasp`), move to the other object, and finally complete the subgoal with `put_inside`,
 1076 `put_on_top`, `put_under` or `put_next_to`.

1078 B.4 PROMPT EXAMPLE

1079 The system message that describes the detailed task information to the agent is shown as follows:

1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133

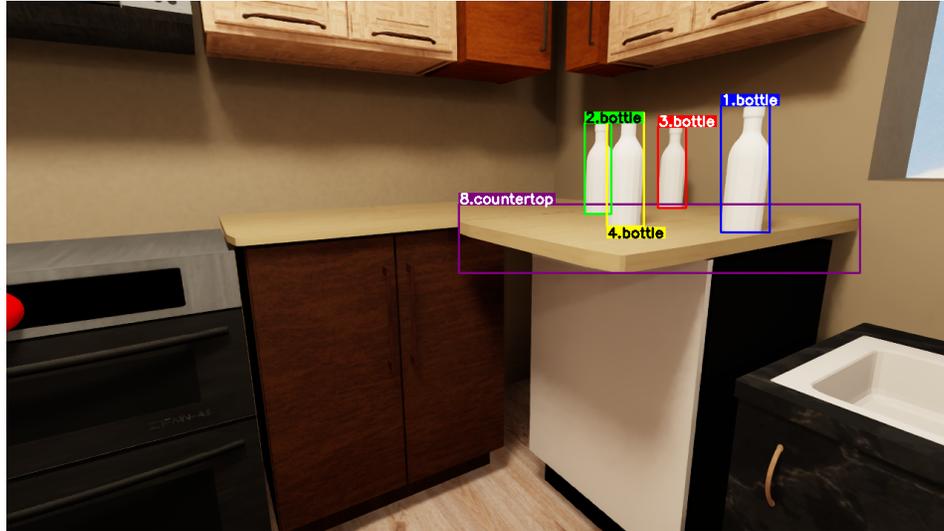


Figure 7: This is an example task of VAB-OmniGibson. The task asks the agent to bring all 4 bottles from the countertop into the fridge. The agent should grasp one bottle, navigate to find the fridge, open the fridge, put the grasped bottle into the fridge, and then repeat this process for the remaining bottles.

```
# Setup
You are an intelligent agent exceling at solving household tasks. You are
  in a household environment given a task to finish.
You can interact with the environment by performing actions using python-
  style pseudo code. For each turn, please call exactly one predefined
  action.

# Valid Actions
## Predefined Action List:
'''
def grasp(obj):
    '''Grasp the object in your hand.
    Args:
        :param obj: the digital identifier of the object to grasp.
    Returns:
        A string message of the environment feedback.
    '''

def move(obj):
    '''Move yourself towards the object.
    Args:
        :param obj: the digital identifier of the object to move towards.
    Returns:
        A string message of the environment feedback.
    '''

def move_to_room(room):
    '''Move yourself to a random position in the room.
    Args:
        :param room: the name of the room to move to.
    Returns:
        A string message of the environment feedback.
    '''

def turn_left():
    '''Turn the robot left 90 degrees.
    Returns:
        A string message of the environment feedback.
    '''

def turn_right():
```

```
1134     '''Turn the robot right 90 degrees.
1135     Returns:
1136         A string message of the environment feedback.
1137     '''
1138 def raise_camera():
1139     '''Raise the camera to see objects that are higher.
1140     Returns:
1141         A string message of the environment feedback.
1142     '''
1143 def lower_camera():
1144     '''Lower the camera to see objects that are lower.
1145     Returns:
1146         A string message of the environment feedback.
1147     '''
1148 def put_inside(obj1, obj2):
1149     '''Put obj1 within your hand inside obj2. If obj2 is openable, make
1150     sure it is open before putting obj1 inside.
1151     Args:
1152         :param obj1: the digital identifier of the object to put inside.
1153         :param obj2: the digital identifier of the object to put inside
1154         of.
1155     Returns:
1156         A string message of the environment feedback.
1157     '''
1158 def put_on_top(obj1, obj2):
1159     '''Put obj1 within your hand to the top of obj2.
1160     Args:
1161         :param obj1: the digital identifier of the object to put on top.
1162         :param obj2: the digital identifier of the object to put on top
1163         of.
1164     Returns:
1165         A string message of the environment feedback.
1166     '''
1167 def put_under(obj1, obj2):
1168     '''Put obj1 within your hand to the bottom of obj2.
1169     Args:
1170         :param obj1: the digital identifier of the object in your hand.
1171         :param obj2: the digital identifier of the object to put obj1
1172         under.
1173     Returns:
1174         A string message of the environment feedback.
1175     '''
1176 def put_next_to(obj1, obj2):
1177     '''Put obj1 within your hand next to obj2.
1178     Args:
1179         :param obj1: the digital identifier of the object in your hand.
1180         :param obj2: the digital identifier of the object to put obj1
1181         next to.
1182     Returns:
1183         A string message of the environment feedback.
1184     '''
1185 def get_fridge_view():
1186     '''Get the image captured by a camera in the fridge. This function is
1187     only valid when you are near a fridge and the fridge is open.
1188     Returns:
1189         A string message of the environment feedback.
1190     '''
1191 def cook(obj):
1192     '''Cook the object.
1193     Args:
1194         :param obj: the digital identifier of the object to cook.
1195     Returns:
1196         A string message of the environment feedback.
1197     '''
1198 def burn(obj):
```

```

1188     '''Burn the object.
1189     Args:
1190         :param obj: the digital identifier of the object to burn.
1191     Returns:
1192         A string message of the environment feedback.
1193     '''
1194 def freeze(obj):
1195     '''Freeze the object.
1196     Args:
1197         :param obj: the digital identifier of the object to freeze.
1198     Returns:
1199         A string message of the environment feedback.
1200     '''
1201 def heat(obj):
1202     '''Heat the object.
1203     Args:
1204         :param obj: the digital identifier of the object to heat.
1205     Returns:
1206         A string message of the environment feedback.
1207     '''
1208 def open(obj):
1209     '''Open the object.
1210     Args:
1211         :param obj: the digital identifier of the object to open.
1212     Returns:
1213         A string message of the environment feedback.
1214     '''
1215 def close(obj):
1216     '''Close the object.
1217     Args:
1218         :param obj: the digital identifier of the object to close.
1219     Returns:
1220         A string message of the environment feedback.
1221     '''
1222 def toggle_on(obj):
1223     '''Toggle on the object.
1224     Args:
1225         :param obj: the digital identifier of the object to toggle on.
1226     Returns:
1227         A string message of the environment feedback.
1228     '''
1229 def toggle_off(obj):
1230     '''Toggle off the object.
1231     Args:
1232         :param obj: the digital identifier of the object to toggle off.
1233     Returns:
1234         None. The environment will check whether the task is completed
1235         and check your score.
1236     '''
1237 '''
1238 ## Reminder
1239 1. You can only hold one object at a time.
1240 2. When moving to a new position, you can always turn left, turn right,
1241    raise camera or lower camera to see around before making a decision.
1242 3. You can only interact with objects within your reach; if not, first
1243    try moving towards it or something close to it.
1244 4. You can only interact with objects that are visible to you (annotated
1245    with a bounding box in the image); if it's not visible, try to move

```

```

1242     inside the room or other rooms and look around to find it. You can
1243     open refrigerators or other enclosures to see inside them.
1244 5. You can interact with objects that are very close to you, such as
1245     those you've just moved towards, even if you don't see them currently
1246     .
1247 6. When you are out of the room and see nothing useful, try moving to a
1248     room.
1249 7. You can always move to something in the same room with you, if you
1250     have seen it before, even though you cannot see it now. So when you
1251     are in a new room, try to move around and see around to record more
1252     objects in your observation so that you can move to them flexibly
1253     afterwards.
1254 8. Don't repeat the failed action in the next round. Try to understand
1255     what went wrong and make a different decision.
1256 9. If you can't complete the task, you can do as much as you can and call
1257     `done()` to finish the task.
1258
1259 # Input
1260 For each dialog, you will be given the following information at the
1261 beginning.
1262 1. Task Goal: The task is finished only when these conditions are met.
1263 2. Reachable Rooms: Rooms you can move to. Please refer to them with
1264     their names provided here.
1265 For each turn, you will be given the following information.
1266 1. Action Feedback: Environment feedback of the last action.
1267 2. At Hand Object: The object you are currently holding.
1268 3. Current Room: The room you are currently in.
1269 4. Vision Input: the image you see from your perspective (or inside the
1270     fridge). All task-related objects appear in your view will be
1271     annotated with bounding boxes and unique identifiers. Please
1272     reference these objects using the digital identifier provided here.
1273     Note that if the object is not annotated with a bounding box, the
1274     object can't be interacted with.
1275
1276 # Output
1277 Now, given these information, you need to think and call the action
1278 needed to proceed with the task. Your response should include 3 parts
1279 in the following format in each turn:
1280 OBSERVATION: <What you observe in the image> Note that the Vision Input
1281 image won't be kept in the dialog, so make sure you capture all the
1282 key information (eg, the identifier of the object you see) here for
1283 future use.
1284 THOUGHT: <Your step-by-step thoughts>
1285 ACTION: <The action code> Note that only one function is allowed in each
1286 dialog turn! Only one line of code is allowed in each dialog turn! If
1287 your output contains multiple actions or multiple turns of actions,
1288 only the first one will be executed!
1289
1290
1291 Here is a concrete example of the task input shown in Fig. 7, where the image is enclosed within
1292 "{}":
1293
1294 Your task is: There are 4 beer bottles on a countertop in the kitchen.
1295 Please put all of them into the fridge.
1296 The reachable rooms during the task are: corridor_0, dining_room_0,
1297 kitchen_0, living_room_0, pantry_room_0, storage_room_0.
1298 Action Feedback: None actions before.
1299 At Hand Object: None.
1300 Current Room: kitchen_0.
1301 Vision Input: {{Image}}

```

1296 C VAB-MINECRAFT

1297
1298 In this section, we provide additional details about VAB-Minecraft that are not covered in the main
1299 paper due to space limitations.

1300 The game Minecraft has become a popular open-world environment for developing generalist embod-
1301 ied agents (Fan et al., 2022; Lifshitz et al., 2024) due to its diverse tasks (e.g., survival, harvest, craft,
1302 combat, and creative tasks), varied environments, and interactive mobs, necessitating generalized
1303 agent abilities. Recent pioneering works (Zhu et al., 2023; Wang et al., 2023a;c) have integrated
1304 LLMs into embodied agents to tackle Minecraft tasks. However, these efforts did not focus on a
1305 standardized pipeline for evaluating LLMs’ planning abilities. So we adapt the JARVIS-1 (Wang
1306 et al., 2023c) pipeline to assess LLMs’ high-level action planning abilities in item-obtaining tasks.
1307

1308 C.1 DETAILED DESCRIPTION

1309
1310 In VAB-Minecraft, we adapt the action space of JARVIS-1 (Wang et al., 2023c) to develop a pipeline
1311 for LMM, enabling it to function as a high-level embodied planner. Additionally, we also use
1312 item-obtaining tasks to benchmark LLMs’ high-level embodied planning abilities. These tasks are
1313 comprehensive, requiring task analysis and decomposition, as well as ingredient collection. Each
1314 aspect respectively challenges an LMM agent’s planning skills and its ability to interact with the
1315 environment.

1316 **Test Set.** We manually annotate 116 different tasks, each with a specific target item and a cor-
1317 responding initial configuration to ensure the task is solvable. For example, Fig. 8 illustrates the
1318 VAB-Minecraft task of obtaining a cake, where we have set up the initial configuration of necessary
1319 surrounding resources and inventory items. These 116 test tasks span the Minecraft tech tree, covering
1320 items across 6 material levels (wood, stone, iron, gold, diamond and netherite) and involving a diverse
1321 range of raw ingredients from various resources: 11 types of plants, 4 types of animals, and 6 types
1322 of hostile mobs. This diversity greatly challenges the agent’s ability to interact with the environment.

1323 **Training Set.** Training trajectories are collected using bootstrapping from two sources: pure
1324 `gpt-4-turbo` bootstrapping on newly designed tasks, and `gpt-4o` bootstrapping with JARVIS-1
1325 memory on tasks from JARVIS-1. For the first type, we design 40 new tasks instantiated in different
1326 world seeds or spawn points, resulting in 512 task instances, and `gpt-4-turbo` bootstraps 176
1327 successful trajectories. For the second type, we use 132 tasks from JARVIS-1, set up in 596 task
1328 instances, and run with memory using `gpt-4o`, resulting in 206 successful trajectories. In total, we
1329 gain 382 successful trajectories.

1330 **Metrics.** We adopt success rate as the evaluation metric in VAB-Minecraft. For a specific item-
1331 obtaining task, if the agent can obtain the specific item within the limitation of 100 rounds, the task is
1332 regarded as successfully completed.

1333 C.2 ACTIONS

1334
1335 In VAB-Minecraft, we provide 6 types of actions for the LMM agent. 4 actions, marked with an
1336 asterisk (*), are adapted from the JARVIS-1 pipeline (Wang et al., 2023c), while the remaining 2 are
1337 newly implemented by us to enhance the LMM agent’s capability to solve a wider range of tasks.

- 1338 • **craft***: Utilize the inventory or crafting table to craft a specific item.
- 1339 • **smelt***: Utilize a furnace to smelt a specific item.
- 1340 • **equip***: Equip a specific item in the agent’s hand.
- 1341 • **teleport_to_spawn**: Teleport the agent back to the spawn point. As we will prepare necessary
1342 ingredients around the agent’s spawn point, this action enables the agent to conveniently collect
1343 these ingredients. This function is also helpful if the agent stuck somewhere (e.g., underground).
- 1344 • **look_up**: Look up the crafting/smelting information about a specific item. This reference guides
1345 the agent to make a plan on how to accomplish the task.
- 1346 • **execute***: Use natural language prompt to instruct a low-level minecraft planner, Steve-1 (Lifshitz
1347 et al., 2024). With proper prompting, it can solve most basic tasks, like mining common blocks,
1348 collecting plants, interacting with animals and hostile mobs, and navigating between different
1349 biomes.



1368 Figure 8: This is an example of VAB-Minecraft task. This task asks the agent to obtain a cake in the
1369 inventory. Initially, we provide 3 buckets and 64 logs in the inventory. Additionally, we grow mature
1370 wheat and sugar cane in front of the agent and spawn a few chickens and cows around it, ensuring
1371 that the agent can conveniently find the necessary ingredients.

1372 C.3 PROMPT EXAMPLE

1373 The system message that describes the detailed task information to the agent is shown as follows:

1374

```

1375 # Setup
1376 You are a skilled Minecraft player. You are born in the survival mode and
1377 asked to obtain a specific item.
1378 You can interact with the game environment by outputting actions using
1379 python-style pseudo code. For each turn, please call exactly one
1380 predefined function.
1381
1382 # Valid Actions
1383 ## Predefined Function List:
1384 ```
1385 def craft(item: str, num: int = 1):
1386     '''Craft specified number of items. Please ensure that you get enough
1387     ingredients and a craft_table in your inventory.
1388     Args:
1389         obj: the name of the item to craft.
1390         num: the number of items to craft. Default is 1.
1391     Returns:
1392         A string message about whether the crafting is successful.
1393     Examples:
1394     >>> craft("wooden_pickaxe")
1395     Successfully crafted 1 wooden_pickaxe.
1396     >>> craft("bookshelf", 2)
1397     Not enough materials for 2 bookshelf. # You don't have 12
1398     planks and 6 books in your inventory.
1399     ```
1400
1401 def smelt(item: str, num: int = 1):
1402     '''Smelt specified number of items. Please ensure that you get enough
1403     fuels, ingredients, a furnace and a **wooden_pickaxe** in your
1404     inventory.
1405     Args:
1406         obj: the name of the item to smelt.
1407         num: the number of items to smelt. Default is 1.
1408     Returns:
  
```

```

1404     A string message about whether the smelting is successful.
1405     Examples:
1406     >>> smelt("iron_ingot", 2)
1407     Successfully smelted 2 iron_ingot.
1408     >>> smelt("glass")
1409     Not enough fuels. # You don't have enough coals, logs or planks
1410     as fuel.
1411     '''
1412 def equip(item: str):
1413     '''Select an item from your inventory to your hand. Note that if you
1414     want to use some item, you must equip it first!
1415     Args:
1416     item: the name of the item to equip.
1417     Returns:
1418     A string message about whether the equipping is successful.
1419     Examples:
1420     >>> equip("diamond_sword")
1421     Successfully equipped diamond_sword.
1422     >>> equip("diamond_axe")
1423     Can not find diamond_axe in inventory. # You must have the item
1424     in your inventory before equipping it.
1425     '''
1426 def teleport_to_spawn():
1427     '''teleport yourself to the spawn position.
1428     Args:
1429     None.
1430     Returns:
1431     A string message about whether the teleportation is successful.
1432     Examples:
1433     >>> teleport_to_spawn()
1434     Successfully teleported.
1435 '''
1436 def look_up(item: str):
1437     '''Look up the information about crafting the item.
1438     Args:
1439     item: the name of the item/tag to look up.
1440     Returns:
1441     A string message about the information of the item. Note that if
1442     the argument is a tag, information about all possible items
1443     will be returned.
1444     Examples:
1445     >>> look_up("iron_pickaxe")
1446     iron_pickaxe: Crafting iron_pickaxe needs 2 stick, 3 iron_ingot.
1447     Every time youcraft iron_pickaxe with the ingredients above,
1448     you will get 1 iron_pickaxe.
1449     >>> look_up("stone_tool_materials")
1450     stone_tool_materials is a tag. Following items belong to this tag
1451     : cobblestone, blackstone.
1452     cobblestone: It is a raw item you can mine from the environment.
1453     blackstone: It is a raw item you can mine from the environment.
1454     '''
1455 def execute(prompt: str, goal_item: Optional[str] = None, num: Optional[
1456 int] = None)
1457     '''Instruct a lower-level executor model to perform some simple tasks
1458     , like mine something, collect something, move to some place.
1459     Args:
1460     prompt: the prompt to instruct the lower-level executor model. It
1461     should be a simple **verb-object phrase**.
1462     goal_item (optional): the name of the item to obtain during the
1463     execution. If the item is obtained, the executor model will
1464     stop.
1465     num (optional): the number of items to obtain.

```

```

1458 Returns:
1459     A string message about the execution.
1460 Negative Examples: # examples that may cause failure
1461     Your Inventory: Now your inventory has 1 stone_pickaxe, 2 stick.
1462     Equipped Item: Now you hold the stone_pickaxe in your hand.
1463     >>> execute("break iron_ore blocks", "iron_ore", 64)
1464     The executor has reached the maximum number of steps for this
1465     turn without completing your subgoal. # Each turn is limited
1466     in time, 64 iron_ore is too much for one turn.
1467
1468     Your Inventory: Now your inventory has 1 wooden_axe, 12 logs, 4
1469     stick, 1 seed, 1 iron_pickaxe.
1470     Equipped Item: Now you hold the wooden_axe in your hand.
1471     >>> execute("find and mine diamond", "diamond_ore", 1)
1472     The executor has reached the maximum number of steps for this
1473     turn without completing your subgoal. # You are not holding
1474     the right tool for mining diamonds. You should equip the
1475     iron_pickaxe first.
1476
1477     Your Inventory: Now your inventory has 64 dirt.
1478     Equipped Item: Now you hold nothing in your hand.
1479     >>> execute("climb on a tree")
1480     The executor has attempted to execute the action according to
1481     your prompt. You should check whether your intention has been
1482     fulfilled. # The executor can't plan for complex tasks; you
1483     have to break down complex tasks into simple ones. For
1484     example, break down the task of 'climb on a tree' into 'find
1485     a tree', 'use dirt blocks to elevate', and 'jump on the tree
1486     '.
1487
1488     Your Inventory: Now your inventory has nothing.
1489     Equipped Item: Now you hold nothing in your hand.
1490     >>> execute("dig a hole and jump in")
1491     Error: No complex sentences allowed. Keep the prompt a simple **
1492     verb-object phrases**. # Your prompt contains multiple tasks
1493     that may be confusing to the executor.
1494
1495     Your Inventory: Now your inventory has 4 logs.
1496     Equipped Item: Now you hold nothing in your hand.
1497     >>> execute("craft a wooden_axe", "wooden_axe", 1)
1498     Error: You cannot use 'execute' to craft items. Use 'craft'
1499     instead. # The executor cannot craft or smelt items, call '
1500     craft' for 'smelt' function instead.
1501
1502     Your Inventory: Now your inventory has 4 logs, 1 crafting_table.
1503     Equipped Item: Now you hold nothing in your hand.
1504     >>> execute("place crafting_table")
1505     Error: You cannot use 'execute' to craft items or place the
1506     crafting_table. Directly use 'craft' instead. No need to
1507     place the crafting_table. # The 'craft' function will
1508     automatically place the crafting_table during crafting.
1509
1510     Your Inventory: Now your inventory has nothing.
1511     Equipped Item: Now you hold nothing in your hand.
1512     >>> execute("hold down left button to punch the tree to collect
1513     wood", "logs", 1)
1514     The executor has reached the maximum number of steps for this
1515     turn without completing your subgoal. # The description of
1516     the task is too complex, it should be a **verb-object phrase
1517     **.
1518
1519 Positive Examples: # good examples for reference
1520     Your Inventory: Now your inventory has stone_pickaxe, stick.
1521     Equipped Item: Now you hold the stone_pickaxe in your hand.
1522     >>> execute("break iron_ore blocks", "iron_ore", 2)

```

```

1512 Your subgoal has been successfully completed by the executor. #
1513     You have seen the iron_ore and you are using the correct tool
1514     . Note that if you haven't seen the iron_ore, you'd better
1515     use `break stone, obtain iron ore` as your prompt.
1516
1517     Your Inventory: Now your inventory has nothing.
1518     Equipped Item: Now you hold nothing in your hand.
1519     >>> execute("collect wood", "logs", 1)
1520     Your subgoal has been successfully completed by the executor. #
1521     The executor can only understand the instructions of simple
1522     **verb-object phrases**.
1523
1524     Your Inventory: Now your inventory has nothing.
1525     Equipped Item: Now you hold nothing in your hand.
1526     >>> execute("dig a hole", "dirt", 4)
1527     Your subgoal has been successfully completed by the executor. #
1528     Your instructions are simple and easy to understand.
1529
1530     Your Inventory: Now your inventory has 1 wooden_axe, 2 stick.
1531     Equipped Item: Now you hold the wooden_axe in your hand.
1532     >>> execute("find a river")
1533     The executor has attempted to execute the action according to
1534     your prompt. You should check whether your intention has been
1535     fulfilled. # The executor has the ability to find the
1536     environment you are looking for, despite the possibility of
1537     failure.
1538
1539     Prompt Examples: # some simple prompts for reference
1540     "chop down the tree", "break leaves", "collect seeds", "break a
1541     flower", "dig down", "break stone, obtain iron ore", "break
1542     gold_ore blocks", "mine diamond ore", "kill sheep", "milk cow", "
1543     combat spider", "find a river", "break stones", "break sand
1544     blocks", "move out of the cave".
1545     '''
1546     ```
1547     ## Reminder
1548     1. You can only call one function in each turn.
1549     2. If you have no idea on how to solve the task or are unfamiliar with
1550     some items, please call the `look_up` function to check the item.
1551     3. For some items that you can not mine or obtain with your bare hand,
1552     try to equip a pickaxe (wooden_pickaxe, stone_pickaxe, ...) before
1553     mining it.
1554     4. Some necessary resources (e.g., mobs, plants) might be prepared for
1555     you near the spawn point. If you're struggling to find certain
1556     ingredients or find yourself stuck somewhere, you can use the `
1557     teleport_to_spawn` function to return there.
1558     5. When calling the executor, keep the positive examples and negative
1559     examples in mind! If the executor cannot complete your subgoal, check
1560     whether you have the right item in your hand, and try to break your
1561     prompt into smaller steps and adjust your subgoal, modify the prompt,
1562     or carefully repeat the prompt.
1563     6. Do not repeat the failed action in the next round. Try to understand
1564     what went wrong and make a different decision.
1565
1566     # Input
1567     For each dialog, you will be given the following information at the
1568     beginning.
1569     - Task Goal: The item you should obtain in your inventory.
1570     For each turn, you will be given the following information.
1571     1. Feedback on the Action: The feedback on the action you output in the
1572     last turn.
1573     2. Your Inventory: The items in your inventory.
1574     3. Equipped Item: The item you are currently holding in your hand.
1575     4. Location and Orientation: including X, Y, Z, Pitch and Yaw. X and Z
1576     are horizontal coordinates; Y is the height. Pitch measures the tilt

```

1566 of the player’s view: 0, positive values and negative values mean the
 1567 player is looking horizontally, downward, and upward, respectively.
 1568 Yaw measures the rotation around the player’s vertical axis: 0 or 360
 1569 degrees north, 90 degrees east, 180 degrees south, and 270 degrees
 1570 west.
 1571 5. Vision Input: What you see from your perspective.
 1572 # Output
 1573 Now, given these information, you need to think and call the action
 1574 needed to proceed with the task. Your response should include 3 parts
 1575 in the following format in each turn:
 1576 OBSERVATION: <What you observe in the image> Note that the Vision Input
 1577 image won’t be kept in the dialog, so make sure you capture all the
 1578 key information (eg, the biome or items you see) here for future use.
 1579 THOUGHT: <Your step-by-step thoughts>
 1580 ACTION: <The action code> Note that only one function is allowed in each
 1581 dialog turn! Only one line of code is allowed in each dialog turn! If
 1582 your output contains multiple functions or multiple turns of
 1583 functions, only the first one will be executed!

1583 Here is a concrete example of the task input shown in Fig. 8, where the image is enclosed within
 1584 ”{{{}}”:

1585 Your task is to get a cake in your inventory.
 1586 Feedback on the Action: No action before.
 1587 Your Inventory: Now your inventory has 64 oak_log, 3 bucket.
 1588 Equipped Item: Now you hold the oak_log in your hand.
 1589 Location and Orientation: Now you locate in X: 431.50, Y: 65.00, Z:
 1590 -158.50, Pitch: 0.00, Yaw: 0.00.
 1591 Vision Input: {{Image}}

1593 D VAB-ANDROIDLAB

1594
 1595 In this section, we provide additional details regarding VAB-AndroidLab that are not covered in the
 1596 main text due to space limitations.
 1597

1598 D.1 DETAILED DESCRIPTION

1600 To introduce the Android Eval benchmark, we developed a framework including an operational
 1601 environment and a benchmark tailored for agents interacting with Android.
 1602

1603 Android Eval benchmark comprises 119 tasks across 8 different apps, offering evaluation suites
 1604 considering the device’s and screen’s state. It implements evaluation frameworks for both the
 1605 ReAct (Yao et al., 2023) and SeeAct (Zheng et al., 2024) methods. For reproducibility, the Android
 1606 virtual device provides standard evaluation virtual machines preloaded with various apps’ operation
 1607 histories and offline data, ensuring that network or temporal factors do not affect evaluations. To
 1608 simulate real-world tasks, we offer Android virtual machine images with randomized operations,
 1609 ensuring evaluations do not have to start from an initial usage state and enabling more complex task
 1610 completion recognition based on the machine and current page state.

1611 D.2 ACTIONS

1612 In VAB-AndroidLab, agents are required to accomplish diverse user tasks through predefined actions.
 1613

- 1614 • **tap**: Tap element with specific id.
- 1615 • **type**: Type the message into the input box and press enter if needed.
- 1616 • **long press**: Tap element with specific id for a long duration.
- 1617 • **swipe**: Swipe with distance and direction.
- 1618 • **finish**: Finish the task with optional message.
- 1619 • **press back**: Press back button.

- **press home:** Press home button.

1622 D.3 METRICS

1624 The metric we designed is directly oriented towards task completion. We can directly assess the
 1625 task's success rate by checking whether the operation sequence includes necessary screens or device
 1626 states that indicate task completion. For example, in setting an alarm time, we sequentially check if
 1627 the task sequence includes the correctly set alarm time and if the alarm is turned on. Specifically, the
 1628 metrics we designed are as follows:

- **Success Rate:** We measure the success rate by device state and screen state for the operation
 1630 task. We measure the success rate for the query task by comparing the model answer with the
 1631 ground truth.

1633 D.4 PROMPT EXAMPLE

1635 Here is the system prompt we use.

```
1636 You are an agent that is trained to complete certain tasks on a
1637 smartphone. You will be
1638 given a screenshot of a smartphone app. The interactive UI elements on
1639 the screenshot are labeled with numeric tags
1640 starting from 1.
```

```
1641 You can call the following functions to interact with those labeled
1642 elements to control the smartphone:
```

```
1644 1.tap(index: int)
```

```
1645 Taps the UI element labeled with the given number.
1646 Example: tap(5)
```

```
1648 2.text(input_str: str)
```

```
1649 Inserts the given text into an input field.
1650 Example: text("Hello, world!")
```

```
1651 Since we use ADB keyboard, if ADB keyboard ON is displayed on the bottom
1652 of the screen, you can use this function.
```

```
1653 If you think that the keyboard is displayed after your previous operation
1654 , you can try to use this function to input text.
```

```
1655 3.long_press(index: int)
```

```
1657 Long presses the UI element labeled with the given number.
1658 Example: long_press(5)
```

```
1660 4. swipe(index: int, direction: str, dist: str)
```

```
1661 Swipes the UI element in the specified direction and distance. "direction
1662 " is a string that
```

```
1663 represents one of the four directions: up, down, left, right. "dist"
1664 determines the distance of the swipe and can be one
1665 of the three options: short, medium, long.
```

```
1666 Example: swipe(21, "up", "medium")
```

```
1667 5. back()
```

```
1668 Simulates a back button press on the smartphone.
```

```
1670 6. home()
```

```
1672 Simulates a home button press on the smartphone.
```

```
1673 7. wait(interval: int)
```

1674
 1675 Pauses the execution for the given number of seconds. Default is 5 second
 1676 .
 1677
 1678 8. finish(message: str)
 1679
 1680 Ends the task and provides the final output. You can return the final
 1681 output of the task as a string.
 1682 Example: finish("Task completed")
 1683
 1684 Now, given the following labeled screenshot, you need to think and call
 1685 the function needed to proceed with the task.
 1686 Your output should include only action part in the given format:
 1687
 1688 Action: <The function call with the correct parameters to proceed with
 1689 the task. If you believe the task is completed or
 1690 there is nothing to be done, you should use finish function. You cannot
 1691 output anything else except a function call
 1692 in this field.>
 1693
 1694 Whenever you think the task is finished, you should use finish function
 1695 to avoid extra operations.
 1696
 1697 If you found yourself in a loop or the task is not proceeding as expected
 1698 , you might consider changing your operation and try other methods.
 1699 If you operate same action 5 times, the program will automatically stop.
 1700 If tap operation is not working, you can try long press operation.
 1701
 1702 You can only take one action at a time, so please directly call the
 1703 function.

1701 E WEBARENA-LITE

1702 In this section, we provide additional details regarding WebArena-Lite that are not covered in the
 1703 main text due to space limitations.

1704 E.1 DETAILED DESCRIPTION

1705 WebArena (Zhou et al., 2023) is designed to evaluate the ability of agents to perform complex user
 1706 tasks described in high-level natural language in a realistic, interactive web environment. To achieve
 1707 this goal, WebArena presented a highly simulated and interactive web environment, which consists
 1708 of five common websites, including Gitlab, map, forum, online shopping, and content management
 1709 platform. It is also equipped with external tools such as sketch pad and calculator, which enhance
 1710 the ability of the agents to perform user tasks. In contrast to other benchmarks where the agents are
 1711 constrained to act as website users, WebArena proposed innovative ways to simulate different user
 1712 roles. For instance, they constructed a content management platform (CMS) and granted the agent
 1713 full administrative privileges. This assesses the agent’s capacity to assume various roles in complex
 1714 scenarios.

- 1715 • **Task Description:** As web GUI agents, LMMs are asked to accomplish user instructions on
 1716 certain websites. For example, on OneStopShop website, an instruction would be “What do
 1717 customers say about brush from sephora”, and LMM agents should search for
 1718 the product, enter the review section, and summarize the customer reviews (or turn out finding no
 1719 review). To enable the action of LMM agents with visual input, we implement HTML SoM (Koh
 1720 et al., 2024) to annotate operable HTML elements with ids on the screenshot, we also provide a
 1721 list of textual information for all clickable elements. LMM agents generate actions and the id of
 1722 elements being operated by playwright.
- 1723 • **Test Set:** We build WebArena-Lite, a subset of 165 representative tasks by selection, refinement,
 1724 and adaptation to multimodality evaluation (i.e., screenshot). Our refinement focuses on resolving
 1725 implausible judge conditions, where 30 tasks are being manually fixed (Cf. Appendix E.4). The

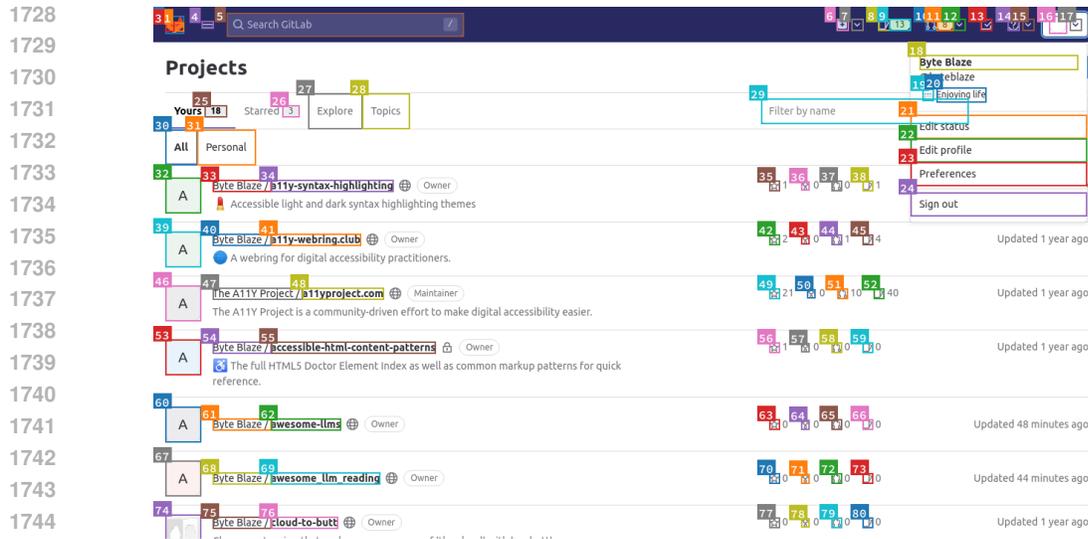


Figure 9: This is an example of WebArena-Lite task where we use the SoM approach to highlight actionable elements. This task requires the agent to modify the user’s status information. To accomplish this, the agent initially clicks on the user’s avatar, which directs them to the status shown in the figure. At this point, the agent should select the option labeled ”(21) Edit Status” in order to access the modification page and complete the task.

implausibility may involve wrong answers, too-strict criteria (e.g., `exact_match`), impossible tasks due to environment bugs, etc. Additionally, we remove cross-website tasks for simplicity of testing.

- **Training Set:** Creating environment-dependent task instructions and trajectories for training on web could be challenging. In VAB, for each website we first summarize the basic functions and valid items for synthetic queries to condition on. Based on summarized functions, we come up with an array of task prototypes (with item placeholders) and manually write playwright scripts as rule-based solvers for each task prototype. We fill task prototypes with both valid and invalid items to yield detailed instructions (later being rephrased by LLMs for expression diversity), and run corresponding solvers on the website to collect groundtruth trajectories with screenshots and operations. 5 authors create around 40 task prototypes with corresponding solvers, and generating 1,186 valid training samples (i.e., instruction, trajectory, and reward function) for WebArena-Lite.

E.2 ACTIONS

In WebArena-Lite, agents are required to accomplish diverse user tasks through a series of predefined actions. However, real-world webpages are often complex, and thus, we provide these actions in order to ensure simplicity and practicality.

- **click:** Click element with specific id.
- **hover:** Hover element with specific id.
- **type:** Type the message into the input box with a specific id and press enter if needed.
- **press:** Emulates a keyboard key combination.
- **scroll:** Scrolls the page up or down.
- **new_tab:** Opens a new tab in the current browser.
- **tab_focus:** Switches to the tab with specific index.
- **close_tab:** Closes the current tab.
- **goto:** Go to specific URL.
- **go_back:** Go back to the previous page.
- **go_forward:** Go to the next page if it exists.

- **stop**: Terminates the operation, returns the response, and exits.

E.3 METRICS

In real-world web browsing scenarios, there can be multiple ways for an agent to accomplish a task. Consequently, WebArena-Lite only considers whether the task has been completed or not, without considering the execution trajectory of the agent, therefore, the metric used in WebArena-Lite is **Success Rate (SR)**. We maintain the evaluation method described by WebArena (Zhou et al., 2023), which can be categorized into three categories based on task type.

- **Question Answering**: Agent needs to give an answer and the score depends on the string-matching result.
- **Webpage Navigation**: Agent must navigate to a specific web page. The completion of the task is dependent on the URL of the page on which the agent terminated.
- **Content modification**: Agent needs to interact with the environment to modify the configuration of the webpage, and the evaluation will extract the content of the page and match it to check whether the content meets the expectations.

In light of the aforementioned considerations, string-matching patterns can be classified into three distinct categories:

- **exact_match**: The response of the agent is scored when it exactly matches the token sequence corresponding to the answer.
- **must_include**: Answers that contain a specific token sequence are considered a match.
- **fuzzy_match**: Utilizes LLMs such as GPT-4 to assist in determining whether an answer is correct.

The selection of appropriate evaluation metrics for distinct types of tasks enables the construction of a comprehensive and relatively accurate test set.

E.4 TASK AMENDMENT

Some tasks in WebArena have typos, incorrect answers, and inaccurate scoring criteria. Therefore, we selected 165 tasks from WebArena with different templates and then corrected 39 of them, as shown in Table E.4. Considering that the model uses natural language to answer the questions, we change the tasks that require exact match to must include or fuzzy match, and also correct the answers.

Table 7: Task instructions fixed in WebArena-Lite

ID	Website	Task	Before	After
7	Map	Tell me the full address of all international airports that are within a driving distance of 50 km to Carnegie Mellon University.	exact_match	fuzzy_match
33	Map	I will arrive Pittsburgh Airport soon. Provide the name of a Hilton hotel in the vicinity, if available. Then, tell me the the shortest walking distance to a supermarket from the hotel.	must_include	fuzzy_match
37	Map	Check if the police station in pittsburgh can be reached in one hour by car from gates building at CMU.	must_include	fuzzy_match
43	CMS	List the top 3 search terms in my store.	hollister, Joust Bag, Antonia Racer Tank	hollister, Joust Bag, nike
65	CMS	Which customer has completed the fifth most number of orders in the entire history?	Jane Doe	Matt Baker
71	Map	What is the zip code of Chatham University?	exact_match	must_include
82	Map	What is the duration required to first walk from Massachusetts Institute of Technology to Harvard University, and then drive to Boston Logan International Airport?	63 min	64 min

Table 7: Task instructions fixed in WebArena-Lite

ID	Website	Task	Before	After
97	Map	Tell me the distance to drive from Carnegie Mellon University to the top computer science school in massachusetts.	must_include	fuzzy_match
98	Map	Where is the nearest tea cafe to University of Pittsburgh, and what is the walking distance to it?	must_include	fuzzy_match
103	Gitlab	Display the list of issues in the <code>kkroening/ffmpeg-python</code> repository that have labels related to questions.		URL: sort by <code>created_date</code> , state is opened
109	CMS	Presents the monthly count of successful orders <code>{{period}}</code> in <code>MM:COUNT</code> format.	January: 11 orders, February: 16 orders	01:11, 02:16
127	CMS	What brands appear most frequently among the top search terms?	Hollister, Joust, Antonia	Hollister
135	Gitlab	How many commits did Eric and Kilian make to <code>allproject</code> on 1/3/2023?	1	0
167	OSS	What are the main criticisms of this product? Please extract the relevant sentences.	must_include	fuzzy_match
215	CMS	What are the key aspects that the customers don't like about Circe ice fleece.	fuzzy_match ("Material quality, ...")	exact_match ("N/A")
225	OSS	What do customers say about brush from sephora.	N/A	No reviews available
235	OSS	Get the order number of my most recent under delivery order.	fuzzy_match	must_include
236	Map	Where is the nearest pharmacy from Carnegie Mellon I can walk within 20mins.	must_include	fuzzy_match
240	OSS	I am doing a market survey for one stop market, show me the most expensive product from competitive swimwear category.		URL
254	Map	What is the phone number of Western Pennsylvania Hospital.	exact_match	fuzzy_match
259	Gitlab	Get me my RSS feed token.	exact_match	must_include
287	Map	How much time does it take from Pittsburgh to Philadelphia by car?	5h 47m	8h 33m
288	CMS	Tell me the name of the customer who has the most cancellations in the history.	exact_match	must_include
296	Gitlab	Show me the command to clone the best GAN python implementation with SSH.	metis.lti.cs.cmu.edu	localhost
311	Gitlab	Tell me who has made the most contributions, in terms of number of commits, to the Pytorch GAN project.	exact_match	must_include
348	CMS	How many reviews our shop received in May 2023?	must_include	fuzzy_match
349	Gitlab	Who else have access to my repo <code>gimmiethat.space</code> , show me their usernames.	exact_match	must_include
367	Map	Measure distance between Carnegie Mellon University and CVS (closest one) by walking.	exact_match	fuzzy_match
376	OSS	Summarize customer reviews for Amazon Echo Dot 3rd generation.	N/A	fuzzy_match ("Serena: it. ...")
392	Gitlab	Post <code>"Good idea"</code> for the merge request related to <code>color utility in allproject.com project</code> .		URL
528	OSS	Draft a refund message via their <code>"contact us"</code> form for the phone screen protector I bought March 2023. It broke after three days of use. The shop requires the order id, the reason and the amount to refund in the message. Don't submit yet.	must_include	fuzzy_match

Table 7: Task instructions fixed in WebArena-Lite

ID	Website	Task	Before	After
538	CMS	Modify the address of order #299 to 456 Oak Avenue, New York, NY, 10001.	... 456 Oak Avenue, Apartment 5B, New York...	... 456 Oak Avenue, New York...
641	Reddit	Post a notice on a virtual meetup for Harry Potter enthusiasts on July 8th in the books subreddit.	must_include	fuzzy_match
645	Reddit	Post in books subreddit about what could machine learning help the correpong field.	must_include	fuzzy_match
657	OSS	Fill the ``contact us`` form in the site for a refund on the speaker I bought, stating that it broke after just three days of use. Also, ensure to include the order number #148 and the product SKU. Don't submit yet, I will check.	must_include	fuzzy_match
668	Gitlab	Submit a merge request for al lyproject.com/redesign branch to be merged into master branch, assign Roshan Jossy as the reviewer.	Justin Armstrong	Roshan Jossy
693	OSS	Draft an email to the shop owner via their contact us function for a coupon as my refund is suppoed to be replaced by a coupon.	program_match	url_match
798	OSS	Change the delivery address for my most recent order to 77 Massachusetts Ave, Cambridge, MA.	fuzzy_match	must_include

E.5 PROMPT EXAMPLE

Here is the system prompt we use, you can find more prompt examples in VisualWebArena (Koh et al., 2024).

```

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

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

The actions you can perform fall into several categories:

Page Operation Actions:
``click [id]``: This action clicks on an element with a specific id on
the webpage.
``type [id] [content]``: 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, i.e., ``type [id] [content] [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 [down]`` or ``scroll [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.

```

```

1944   ``close_tab``: Close the currently active tab.
1945
1946   URL Navigation Actions:
1947   ``goto [url]``: Navigate to a specific URL.
1948   ``go_back``: Navigate to the previously viewed page.
1949   ``go_forward``: Navigate to the next page (if a previous 'go_back'
1950     action was performed).
1951
1952   Completion Action:
1953   ``stop [answer]``: Issue this action when you believe the task is
1954     complete. If the objective is to find a text-based answer, provide
1955     the answer in the bracket.
1956
1957   Homepage:
1958   If you want to visit other websites, check out the homepage at http://
1959     homepage.com. It has a list of websites you can visit.
1960     http://homepage.com/password.html lists all the account name and password
1961     for the websites. You can use them to log in to the websites.
1962
1963   To be successful, it is very important to follow the following rules:
1964   1. You should only issue an action that is valid given the current
1965     observation
1966   2. You should only issue one action at a time.
1967   3. You should follow the examples to reason step by step and then issue
1968     the next action.
1969   4. Generate the action in the correct format. Start with a "In summary,
1970     the next action I will perform is" phrase, followed by action inside
1971     ``````. For example, "In summary, the next action I will perform is
1972     ``click [1234]``".
1973   5. Issue stop action when you think you have achieved the objective. Don'
1974     t generate anything after stop.

```

1973 F VAB-CSS

1975 In this section, we provide additional details regarding VAB-CSS that are not covered in the main
 1976 text due to space limitations.

1979 F.1 DETAILED DESCRIPTION

1981 Existing datasets for frontend design have two major shortcomings: 1) They focus mainly on single-
 1982 round interactions, and 2) They do not provide definitive success metrics for individual tasks. Instead,
 1983 these benchmarks assess using continuous metrics like CLIP score (Si et al., 2024) or qualitative
 1984 analysis only (Laurençon et al., 2024). The reason is that they expect the model to output an
 1985 entire HTML file replicating the target web design, which is too challenging and unrealistic for
 1986 current LMMs. Therefore, employing a definitive success rate as the metric is meaningless for them.
 1987 Consequently, they may fail to adequately assess LMMs' potential in serving as adaptive agents
 1988 that can make new decisions based on varying observations. Also, a binary success rate is often
 1989 more decisive and crucial to determine whether agents can faithfully execute human instructions,
 1990 which is essential for practical use. To address these issues, we introduce a VAB-CSS, which is
 1991 better tailored for evaluating multimodal agents. In VAB-CSS, an agent is expected to strictly take a
 1992 sequence of actions using our provided toolkit to accomplish a task (Section. F.3). Specifically, it
 1993 needs to iteratively refine the CSS definition based on the rendering outcomes it receives. The more
 1994 constrained action space based on our toolkit, compared to outputting an entire HTML file, along
 1995 with a more practical goal for current LMMs (i.e., CSS bug-fixing), makes it possible to evaluate a
 1996 definitive success rate for a given task. Additionally, VAB-CSS makes minimal assumptions in terms
 1997 of simplifying the task environment, such as embedding all CSS definitions within a single HTML
 page or replacing images with placeholders in existing datasets. Instead, the agent directly operates
 over the entire web frontend project to fix the CSS style. See a comprehensive checklist in Table 8.

Table 8: A fine-grained comparison of VAB-CSS with existing datasets on web frontend development. VAB-CSS provides both training and test data. Additionally, its multi-round nature, definitive success rate metric, and multi-file environment make it well-suited as a practical multimodal agent task.

	Train	Test	Multi Round	Definitive Eva.	Multi-File Env.
WebSight (Laurençon et al., 2024)	✓	✗	✗	✗	✗
Design2Code (Si et al., 2024)	✗	✓	✗	✗	✗
VAB-CSS	✓	✓	✓	✓	✓

F.2 DATA COLLECTION

Random CSS Corruption. To ensure the task is manageable for LMMs, each task instance involves corrupting a single categorical property of a random CSS rule by either altering its value or removing it entirely. Note that, even fixing a single corruption is already highly challenging for current LMMs, and a tiny corruption can often lead to a drastic change in visual effect (see Figure 10). We can increase the task’s complexity in the future by involving multiple corruptions once the single-corruption task has been mastered.

Human Annotations. Existing LMMs struggle to identify the difference between the current rendering and the target design, so we manually annotate each instance with a natural language description of the difference between the two images. Such natural language descriptions could serve as additional clues for the agent to perceive the visual difference (see a concrete example of annotation in Figure 10).

Training Trajectories. To collect training trajectories, we primarily sample from the predictions of `gpt-4o` on our training instances, retaining the successful trajectories for training. Given the success rate of `gpt-4o` is around 35%, we also sample its trajectories in a more lenient setting where the target CSS rule to edit is provided as input. For task instances where `gpt-4o` succeeds in the lenient setting, we combine its successful trajectory with its failure trajectory in the standard setting to create a more realistic trial-and-error trajectory.

F.3 ACTIONS

In VAB-CSS, the agent is expected to interact with a practical frontend project, potentially with numerous CSS files, to fix its style issues. Inputting the entire project directly into the agent is impractical and inefficient. Instead, the agent has access only to screenshots and the current HTML code. To facilitate effective navigation and editing within the project, we provide the agent with a toolkit. This toolkit allows the agent to locate and edit incorrect CSS definitions seamlessly, without needing to know the specific file containing the CSS rule.

- **get_selectors_by_html_element:** This function allows the agent to locate a list of CSS selectors, potentially from various files, associated with an HTML element whose style appears to be incorrect.
- **select_rule:** This function allows the agent to check the definition of a CSS rule by specifying a CSS selector.
- **edit_rule:** This function enables the agent to update the property value of a CSS rule for a specified CSS selector.
- **revert_last_edit:** During the trial and error, the agent can revert an edit it later determines to be incorrect.

F.4 METRICS

As discussed earlier, a critical feature of VAB-CSS, compared with existing benchmarks, is its definitive success rate evaluation. The most straightforward way to determine whether a task is successfully handled is to check whether the SSIM similarity between the target design and the final rendering is 1.0. However, we have observed that this can be too strict. Typically, an SSIM greater

2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105

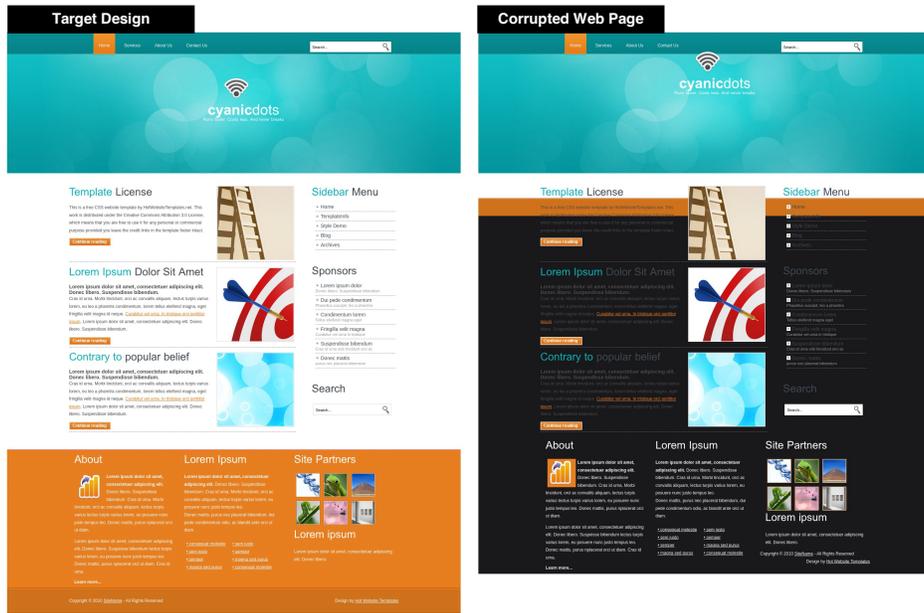


Figure 10: This is an example of our annotation task. Authors are shown the target design and a corrupted web page side by side to prompt them to describe necessary adjustments in natural language. In this example, the instruction is: “Correct the background color of the footer and main section, and adjust the positioning of elements, including centering the website logo in the header by moving it downward.” The two screenshots, along with the HTML code and annotated instruction, will collectively serve as the initial task input for the agent.

than 0.9 indicates minimal differences that are hard for humans to perceive.⁴ Therefore, we define a task as successful if the final similarity is greater than 0.9. Finally, we adopt two metrics on our entire test set.

- **Success Rate (SR):** This is the primary metric indicating the ratio of tasks in the test set that have been successfully fixed based on our definition.
- **Improve Rate (IR):** This metric evaluates the ratio of tasks where the final rendering is more similar to the target design than the initial rendering. It serves as a complementary soft metric to the success rate.

F.5 PROMPT EXAMPLE

The system message that describes the detailed task information to the agent is shown as follows:

You are a CSS agent. You will be given a target screenshot and an html file. Your job is to correct perceive the layout difference between the current rendering and the target screenshot, then accordingly fix the css rules used in the html file to match the target screenshot. To facilitate the process, you can use the following tools provided by the system:

1. get_selectors_by_html_elements

Sometimes, the exact selector of the rule you want to edit is not clear. This tool takes the html element specification that could be directly passed to soup.find_all as input and returns the matched selectors. For example, `get_selectors_by_html_elements("a", {'data-custom': 'custom-value'}, string='haha', class_='xxx')`. The argument should be the string representation of valid arguments of the find_all method in BeautifulSoup, which means we can directly do `eval(f"soup.find_all({argument})")`. Please strictly stick to the usage of BeautifulSoup.

⁴This threshold of 0.9 is an empirical choice based on our observations.

2106 Make sure the arguments are valid (e.g., the tag name must be wrapped
 2107 with quotes, attributes should be a dictionary). You can use this
 2108 tool to first find the selector of the rule of a specific html
 2109 element whose style you want to fix.

2110 2. select_rule
 2111 This takes the css rule's selectorText as input and returns the rule. You
 2112 can use this tool to view the properties of a rule, which may help
 2113 you to decide which rule to edit. Usually, it's recommended to first
 2114 use this tool to view the rule before deciding which rule to edit.

2115 3. edit_rule
 2116 This takes the css rule's selectorText, the property name, and the value
 2117 of the property as input. You can use this tool to change the value
 2118 of a property of a rule or insert a new property to the rule, if you
 2119 believe this change would make the rule closer to the target
 2120 screenshot. Note that, most of the layout issues are related to the
 2121 categorical properties, such as border, float, display, overflow,
 2122 position, etc.

2123 4. revert_last_edit
 2124 This tool reverts the last single edit you made. You can use this tool to
 2125 undo the last edit, if you believe it was a mistake. This action
 2126 takes no arguments.

2127 Make sure the selectorText is valid based on the html file, i.e., it's
 2128 from the class or id of the html elements. In addition, please focus
 2129 on the major layout issue! Ignore the font size, font family, and
 2130 color of the text, even if you believe they are not perfect.

2131 You can only take ONE action at a time!! For each step, you may first
 2132 state your thought, then take an action following the format of
 2133 Thought: ...
 2134 Action: ... (do not add any linebreak after the colon).
 2135 For example, you may output
 2136 "Thought: I think I should adjust the alignment property of the rule,
 2137 because the target screenshot shows the text should be centered.
 2138 Action: edit_rule('.templatemo_menu li', 'text-align', 'center')".

2139 After editing a rule or inserting a rule, you will see the updated
 2140 screenshot of the html file. You should decide your next action (e.g
 2141 ., to revert the last edit or keep adjusting the css) based on the
 2142 updated screenshot. If you think you have already fixed the css style
 2143 , please say exactly "I have fixed the css style".

2144 Please strictly follow the format specified above, and please don't
 2145 repeat the same action in multiple rounds. Also note that, you don't
 2146 need to worry about how these tools are executed, your job is just to
 2147 correctly predict the tool invocation.

2148 Here is a concrete example of the task input shown in Fig. 10, where variables are enclosed within
 2149 “{}”:

2150 Here is a screenshot of the target design:
 2151 {{Image 1}}
 2152 Here is the screenshot of the current web page:
 2153 {{Image 2}}
 2154 Here is the HTML code of the current web page:
 2155 {{HTML file}}

2156 Correct the background color of the footer and main section, and adjust
 2157 the positioning of elements, including centering the website logo in
 2158 the header by moving it downward.

2159

G CASE STUDIES

G.1 VAB-OMNIGIBSON CASES

In this section, we show successful and failed examples of `gpt-4o` and `InternVL-2` in Figure 11, Figure 12, Figure 13 and Figure 14.

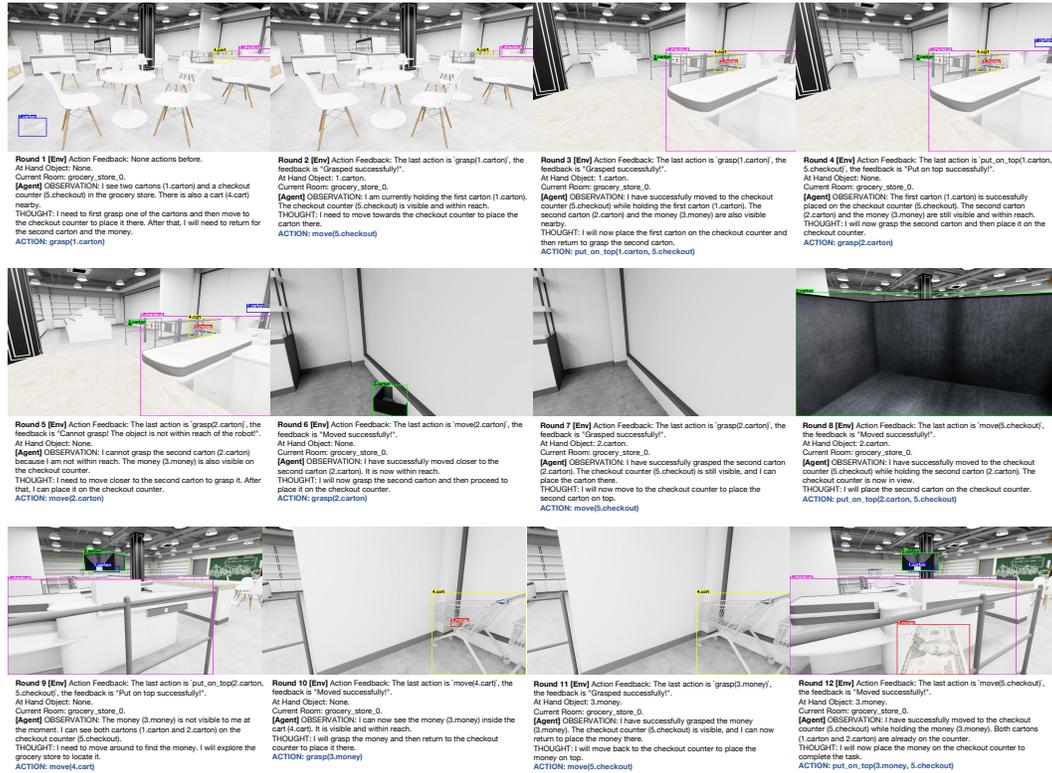


Figure 11: This is an example of `gpt-4o` successfully completing a VAB-OmniGibson task. Task instruction: “Place the two cartons and the money on the checkout counter in the grocery store.”

2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267

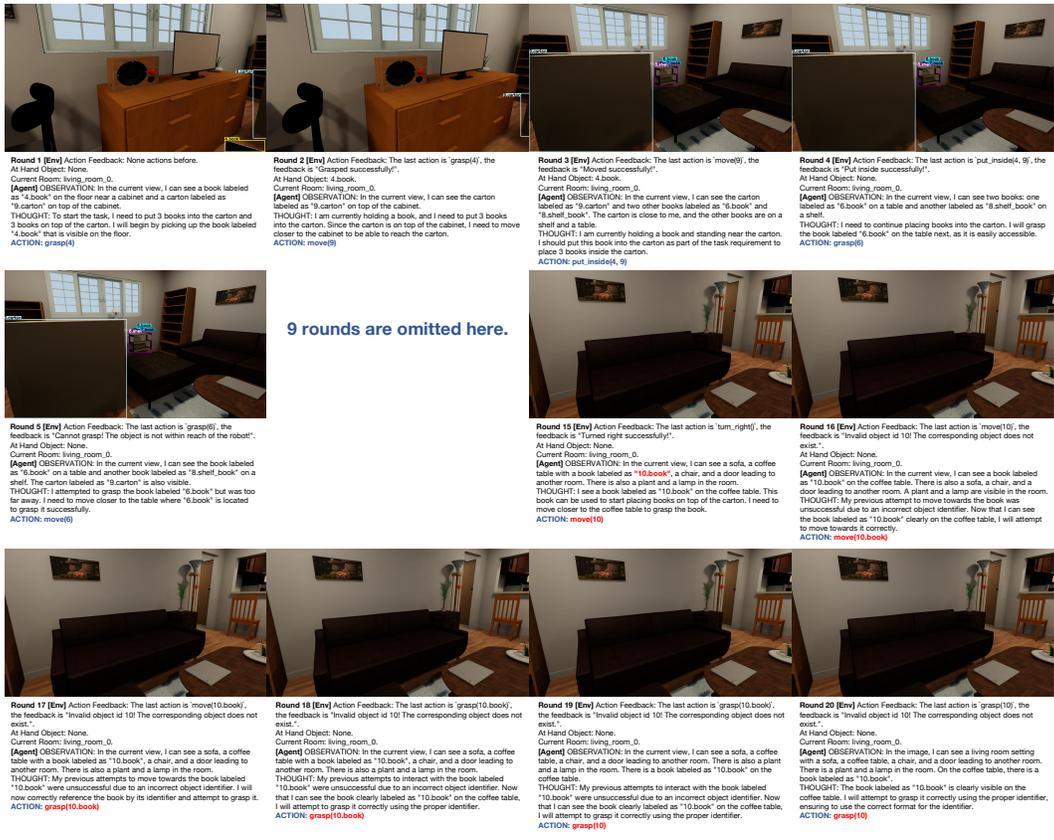


Figure 12: This is an example of `gpt-4o` failing to complete a VAB-OmniGibson task. Task instruction: “There are 7 books in the living room. Please put 3 books into the carton, and 3 books on top of the carton.” `gpt-4o` incorrectly identified the presence of “10.book” in the image, and the erroneous actions are highlighted in red.



Figure 13: This is an example of `InternVL-2` successfully completing a VAB-OmniGibson task. Task instruction: “Pick up a document from the breakfast table or the floor, and then throw it into the recycling bin in the private office.”

2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321

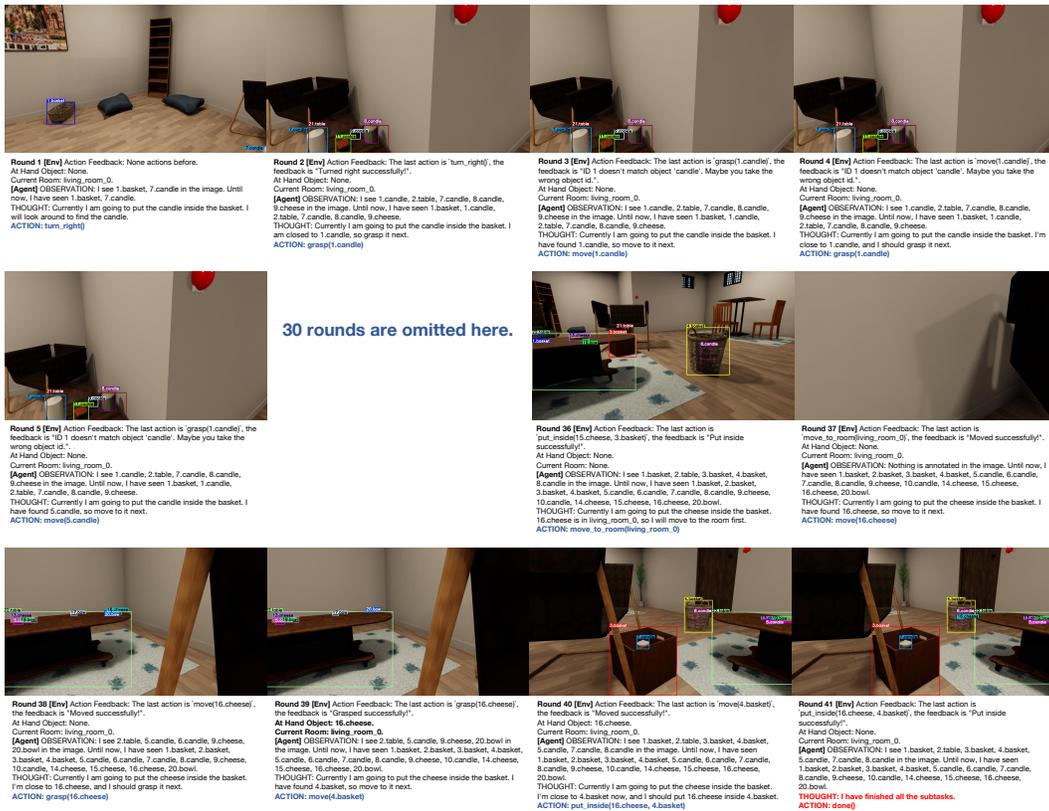


Figure 14: This is an example of InternVL-2 failing to complete a VAB-OmniGibson task. Task instruction: “There are 4 candles, 4 cheeses and 4 baskets in the living room. Please place 5.candle and 13.cheese into 1.basket, 6.candle and 14.cheese into 2.basket, 7.candle and 15.cheese into 3.basket, 8.candle and 16.cheese into 4.basket.” InternVL-2 didn’t put “5.candle” into the basket, but terminated the task with action “done()”.

2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

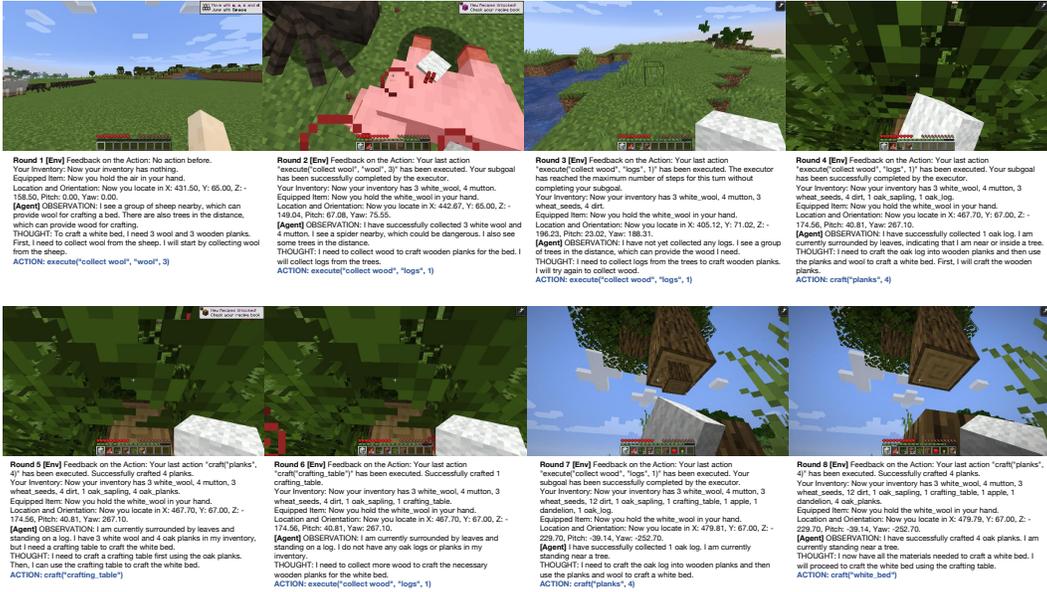


Figure 15: This is an example of gpt-4o successfully completing a VAB-Minecraft task. Task instruction: "Your task is to get a white bed in your inventory."

G.2 VAB-MINECRAFT CASES

In this section, we show successful and failed examples of gpt-4o and InternVL-2 in Figure 15, Figure 16, Figure 17 and Figure 18.

2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429

Round 1 [Env] Feedback on the Action: No action before. Your inventory: Now your inventory has nothing. Equipped Item: Now you hold the air in your hand. Location and Orientation: Now you locate in X: 431.50, Y: 65.00, Z: -158.50, Pitch: 0.00, Yaw: 0.00. [Agent] OBSERVATION: I see a plains biome with grass blocks, a few trees in the distance, and a small body of water to the right. There are also some dark blocks on the ground which could be a cave entrance or a small hole. THOUGHT: To craft a fishing rod, I need 3 sticks and 2 strings. I can obtain sticks by crafting them from wood planks, which in turn can be crafted from logs. I can get logs by chopping down trees. For strings, I need to kill spiders or find a cobweb to break. Since I have nothing in my inventory, my first step should be to collect wood from the trees. I'll start by executing an action to collect wood. ACTION: execute/collect wood, logs, 3

Round 2 [Env] Feedback on the Action: Your last action "execute/collect wood, logs, 3" has been executed. The executor has reached the maximum number of steps for this turn without completing your subgoal. Your inventory: Now your inventory has 2 oak_log, 1 oak_sapling. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 465.27, Y: 65.00, Z: -177.68, Pitch: -13.20, Yaw: 24.76. [Agent] OBSERVATION: The image shows a plains biome with a clear sky, indicating it is daytime. There are grass blocks, a few trees, and a hill in the distance. There are no visible spiders or cobwebs for string, and no immediate threats are apparent. THOUGHT: I have 2 oak logs in my inventory, which is enough to craft the sticks needed for a fishing rod, but I still need one more log to have enough planks for crafting a crafting table. Since I didn't complete the subgoal of collecting 3 logs in the last turn, I should try to collect at least one more log. After that, I can craft a crafting table and then stick from the logs. ACTION: execute/collect wood, logs, 1

Round 3 [Env] Feedback on the Action: Your last action "execute/collect wood, logs, 1" has been executed. Your subgoal has been successfully completed by the executor. Your inventory: Now your inventory has 3 oak_log, 1 oak_sapling. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows a first-person view of the player standing in front of a recently chopped oak log block, with grass and foliage around. The player's health bar is visible, indicating that they are in survival mode. THOUGHT: Now that I have 3 oak logs, I can craft them into planks and then create a crafting table. With the crafting table, I can then craft the sticks needed for the fishing rod. However, I still need to obtain string, which typically comes from spiders or cobwebs. For now, I will focus on crafting the necessary items from the logs I have collected. ACTION: craft/crafting_table

Round 4 [Env] Feedback on the Action: Your last action "craft/crafting_table" can not be executed. Not enough materials for 1 crafting table. Your inventory: Now your inventory has 3 oak_log, 1 oak_sapling. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's perspective looking down at an oak log block on the ground, with grass and other foliage around. The player's hand is visible, holding an oak log. THOUGHT: It seems there was a misunderstanding in my previous action. I need to first craft the oak logs into planks before I can craft a crafting table. Each oak log can be crafted into 4 planks, and I need 4 planks to make a crafting table. I will craft the planks first and then proceed to craft the crafting table. ACTION: craft/planks, 4

Round 5 [Env] Feedback on the Action: Your last action "craft/planks, 4" has been executed. Successfully crafted 4 planks. Your inventory: Now your inventory has 2 oak_log, 1 oak_sapling, 4 oak_planks. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's perspective in Minecraft, looking at an oak log block on the ground with grass and foliage around. The player's health bar and hunger are visible, with oak logs and oak planks in the hotbar. The environment appears to be a forest or plains biome. THOUGHT: Now that I have 4 oak planks, I can craft a crafting table. After crafting the crafting table, I will be able to use it to craft sticks, which are one of the components needed for a fishing rod. However, I still need to obtain string, which I can get from spiders or cobwebs. For now, I will proceed with crafting the crafting table. ACTION: craft/crafting_table

Round 6 [Env] Feedback on the Action: Your last action "craft/crafting_table" has been executed. Successfully crafted 1 crafting table. Your inventory: Now your inventory has 2 oak_log, 1 oak_sapling, 1 crafting_table. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's perspective in Minecraft, looking down at an oak log block on the ground with grass and foliage around. The player's health bar and hunger are visible, with oak logs and a crafting table in the hotbar. The environment appears to be a forest or plains biome. THOUGHT: With the crafting table now in my inventory, I can craft sticks from the remaining oak logs. I need to craft these sticks in order to eventually make a fishing rod. However, I still need to obtain string, which I can get from spiders or cobwebs. For now, I will focus on crafting the sticks. ACTION: craft/sticks, 4

Round 7 [Env] Feedback on the Action: Your last action "craft/sticks, 4" can not be executed. Not enough materials for 4 sticks. Your inventory: Now your inventory has 2 oak_log, 1 oak_sapling, 1 crafting_table. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's perspective in Minecraft, looking down at an oak log block on the ground with grass and foliage around. The player's health bar and hunger are visible, with oak logs and a crafting table in the hotbar. The environment appears to be a forest or plains biome. THOUGHT: It seems I made an error in the quantity of sticks I attempted to craft. I need to convert the remaining oak logs into planks first before I can craft the sticks. Each oak log can be crafted into 4 planks, and I need 2 planks to craft 4 sticks. I will craft more planks from the oak logs and then proceed to craft the sticks. ACTION: craft/planks, 2

Round 8 [Env] Feedback on the Action: Your last action "craft/planks, 2" has been executed. Successfully crafted 2 planks. Your inventory: Now your inventory has 1 oak_log, 1 oak_sapling, 1 crafting_table, 4 oak_planks. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking at an oak log block on the ground with grass and foliage around. The player's health bar and hunger are visible, with an oak log and oak planks in the hotbar. The cursor is pointing at the oak log, indicating the player is ready to interact with it or place it. THOUGHT: Now that I have a total of 8 oak planks, I can craft the sticks needed for the fishing rod. I need to convert the remaining oak logs into planks first before I can craft the sticks. Each oak log can be crafted into 4 planks, and I need 2 planks to craft 4 sticks. I will craft more planks from the oak logs and then proceed to craft the sticks. ACTION: craft/sticks, 4

Round 9 [Env] Feedback on the Action: Your last action "craft/sticks, 4" has been executed. Successfully crafted 4 sticks. Your inventory: Now your inventory has 1 oak_log, 1 oak_sapling, 2 oak_planks, 4 sticks, 1 crafting_table. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking down at a placed oak log block on the ground with grass and foliage around. The player's health bar and hunger are visible, with an oak_log, oak planks, sticks, and a crafting table in the hotbar. The environment appears to be a forest or plains biome during the daytime. THOUGHT: With the sticks now crafted, the next step is to obtain string to complete the fishing rod. Since it is daytime, spiders will be rare to find. I could look for a cave or a dark area where spiders might spawn, or I could wait until night when spiders will spawn naturally in the overworld. However, since I don't control the time, I should try to find a cave or dark area to look for spiders. Before I do that, I should equip a sword to defend myself and to break cobwebs if I find any. ACTION: equip/sword, 1

Round 10 [Env] Feedback on the Action: Your last action "equip/sword, 1" has been executed. Successfully equipped oak_log. Your inventory: Now your inventory has 1 oak_sapling, 1 crafting_table, 2 oak_planks, 4 sticks, 1 oak_log. Equipped Item: Now you hold the oak_log in your hand. Location and Orientation: Now you locate in X: 478.28, Y: 68.00, Z: -229.70, Pitch: 36.71, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking down at an oak log block placed on the ground with grass and foliage around. The player's health bar and hunger are visible, with an oak_log equipped in the player's hand, and oak planks, sticks, and a crafting table in the hotbar. The environment appears to be a forest or plains biome during the daytime. THOUGHT: Now that I have equipped an oak_log, I can use it as a rudimentary weapon or tool. However, it would be more effective to craft a wooden sword for defense and to fight spiders. Before I go looking for spiders, I should craft a wooden sword using the planks and sticks I have. This will improve my chances of successfully obtaining string from spiders. ACTION: craft/wooden_sword

Round 11 [Env] Feedback on the Action: Your last action "craft/wooden_sword" has been executed. Successfully crafted 1 wooden_sword. Your inventory: Now your inventory has 1 wooden_sword, 1 oak_sapling, 1 crafting_table, 3 sticks, 1 oak_log. Equipped Item: Now you hold the wooden_sword in your hand. Location and Orientation: Now you locate in X: 478.39, Y: 67.00, Z: -229.70, Pitch: 61.00, Yaw: 188.95. [Agent] OBSERVATION: The image shows the player's first-person view looking directly down at the ground, with redstone dust on the grass blocks. The player's health bar and hunger are visible, with a wooden sword equipped in the player's hand, and other items including a crafting table and sticks in the hotbar. THOUGHT: The presence of redstone dust suggests that I might be near a redstone contraption or that it was placed there for some reason. However, this does not directly help with my current task of obtaining a fishing rod. I need to find spiders to collect string. Since I am now equipped with a wooden sword, I should look for a cave or a dark area where spiders are likely to spawn. I will execute an action to search for a cave in hopes of encountering spiders. ACTION: execute/find_a_cave

Round 12 [Env] Feedback on the Action: Your last action "execute/find_a_cave" has been executed. The executor has attempted to execute the action according to your prompt. You should check whether your intention has been fulfilled. Your inventory: Now your inventory has 1 wooden_sword, 1 oak_sapling, 1 crafting_table, 3 sticks, 1 oak_log. Equipped Item: Now you hold the wooden_sword in your hand. Location and Orientation: Now you locate in X: 538.50, Y: 63.90, Z: -120.27, Pitch: 33.51, Yaw: 408.51. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking at a swamp biome with water, lily pads, and grass. In the distance, there is a cave entrance visible with some sand flowing out of it. There are also some sheep visible on the right side of the image. THOUGHT: The cave entrance in the distance is a promising location to find spiders, which can drop string when defeated. Since I am equipped with a wooden sword, I am prepared to fight any spiders I encounter. I should head towards the cave and look for spiders inside or around the entrance. ACTION: execute/move_to_cave_entrance

Round 13 [Env] Feedback on the Action: Your last action "execute/move_to_cave_entrance" has been executed. The executor has attempted to execute the action according to your prompt. You should check whether your intention has been fulfilled. Your inventory: Now your inventory has 2 lily_pad, 1 oak_sapling, 1 crafting_table, 2 rotten_flesh, 3 sticks, 1 wooden_sword, 1 oak_log. Equipped Item: Now you hold the wooden_sword in your hand. Location and Orientation: Now you locate in X: 581.75, Y: 68.00, Z: -47.65, Pitch: 21.41, Yaw: 354.63. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking at a swamp biome with various trees and grass. The player is standing on a block with a snowy texture, which could be either snow or a white wood block. The player's health bar and hunger are visible, with a wooden sword equipped in the player's hand. THOUGHT: The player seems to have moved from the previous location but does not appear to be at the cave entrance yet. Since the goal is to find spiders to obtain string for the fishing rod, I should continue to move towards the cave entrance that was previously spotted. Once there, I can search for spiders inside or near the cave. ACTION: execute/move_to_cave_entrance

Round 14 [Env] Feedback on the Action: Your last action "execute/move_to_cave_entrance" has been executed. The executor has attempted to execute the action according to your prompt. You should check whether your intention has been fulfilled. Your inventory: Now your inventory has 2 lily_pad, 1 oak_sapling, 1 crafting_table, 2 rotten_flesh, 3 sticks, 1 wooden_sword, 1 oak_log. Equipped Item: Now you hold the wooden_sword in your hand. Location and Orientation: Now you locate in X: 443.46, Y: 63.04, Z: -64.78, Pitch: 46.22, Yaw: -228.66. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking at the surface of a body of water. The player is partially submerged, as indicated by the water overlay on the screen. The player's health bar and hunger are visible, with a wooden sword equipped in the player's hand, and there are three sticks and two lily pads in the hotbar. There is no immediate sign of spiders or string. THOUGHT: The player is in the water, which is not an ideal location for spiders. I need to get back on land and continue searching for spiders to obtain string. Since it is still daytime, I should look for shaded areas under trees or other dark spots where spiders might be hiding. ACTION: execute/move_to_land

Round 15 [Env] Feedback on the Action: Your last action "execute/move_to_land" has been executed. The executor has attempted to execute the action according to your prompt. You should check whether your intention has been fulfilled. Your inventory: Now your inventory has 2 lily_pad, 1 oak_sapling, 1 crafting_table, 2 rotten_flesh, 3 sticks, 1 wooden_sword, 1 oak_log. Equipped Item: Now you hold the wooden_sword in your hand. Location and Orientation: Now you locate in X: 432.30, Y: 66.02, Z: 74.21, Pitch: -16.11, Yaw: 188.90. [Agent] OBSERVATION: The image shows the player's first-person view in Minecraft, looking at a dirt block wall, likely part of a hill or mountain. The player's health bar and hunger are visible, with a wooden sword equipped in the player's hand, and there are three sticks and two lily pads in the hotbar. The player's health is slightly reduced, and there are two pieces of rotten flesh, which suggests a recent combat encounter, possibly with a zombie. THOUGHT: The player seems to have been in combat recently, as indicated by the rotten flesh in the inventory, which is dropped by zombies. My goal is to obtain string from spiders to craft a fishing rod. Since the current view is obstructed by the dirt wall, I need to move to an open area to continue searching for spiders. ACTION: execute/move_out_of_the_cave

Wander for another 25 rounds without finding a spider...

Figure 16: This is an example of gpt-4o failing to complete a VAB-Minecraft task. Task instruction: "Your task is to get a fishing_rod in your inventory." gpt-4o didn't notice the spiders near the spawn point and couldn't find any other spiders elsewhere. Instead of wandering around (actions marked as red), it should "teleport_to_spawn" to find spiders.

2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483

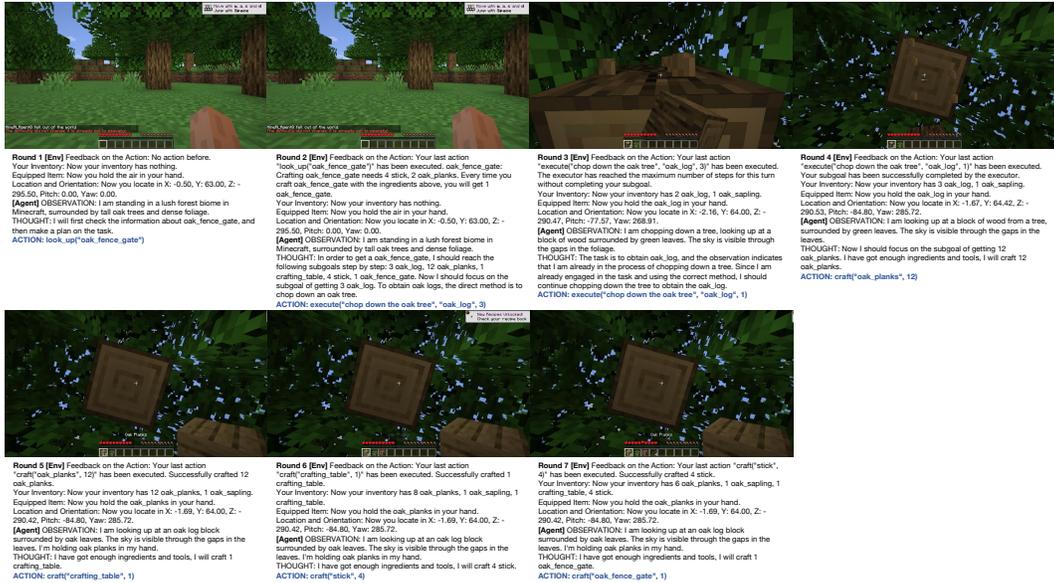


Figure 17: This is an example of InternVL-2 successfully completing a VAB-Minecraft task. Task instruction: "Your task is to get a oak_fence_gate in your inventory."

2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537



Figure 18: This is an example of InternVL-2 failing to complete a VAB-Minecraft task. Task instruction: “Your task is to get a wooden shovel in your inventory.” InternVL-2 made a wrong plan for it (marked as red in Round 2), and persistently tried to obtain sticks by chopping down trees, which is infeasible (actions marked as red from Round 7 to 16).

G.3 VAB-ANDROIDLAB CASES

In this section, we show successful and failed examples of $gpt-4o$ in Figure 19 and Figure 20.

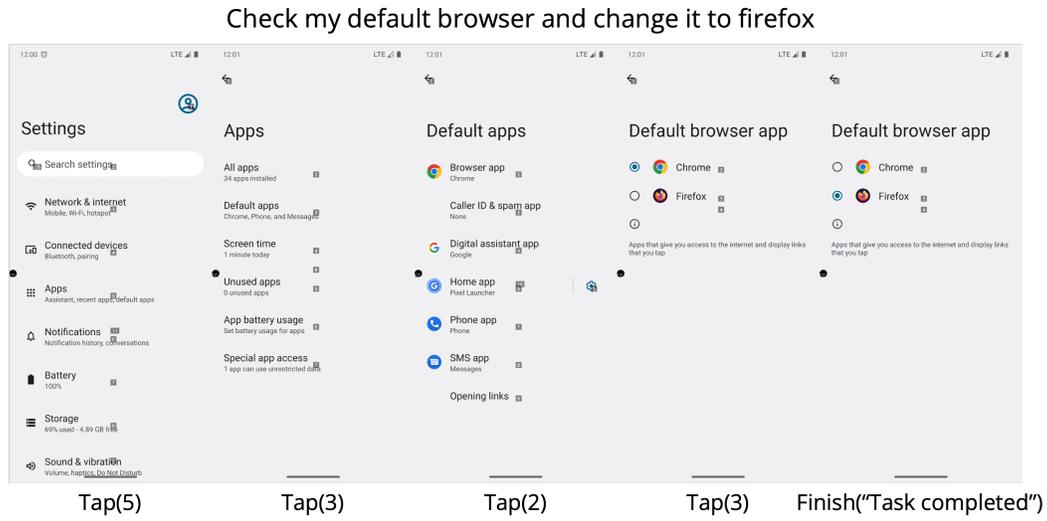


Figure 19: This is an example of $gpt-4o$ successfully completing a VAB-AndroidLab task. Task instruction: “Check my default browser and change it to firefox”

2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645

Check the walking distance and time between Bus Stop of Stanford Campus Oval and Bus Stop of Oxford Street & University Avenue

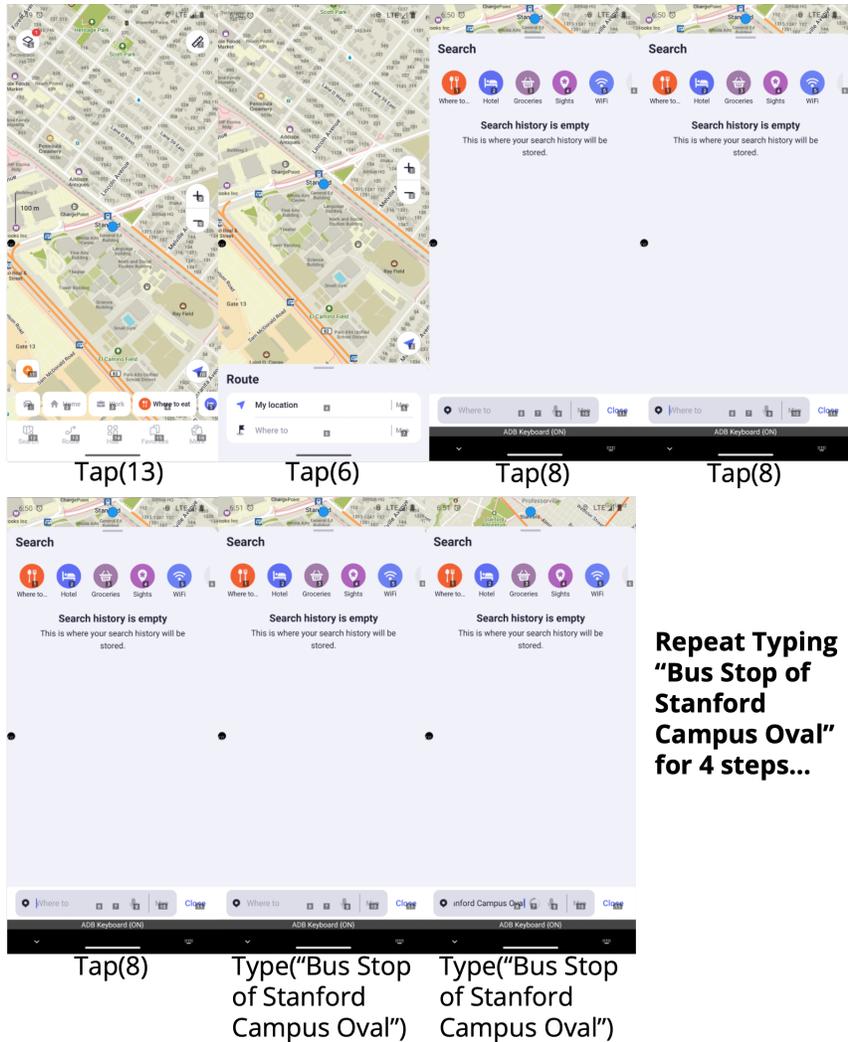


Figure 20: This is an example of `gpt-4o` failing to complete a VAB-AndroidLab task. Task instruction: "Check the walking distance and time between Bus Stop of Stanford Campus Oval and Bus Stop of Oxford Street & University Avenue" `gpt-4o` repeated doing typing without noticing the success of previous typing.

2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699

G.4 WEBARENA-LITE CASES

In this section, we show successful and failed examples of `gpt-4o` in Figure 21 and Figure 22.

What is the phone number of Western Pennsylvania Hospital

<p>Let's think step-by-step. The objective is to find the phone number of Western Pennsylvania Hospital. This requires a search action to locate the hospital on the map. I will type "Western Pennsylvania Hospital" into the search input box whose ID is [15] and press Enter to perform the search.</p> <p>In summary, the next action I will perform is <code>type [15] [Western Pennsylvania Hospital] [1]</code></p>	<p>Let's think step-by-step. The objective is to find the phone number of Western Pennsylvania Hospital. Currently, the search results show the hospital's address and some additional links. To find more information, including the phone number, I should follow the link to more results.</p> <p>In summary, the next action I will perform is <code>click [21]</code>.</p>	<p>Let's think step-by-step. The current objective is to find the phone number of Western Pennsylvania Hospital.</p> <p>Currently, the search results indicate two different hospital-related entries. Since the goal is to find information specific to the Western Pennsylvania Hospital, I need to investigate the link that corresponds to this hospital.</p> <p>In summary, the next action I will perform is to click the link corresponding to the Western Pennsylvania Hospital to view more details.</p> <p>In summary, the next action I will perform is <code>click [20]</code>.</p>	<p>Let's think step-by-step. The objective is to find the phone number of the Western Pennsylvania Hospital. From the observations, there is an element labeled "phone" which is followed by the phone number of the hospital. The phone number is +1 412 578 5000.</p> <p>I believe I have achieved the objective. I will issue the stop action with the answer.</p> <p>In summary, the next action I will perform is <code>stop [+1 412 578 5000]</code></p>
---	---	---	--

Figure 21: This is an example of `gpt-4o` successfully completing a WebArena-Lite task. Task instruction: “What is the phone number of Western Pennsylvania Hospital”

2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753

Promote auth0/angular-storage to subreddit technology with the description from the repo itself.

Let's think step-by-step. To promote the 'auth0/angular-storage' library to the 'technology' subreddit, I'll first need to click the "Submit" button and then fill in the necessary details. In summary, the next action I will perform is ""click [5]""

Let's think step-by-step. The objective is to promote 'auth0/angular-storage' to the subreddit 'technology' with the description from the repository itself. Here are the steps: 1. Enter the URL of the 'auth0/angular-storage' repository. 2. Enter a title for the post. 3. Enter the body with the description from the repository. 4. Select the subreddit 'technology'. 5. Submit the post. I'll start by entering the URL in the URL field. The URL is "https://github.com/auth0/angular-storage". In summary, the next action I will perform is ""type [9] [https://github.com/auth0/angular-storage]""

Let's think step-by-step. The objective is to promote 'auth0/angular-storage' to the subreddit 'technology' with the description from the repository itself. The next step is to add a title and body to the submission. First, I will set the title. Then I will add the description to the body. Lastly, I will select the forum (subreddit) as 'technology'. In summary, the next action I will perform is ""type [10] [Auth0 Angular Storage]""

Let's think step-by-step. The objective is to promote "auth0/angular-storage" to the subreddit "technology" with the description from the repository itself. I need to:
1. Provide the URL to the repository in the designated field.
2. Select the subreddit "technology".
3. Add the description from the repository itself to the body section.
First, let's input the URL to the repository. The field for the URL has the ID [9].
In summary, the next action I will perform is ""type [9] [https://github.com/auth0/angular-storage]""

Let's think step-by-step. To promote the repository to the subreddit technology, the steps are:
1. Provide the title for the post.
2. Provide the description in the body section.
3. Select the appropriate subreddit ("technology" in this case).
4. Submit the post.
First, let's start by entering the title. The title will be "Auth0 Angular Storage". I will type this into the title field identified by [10].
In summary, the next action I will perform is ""type [10] [Auth0 Angular Storage]""
(Start Repeat Doing.....)

Figure 22: This is an example of gpt-4o failing to complete a WebArena-Lite task. Task instruction: "Promote auth0/angular-storage to subreddit technology with the description from the repo itself." gpt-4o repeated doing typing without noticing the success of previous typing.

2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807

G.5 VAB-CSS CASES

We show successful and failed cases of both `gpt-4o` and `InternVL-2` in Figure 23, Figure 24, Figure 25 and Figure 26.



Figure 23: This is an example of `gpt-4o` successfully completing a VAB-CSS task.

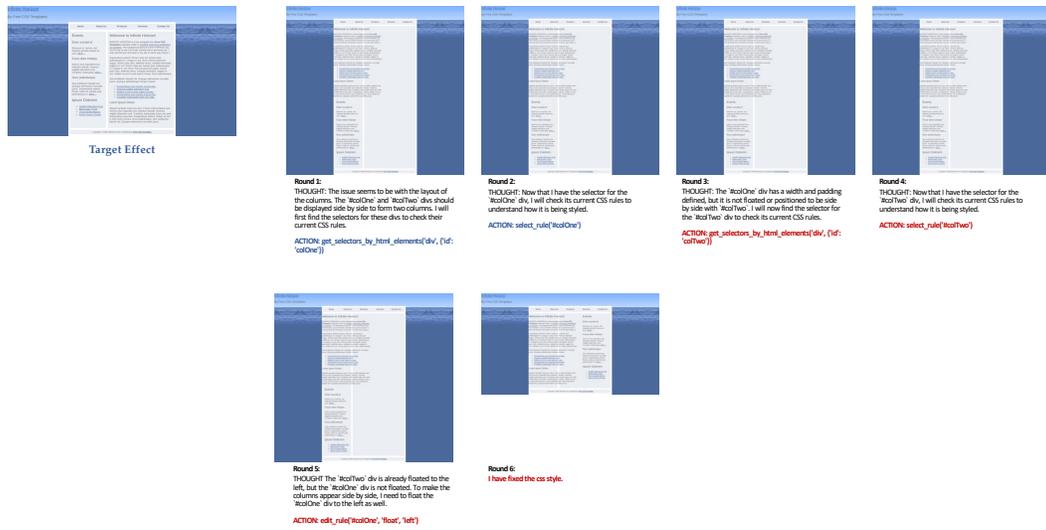


Figure 24: This is an example of `gpt-4o` failing to complete a VAB-CSS task.

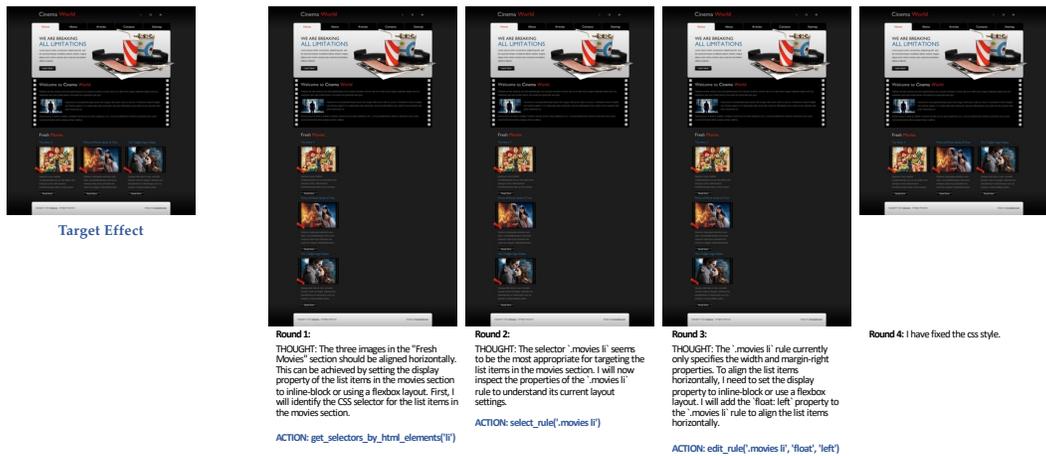


Figure 25: This is an example of `InternVL-2` successfully completing a VAB-CSS task.

2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861



Figure 26: This is an example of InternVL-2 failing to complete a VAB-CSS task.

G.6 ERROR RECOVERY CASES

In this section, we show examples of error recovery in each environment, as illustrated in Figure 27, Figure 28, Figure 29, Figure 30, Figure 31.



Figure 27: In VAB-OmniGibson, the agent encounters a failure when attempting to place the document into a closed bin. Upon receiving feedback from the environment, the agent first opens the bin and successfully deposits "2.document" into it.

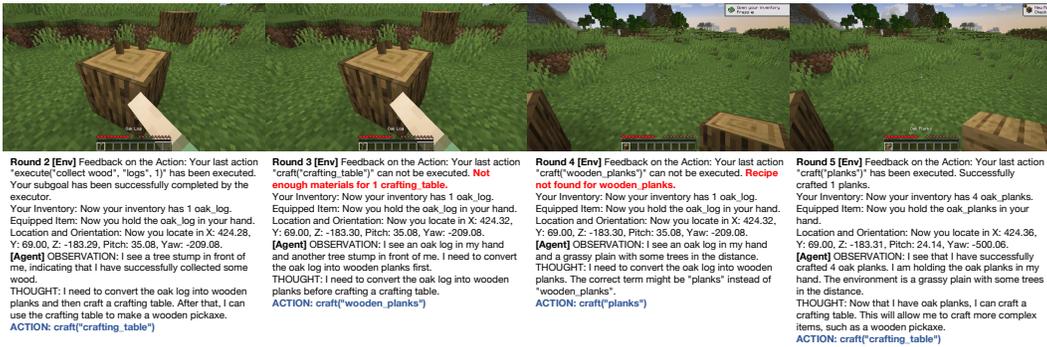
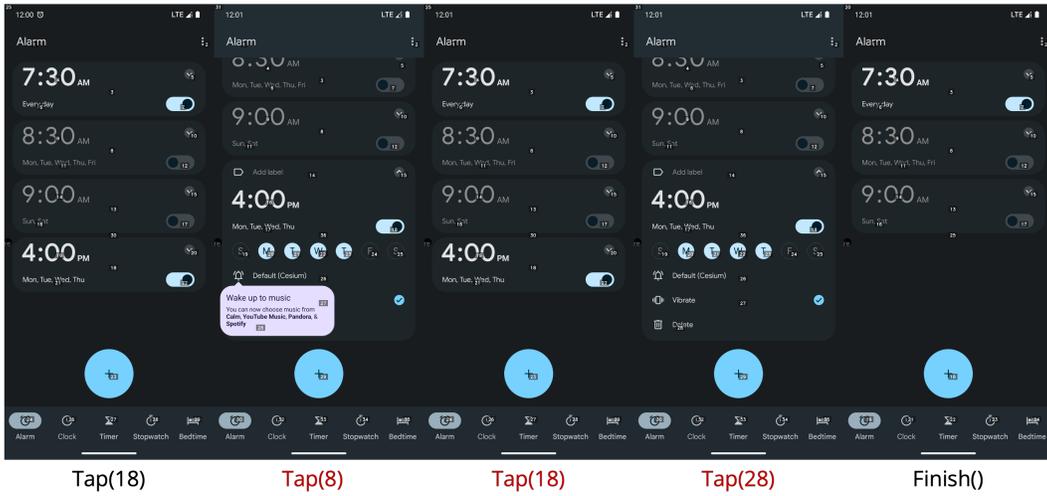


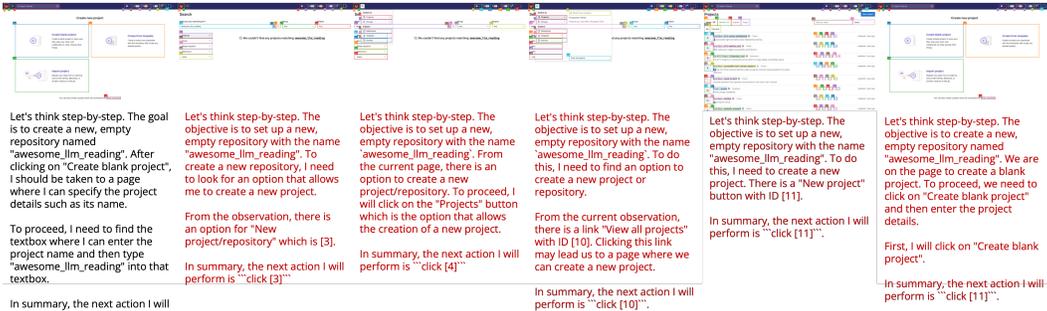
Figure 28: In VAB-Minecraft, the agent initially attempts to craft a crafting_table without the required ingredients. Upon receiving feedback, the agent first attempts to craft "wooden_planks", which is an incorrect ingredient name. Subsequently, it corrects its approach by crafting "planks" and is then able to successfully craft the crafting_table.

2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930



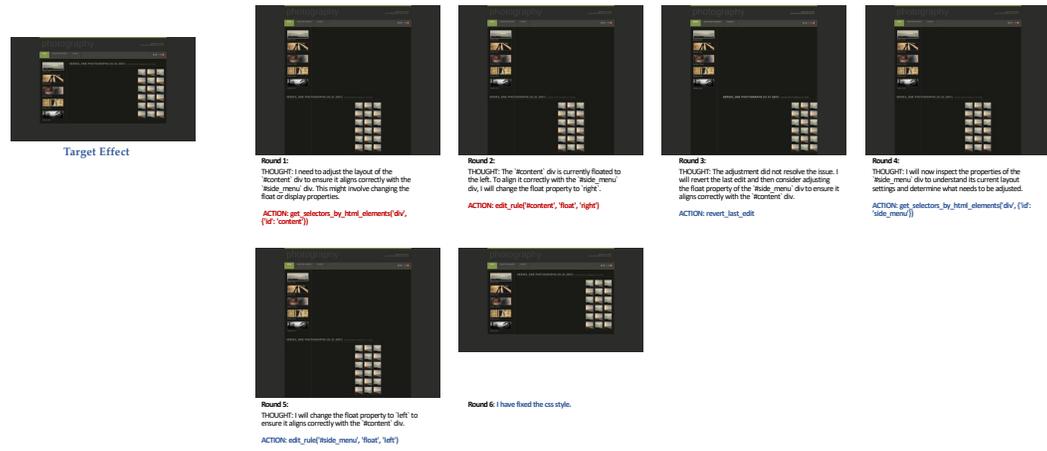
2931 Tap(18) Tap(8) Tap(18) Tap(28) Finish()
2932
2933 Figure 29: In VAB-AndroidLab, the agent initially clicks on the correct alarm to be deleted. However,
2934 a popup unexpected appears and trigger a wrong clicking of the agent. After the wrong action, it
2935 realizes the error and again perform the correct action.
2936

2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948



2949 Figure 30: In WebArena-Lite, the agent initially searches for the target repository that need it to
2950 create. After realizing the problem, it struggles to return to the repository creating page to finally
2951 perform the correct action.
2952

2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969



2967 Figure 31: In VAB-CSS, the agent initially edits a wrong CSS rule. After rendering the new webpage,
2968 the agent realizes that this is not the correct change to make so it decides to revert the last edit and
2969 makes a new edit that finally fixes the layout.