StarDojo: Benchmarking Open-Ended Behaviors of Agentic Multimodal LLMs in Production–Living Simulations with Stardew Valley

Weihao Tan[1]*, **Changjiu Jiang**[1]*, **Yu Duan**[1]*, **Mingcong Lei**[2], **Jiageng Li**[1], **Yitian Hong**[3], **Xinrun Wang**[4], **Bo An**[1]

[1] Nanyang Technological University, Singapore [2] The Chinese University of Hong Kong, Shenzhen
[3] East China University of Science and Technology [4] Singapore Management University
weihao001@ntu.edu.sg boan@ntu.edu.sg
Project website: https://weihaotan.github.io/StarDojo

Abstract

Autonomous agents navigating human society must master both production activities and social interactions, yet existing benchmarks rarely evaluate these skills simultaneously. To bridge this gap, we introduce StarDojo, a novel benchmark based on Stardew Valley, designed to assess AI agents in open-ended production–living simulations. In StarDojo, agents are tasked to perform essential livelihood activities such as farming and crafting, while simultaneously engaging in social interactions to establish relationships within a vibrant community. StarDojo features 1,000 meticulously curated tasks across five key domains: farming, crafting, exploration, combat, and social interactions. Additionally, we provide a compact subset of 100 representative tasks for efficient model evaluation. The benchmark offers a unified, user-friendly interface that eliminates the need for keyboard and mouse control, supports all major operating systems, and enables the parallel execution of multiple environment instances, making it particularly well-suited for evaluating the most capable foundation agents, powered by multimodal large language models (MLLMs). Extensive evaluations of state-of-the-art MLLMs agents demonstrate substantial limitations, with the best-performing model, GPT-4.1, achieving only a 12.7% success rate, primarily due to challenges in visual understanding, multimodal reasoning and low-level manipulation. As a user-friendly environment and benchmark, StarDojo aims to facilitate further research towards robust, open-ended agents in complex production-living environments. Code and demos can be found at https://stardojo2025.github.io/stardojo.

1 Introduction

The complexity of human society is embodied in "**production-living systems**" [9], where individuals simultaneously engage in resource-generating activities (like farming, manufacturing, and crafting) while participating in social exchanges, cultural practices, and adaptive responses to environmental changes. These systems represent the fundamental ways humans navigate the world, requiring constant balancing between productive output, resource management, social integration, and environmental adaptation. Developing human-level agentic multimodal LLMs (MLLMs) that can navigate production-living systems represents a crucial frontier in AI research, moving beyond narrow task completions such as question answering [1], code generation [18] and math reasoning [36], towards

*Equal contribution

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Multi-Turn Interactions in Large Language Models.

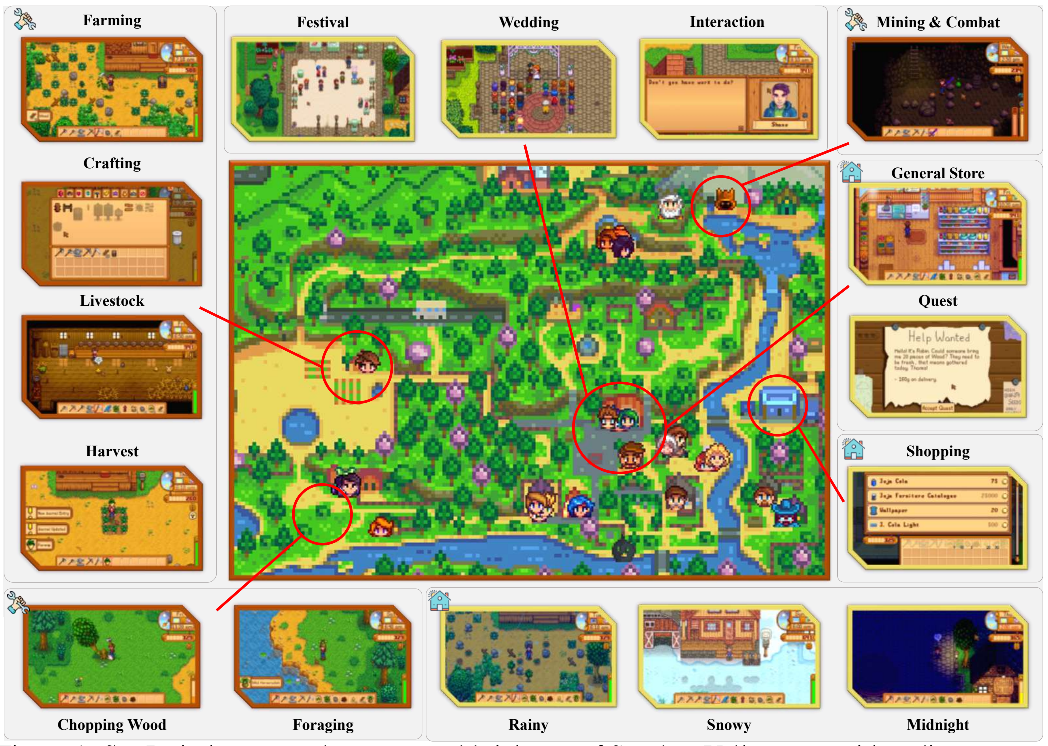

Figure 1: StarDojo leverages the open-world richness of Stardew Valley to provide a diverse array of scenarios and activities. Agents are required not only to engage in a wide range of production activities, such as farming, crafting, mining, logging, and animal husbandry, but also to participate in various social events, including trading, conversing with NPCs, and even starting families. In addition, agents need to adapt to dynamic changes in time and weather, thereby reflecting the multifaceted nature of real-world living and social interaction.

systems that can handle the interplay between productive activities and social dynamics. This comprehensive capability is essential for Artificial General Intelligence (**AGI**), as it encompasses the holistic integration of cognitive, practical, and social intelligence that has enabled human thriving across diverse environments. However, the capability is seldom evaluated by existing interactive benchmarks such as Atari [5], PySC2 [37], MineDojo [8] and OSWorld [43].

To bridge the gap, we introduce StarDojo, a novel environment and benchmark consisting of 1,000 comprehensive tasks based on the production-living dynamics of the popular simulation game, Stardew Valley. As illustrated in Figure 1, agents undertake tasks such as clearing farmland, tilling soil, planting and watering crops, harvesting produce, raising animals, mining and foraging for resources, and crafting essential tools and equipment. Additionally, agents must explore diverse maps, collect various items, combat monsters, trade with merchants to generate income, upgrade and expand their farms, and complete numerous in-game quests. Social progress involves participating in festivals and community events, building interpersonal relationships, and may eventually lead to marriage and raising children. The environment realistically simulates time, stamina, weather patterns, and seasonal cycles, all of which significantly impact both production tasks and social interactions, presenting further challenges for agents. To facilitate development and evaluation for researchers, we also present StarDojo-Lite, a curated subset comprising 100 core tasks that focus on the essential skills typically encountered during the game's early stages.

To facilitate the development and evaluation of agent behaviors, StarDojo offers four key features: 1) **Unified User-friendly Interface**: Provides an intuitive Python interface to interact seamlessly with Stardew Valley's game engine implemented in C#, simplifying interaction and internal state capture, eliminating the need for manual screenshots and keyboard/mouse inputs to obtain observation and action. 2) **Automated Evaluation**. Includes comprehensive evaluation scripts for all tasks, ensuring reliable and reproducible agent performance assessments. 3) **System Compatibility**. Supports major operating systems (Ubuntu, macOS, and Windows) to ensure wide accessibility. 4) **Parallelized Environments**. Allows multiple headless environment instances to run concurrently, significantly enhancing efficiency for evaluation and data collection.

Through extensive evaluations, we demonstrate that tasks within StarDojo present significant challenges even for agents with state-of-the-art MLLMs. Our assessments cover cutting-edge models, including GPT-4.1 (& mini) [22], Claude-3.7 Sonnet [3], Gemini 2.5 Pro [10], and the open-source Llama 4 Maverick [20], Qwen2.5-VL-72B [4] and Gemma 3 27B [34]. These agents achieve performance ranging from 4% to 12.7% on the StarDojo-Lite task suite. While agents successfully complete some easy-level tasks requiring fewer than 30 steps, they exhibit near-zero success rates on medium and hard tasks demanding extended action sequences. Additionally, tasks involving long-term navigation and combat are particularly challenging. We observed that even advanced models struggle significantly with accurately identifying crucial visual elements, such as character locations, entrances, trees, NPCs, and crops, which makes them difficult to determine whether the object is already within the reach of interaction. This makes it hard for them to tell whether something is close enough to interact with, which in turn prevents them from finishing tasks that require moving across different areas or going in and out of buildings, impeding their ability to engage in comprehensive long-term planning. We release StarDojo as an open-source environment and benchmark, providing setup instructions, robust evaluation scripts, comprehensive documentation, and baseline implementations. We hope it can facilitate research into robust, open-ended decision-making agents capable of lifelong learning and long-term planning in production-living systems.

Table 1: Comparison of StarDojo with representative existing environments. The columns indicate whether the environment supports open-ended interaction and continuous learning, evaluates the ability to perform long-term planning, reflects scenarios relevant to real-world daily life, includes production activities (e.g., farming, hunting, and crafting), simulates human society with social interactions, implements a realistic economy supporting production and social activities, and provides language-based APIs for interaction with LLMs.

Environment	Open-Endness	Long-Term Planning	Daily Life	Production Activities	Human Society	Economy System	Language APIs
Atari [5]	×	×	×	×	×	×	×
VirtualHome [26]	×	×	√	×	×	×	×
SMAC [29]	×	×	×	×	×	×	×
TextWorld [7]	×	×	√	×	×	×	√
Smallville [24]	√	√	√	×	√	×	√
OSWorld [43]	×	√	√	×	×	×	√
Crafter [13]	√	×	×	√	×	×	×
MineDojo [8]	√	√	×	√	×	×	√
CivRealm [28]	√	√	×	√	√	√	√
StarDojo	√	√	√	√	√	√	√

2 Related Work

Benchmarks for MLLMs Agents. The development of robust benchmarks for evaluating decision-making agents across various scenarios has been a critical focus in AI research. Traditional reinforcement learning (RL) benchmarks [5, 35, 11, 29] predominantly emphasize low-level control tasks and non-realistic environments. More realistic simulators [6, 16, 17, 27, 26], usually focus on embodied tasks in household scenarios, lacking significant environmental dynamics and diverse social activities. Recent advancements in generative agents have enabled large language models (LLMs) to simulate human social behaviors in interactive environments [24, 2, 7]. Their limited action spaces, primarily restricted to dialogue, hinder engagement in broader, real-world-inspired tasks that involve production, consumption, and resource management. On the other side, while GUI benchmarks [31, 48, 15, 43] also provide interactive environments, they focus on short-term software manipulations. The most relevant work is MineDojo [8], which offers a diverse range of tasks within the open-ended environment of Minecraft. However, its gameplay primarily focuses on interactions with nature, with limited opportunities for human-like social interactions. Additionally, its complex 3D navigation controls pose significant challenges, particularly for LLM-based agents to complete even simple tasks, further limiting its suitability for more complex activities. Another relevant benchmark, CivRealm [28], evaluates agents' strategic decision-making at a country level within a turn-based, Civilization-like game. While CivRealm presents a variety of tasks including managing population, production, and economy, its scope remains at a macro-strategic level, distinct from StarDojo's focus on granular, individual-level decision-making. Additionally, recent works [23, 47] tend to evaluate MLLMs on traditionally RL environments in simple game scenarios, lacking semantic richness and social interaction.

As shown in Table 1, StarDojo addresses the limitations of prior benchmarks by providing a comprehensive, open-ended evaluation platform for decision-making agents in a dynamic environment.

StarDojo allows for rich, multimodal interactions that encompass farming, trading, crafting, exploration, and social relationships. Its unique combination of complexity, realism, and diversity makes it a valuable testbed for advancing agents in real-world-like environments.

MLLMs Agents. Traditional reinforcement learning (RL) agents [21, 19, 30, 12] primarily focus on low-level control and fail to leverage natural language understanding, making them unsuitable for tasks requiring complex reasoning, long-term planning, and social interactions. Recent advancements in LLM-based agents have significantly expanded AI capabilities by integrating reasoning mechanisms such as chain-of-thought (CoT) prompting [44] and reflection [32]. Modular frameworks and multi-agent architectures have enabled LLM-based agents to achieve remarkable performance in tasks like code generation [14, 41, 40] and GUI manipulation [46, 45, 42, 39]. Additionally, Voyager [38] has demonstrated strong in-context lifelong learning abilities, showcasing exceptional proficiency in the open-ended world of Minecraft. However, Voyager's strong reliance on built-in APIs makes it challenging to adapt to other games. Cradle [33] successfully completes meaningful tasks across multiple commercial video games and software applications with a unified interface without the need to access APIs. Its preliminary experiments in Stardew Valley highlight the limitations of current state-of-the-art agents, particularly in handling multi-modal understanding, long-term planning, and resource management, which reveals the importance of extending Stardew Valley as a well-developed benchmark for decision-making agents.

3 StarDojo

3.1 Introduction to Stardew Valley

Stardew Valley is an open-ended simulation RPG game where players inherit a run-down farm. Players must thoughtfully manage their farming strategies, explore the surrounding village, and gather diverse resources to revitalize the farm. Players are encouraged to build meaningful relationships with local villagers and participate in community events. More details can be found in Appendix B.

Realistic Dynamics. Each in-game day in Stardew Valley lasts from 6 AM to 2 AM. When night falls, outdoor activities are affected by the darkness. Staying awake past midnight will result in penalties. Players start each day with 270 energy points, spent through activities like farming and mining, and restored by sleeping or eating. The game features four 28-day seasons and different daily weather conditions, each affecting farming, forage and other events. Effective management of time and energy is crucial to maximizing productivity and overcoming the game's primary challenges.

Rich Production Activities. Various production activities form the core gameplay of Stardew Valley. Players perform essential tasks such as clearing debris, chopping wood, tilling soil, planting, watering, and harvesting crops, as well as raising livestock like cows and chickens. Other activities include fishing at the beach, mining and combat in the mines, and foraging in the forest. Engaging in these tasks gradually improves character skills, unlocking more than 100 crafting recipes. Crafting allows players to create useful tools, machinery, and decorative items, significantly enhancing farm productivity and exploration efficiency.

Diverse Social Interaction. Beyond farm production, Stardew Valley features 45 unique NPCs, each with distinct personalities, daily routines, and special heart events. Players can build friendships by offering gifts, and may even date, marry, and raise children with villagers. Periodically, the town hosts festivals and special events, such as the Egg Festival and Stardew Valley Fair, which provide valuable opportunities to strengthen community relationships. Additionally, various quests guide players to explore the village, interact with citizens and collect valuable resources. Additionally, the game's comprehensive economic system requires strategic resource management, investment, and efficient planning to generate income from production activities, adapting strategically to seasonal demands and market conditions to achieve long-term financial success.

Overall, Stardew Valley serves as an ideal environment for decision-making agents in the production–living simulation. Its well-integrated systems of time management, resource allocation, economic planning, and social interaction provide a dynamic and complex environment that requires strategic thinking and adaptability. The game's structured yet open-ended nature makes it an excellent testbed for evaluating decision-making capabilities in simulated real-world conditions.

3.2 Architecture

As a typical commercial video game, Stardew Valley only supports human-like interaction, e.g., observing gameplay through screenshots and using keyboard and mouse to control. Additionally, the game window must remain active and in the foreground, significantly restricting automated gameplay and preventing simultaneous execution of multiple instances. As shown in Figure 2, to

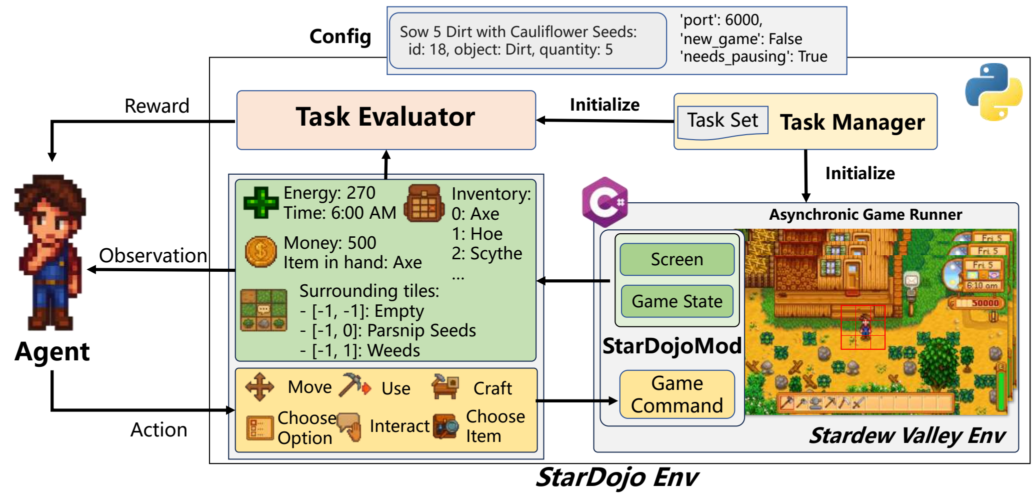

Figure 2: StarDojo environment is initiated by configurable task files. It communicates with parallel game engines through StarDojoMod to obtain internal game states and execute commands, which will be encapsulated as observations and actions by the Python Wrapper.

overcome these limitations, we introduce the carefully designed StarDojo environment, enabling efficient interaction and comprehensive evaluation of agents.

Unified User-friendly Interface. We present StarDojoMod, a novel extension built upon the Stardew Modding API (SMAPI)[25], which is a widely adopted, open-source modding framework designed specifically for Stardew Valley. SMAPI offers developers extensive APIs that expose key game events and internal states, facilitating the creation of interactive and sophisticated mods. Based on SMAPI, StarDojoMod provides structured and efficient interactions between agents and the game environment. It communicates in real-time with the Stardew Valley game engine through a socket server, granting agents direct access to rendered gameplay images, saving the time-consuming screen captures, internal game states (such as character positions, statuses, and environmental information), and enabling diverse callable functions as action skills beyond traditional keyboard and mouse inputs. Moreover, we implemented a configurable pause-and-resume mechanism by directly modifying the inner states of the game, allowing the game to pause during model inference and agent planning, and resume before action execution. Inherited from SMAPI, StarDojoMod is implemented in C# to be consistent with the game engine. To enhance ease-of-use and accessibility of the environment, we provide a user-friendly Python Wrapper based on the StarDojoMod for observation retrieval, action execution, and task customization, empowering users to engage with the StarDojo environment effortlessly.

System Compatibility. Stardew Valley is one of the few games that can be played on all mainstream operating systems (Linux, macOS and Windows). We also ensured the compatibility of StarDojoMod and the Python Wrapper, enabling the entire environment to run seamlessly across different systems.

Parallel Execution. Our architecture is designed for scalability and parallel execution. Each instance of Stardew Valley is independently managed through unique ports, enabling simultaneous control of multiple game instances without interference. Communication efficiency is further enhanced through the use of shared memory, reducing observation retrieval time to as little as 30 ms. Furthermore, StarDojoMod supports headless operation through the X Virtual Framebuffer (Xvfb), enabling compatibility with Linux systems without graphical interfaces, thus broadening accessibility across diverse hardware and system configurations.

3.3 Observation and Action Spaces

Observation Space. StarDojo offers a comprehensive observation space that integrates both visual and textual modalities to accommodate a wide range of agent architectures. Each observation includes a gameplay screenshot alongside detailed textual information describing the game state. This textual state contains character status (such as health and energy), local tile information ($n \times n$ tiles surrounding the agent), and global information (such as time, weather, and the positions of NPCs and buildings). Recent work [33] highlights the difficulties that current MLLMs encounter when interpreting Stardew Valley's distinctive art style and meeting its precise manipulation demands. By combining visual and textual observations, StarDojo ensures that agents can leverage both high-level context and fine-grained environmental cues for more robust and informed decision-making. For our

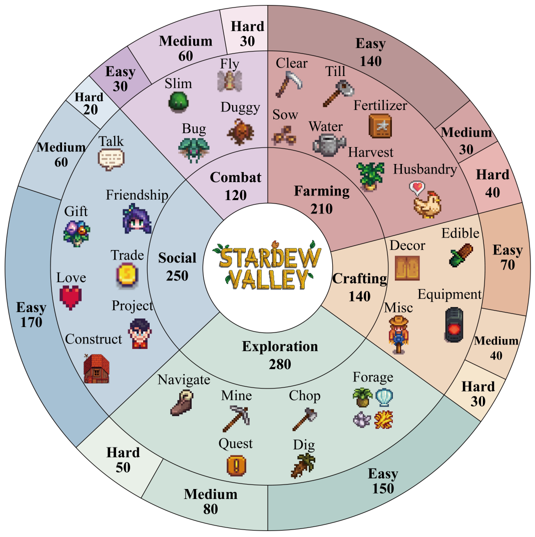

Figure 3: Distribution of 1000 tasks across five categories: Farming, Crafting, Exploration, Combat and Social in StarDojo, each with Easy, Medium, and Hard difficulties.

Table 2: Task statistics of StarDojo-Lite. The task suite is made up of the most representative early-stage tasks from each category.

Category	Easy	Medium	Hard	Total
Farming	14	3	4	21
Crafting	7	4	3	14
Exploration	15	8	5	28
Combat	3	6	3	12
Social	17	6	2	25
Total	56	23	21	100

experiments, we selected a subset of this information to fairly evaluate agent behaviors. Full details of the observation space are provided in Appendix C.1.

Action Space. The action space in StarDojo is designed to encompass the full range of activities that can be performed in the original Stardew Valley using a keyboard and mouse. Actions are carefully abstracted to eliminate redundant operations while retaining the core decision-making challenges inherent to the game. We define ten fundamental actions, which together are sufficient to cover most gameplay scenarios: *move(x, y)*, *use(direction)*, *interact(direction)*, *choose_item(slot_index)*, *attach_item(slot_index)*, *detach_item()*, *craft(item)*, *choose_option(option_index, quantity, direction)*, *menu(option, menu_name)* and *navigate(name)*. In our experiments, we excluded the *navigate* action from the available action space. While *navigate* provides a high-level shortcut for moving between maps, our primary objective is to evaluate the agent's realistic exploration abilities and its performance using an action space that more closely mirrors human interactions. Detailed descriptions of the action space can be found in Appendix C.2.

3.4 Tasks

As shown in Figure 3, we carefully curate 1000 tasks to benchmark agents' various behaviors in StarDojo. These tasks are divided into five distinct categories, **Farming**, **Crafting**, **Exploration**, **Combat** and **Social**, which covers most of the production-living activities in the early and middle stages of the game. Each task is classified into three difficulties, easy, medium, and hard, with a heuristic maximum steps of 30, 50 and 150, based on their complexity and the time consumed. As shown in Table 2, to facilitate efficient agent evaluation, we curate a representative smaller task suite, called StarDojo-Lite, comprising 100 core tasks from the full task collection, balancing coverage and practicality. This lite task set covers most of the representative activities in the early stage of the game. Detailed descriptions of the task set can be found in Appendix C.3.

Playthrough. In addition to modular tasks, StarDojo also provides an extended Playthrough task designed to comprehensively assess an agent's strategic decision-making over prolonged gameplay. Starting from scratch, the agent's objective is to accumulate 1 million in-game currency, which is a significant milestone typically achieved over hundreds of in-game days. Success requires efficient seasonal planning, unlocking new maps through quests, careful resource and energy management, skill progression for advanced crafting, and strategic relationship-building with NPCs.

Initial Config and Setup. Some tasks require the agent to possess specific equipment (e.g., a sword), have certain items (e.g., sufficient crop seeds in inventory), or have completed particular game progress (e.g., unlocking the mines). To establish the initial state for each task, we provide multiple saved game files reflecting various stages of progression, along with specialized task-specific functions utilizing StarDojoMod commands. At the start of each task, StarDojo automatically loads the corresponding saved game and executes these task-specific commands, ensuring all necessary prerequisites, such as items, equipment, skill levels, date and farm status, are appropriately configured.

Table 3: Success rates(%) and standard deviation of agents with different base models on StarDojo-Lite task set, ranging over five categories (Farming, Crafting, Exploration, Combat and Social) and three levels of difficulty. Each task is evaluated over three runs.

Model	Task	Farming	Crafting	Exploration	Combat	Social	Total
GPT-4.1	**Easy**	**31.0**±4.1	**52.4**±8.3	**15.6**±3.9	**11.1**±19.3	7.8±6.8	**21.4**±1.8
	Medium	0.0±0.0	0.0±0.0	8.3±7.2	0.0±0.0	0.0±0.0	2.7±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	**20.6**±2.8	**26.2**±4.1	**10.7**±0.0	**2.8**±4.8	5.3±4.6	**12.7**±0.6
Gemini 2.5 Pro	**Easy**	26.2±4.1	47.6±8.3	6.7±6.7	0.0±0.0	**11.8**±5.9	17.9±1.8
	Medium	0.0±0.0	0.0±0.0	8.3±7.2	0.0±0.0	0.0±0.0	2.7±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	17.5±2.8	23.8±4.1	6.0±2.1	0.0±0.0	**8.0**±4.0	10.7±0.6
Claude 3.7 Sonnet	**Easy**	**31.0**±4.1	28.6±0.0	8.9±10.2	0.0±0.0	7.8±3.4	16.1±4.7
	Medium	0.0±0.0	0.0±0.0	**16.7**±7.2	0.0±0.0	0.0±0.0	**5.3**±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	**20.6**±2.8	14.3±0.0	9.5±5.5	0.0±0.0	5.3±2.3	10.3±2.5
GPT-4.1 mini	**Easy**	26.2±4.1	4.8±8.3	4.4±7.7	**11.1**±19.3	7.8±3.4	11.3±2.7
	Medium	0.0±0.0	0.0±0.0	8.3±7.2	0.0±0.0	0.0±0.0	2.7±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	17.5±2.8	2.4±4.1	4.8±5.5	**2.8**±4.8	5.3±2.3	7.0±1.7
Llama 4 Maverick	**Easy**	28.6±0.0	28.6±0.0	0.0±0.0	0.0±0.0	7.8±3.4	13.1±1.0
	Medium	0.0±0.0	0.0±0.0	4.2±7.2	0.0±0.0	0.0±0.0	1.3±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	19.1±0.0	14.3±0.0	1.2±2.1	0.0±0.0	5.3±2.3	7.7±0.6
Qwen2.5 VL 72B	**Easy**	16.7±8.3	9.5±8.3	13.3±13.3	0.0±0.0	9.8±3.4	11.9±5.5
	Medium	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	11.1±5.5	4.8±4.1	7.1±7.1	0.0±0.0	6.7±2.3	6.7±3.1
Gemma 3 27B	**Easy**	9.5±4.1	0.0±0.0	2.2±3.9	0.0±0.0	**11.8**±0.0	6.6±2.1
	Medium	0.0±0.0	0.0±0.0	4.2±7.2	0.0±0.0	0.0±0.0	1.3±2.3
	Hard	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
	Total	6.4±2.8	0.0±0.0	2.4±2.1	0.0±0.0	**8.0**±0.0	4.0±1.0

Automatic Evaluation. Given the extensive number of existing tasks and the vast potential for future additions, it is essential to establish a reusable, scalable, adaptive, and efficient evaluation mechanism. StarDojo addresses this need by implementing a general evaluation system that continuously monitors task progression and provides immediate reward feedback to agents. Specifically, after agents finish executing actions at each step, the evaluator is invoked to determine whether tasks have been successfully completed or have reached the predefined maximum number of steps. To achieve this, the evaluator maintains the previous game state information, compares it with the current state obtained via StarDojoMod, identifies incremental changes indicating progress, and accurately assesses task completion criteria based on these observed differences. By leveraging a comprehensive observation space for incremental progress tracking, our evaluation approach effectively mitigates discrepancies caused by varying initial conditions, ensuring robust adaptability across diverse tasks.

4 Empirical Studies

In this section, we benchmark the current state-of-the-art MLLM-based agents on StarDojo and provide a comprehensive study and analysis of agents' behaviors and limitations.

Agentic MLLM Baselines. We mainly evaluated seven cutting-edge MLLMs, serving as the foundation of agent, on StarDojo-Lite task set: GPT-4.1 series [22](gpt-4.1-2025-04-14 & gpt-4.1-mini-2025-04-14), Claude 3.7 Sonnet [3] (claude-3-7-sonnet-20250219), Gemini 2.5 Pro [10] (gemini-2.5-pro-preview-03-25), from the closed-source community and Llama 4 Maverick [20] (Llama-4-Maverick-17B-128E-Instruct), Qwen2.5 VL [4] (qwen2.5-vl-72b-instruct) and Gemma 3 [34] (gemma-3-27b-it) from the open-source community.

Settings. If not mentioned explicitly, all experiments are conducted under the following settings: Agents have access to both visual and textual observations for their decision-making process. Visual observations are provided at a resolution of 720P (1280×720). Textual observations include 7×7 agent-centered surrounding information and other global information like time, date and budget. In addition to receiving observations from the current timestep, agents are also provided with history information from the previous timestep, enabling agents to reflect on past states and facilitating consistency in decision-making. Agents can output at most two skills as an action to be executed sequentially. After executing all the actions, the environment is paused until the agent outputs the next action. Each task is evaluated over three runs. More details of settings are provided in Appendix D

4.1 Qualitative Analysis

As shown in Table 3, across all evaluated models, GPT-4.1 achieves the highest overall success rate of 12.7% across all tasks, while other models perform below 11%. Among open-source models, Llama 4 Maverick demonstrates the strongest performance, benefiting from its larger model size. Most successful completions are limited to easy tasks, whereas all models struggle significantly with medium and hard tasks, achieving near-zero success rates due to increased task complexity and longer sequences of required actions. Models show some proficiency in farming and crafting tasks but exhibit considerable difficulty in exploration, combat, and social interactions.

Visual Understanding. The most common failures come from agents' inability to accurately locate and approach target objects in visual observations, severely restricting subsequent manipulation and interaction. All models exhibit difficulties in reliably detecting and pinpointing target positions. They often struggle to interpret the status of tiles, even those immediately adjacent to the character, such as whether a tile is tilled, seeded, watered, or obstructed. Moreover, models consistently fail to accurately identify and locate entrances or exits of buildings and map transitions, significantly impacting tasks that involve navigating across scenarios or exploring areas beyond the current observation. These limitations directly lead to low success rates in tasks categorized under *Exploration* and *Social*. However, agents achieve comparatively better performance in *Farming* tasks, primarily because these tasks are confined to the limited farm land, where targets (e.g., wood, weeds, crops) are typically within immediate visual proximity and accessible to the character.

Multimodal Reasoning. Besides game screenshots as visual inputs, we provide additional textual observations to augment the agents' decision-making capabilities. We observed that supplementary information describing surrounding tiles significantly enhances agents' interaction with target objects. Specifically, agents initially utilize visual input for rough directional navigation towards targets until these objects become explicitly described in the textual surroundings. Subsequently, agents rely on the detailed textual descriptions for precise navigation and accurate interactions. This multimodal input enables advanced models, such as GPT-4.1, Claude 3.7 Sonnet, Gemini 2.5 Pro, and Llama 4 Maverick, to successfully complete easy-level farming tasks, such as clearing weeds, stones, or tile holes and sowing seeds. However, lower-performing models like Qwen2.5 VL and Gemma 3 frequently hallucinate when interpreting textual observations, often referencing nonexistent or inaccurately described items and statuses in their reasoning. Moreover, we identified consistent discrepancies between image-based reasoning and textual observations even in advanced models. For instance, a tree visually located several tiles away might incorrectly be interpreted as immediately accessible, despite textual observations clearly indicating an empty tile directly above the character. Such inconsistencies lead even advanced models like GPT-4.1 and Claude 3.7 Sonnet to continuously attempt interactions, such as chopping inaccessible trees.

Low-level Control. All models effectively interpret and execute basic skills, such as *move*, *interact*, *use*, and *choose_item*. They demonstrate proficiency in using appropriate tools and items for basic activities, including hoeing, sowing, chopping, watering, and planting. However, agents typically move cautiously, often advancing only a single tile per action. This overly meticulous movement frequently results in reaching the maximum allowed steps before arriving at distant targets, severely impacting performance in tasks requiring extensive navigation, such as *Exploration* and *Social*. Additionally, less capable models such as GPT-4.1 mini, Qwen2.5 VL, and Gemma 3 struggle significantly to invoke the *craft* action correctly, which directly contributes to their notably lower success rates in *Crafting* tasks.

Long-term Planning. All evaluated models show near-zero success rates on medium and hard tasks, mainly due to the complexity of these tasks, which usually involve chaining multiple easy-level subtasks. Efficient and precise long-distance navigation remains the primary bottleneck at this stage. Nevertheless, when explicitly provided with necessary resources or placed directly at essential locations (e.g., general shops), agents demonstrate potential in completing basic crafting and social interaction tasks, as evidenced by their successful performance in easy-level scenarios. However, their inability to autonomously navigate complex scenarios, such as returning home or sleeping on time, significantly restricts their ability to complete tasks spanning multiple in-game days, greatly limiting their effectiveness in long-horizon tasks.

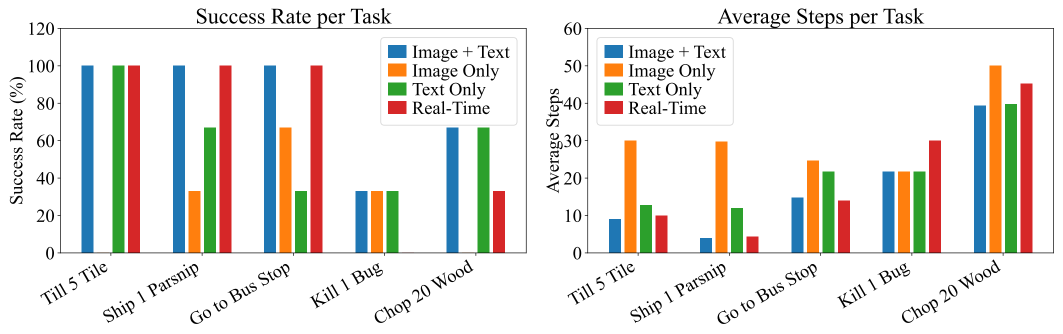

Figure 4: Ablation results of GPT-4.1-based agents on five representative tasks under four different settings: *Image + Text* (default setting), *Image Only*, *Text Only*, and *Real-Time* (without game pausing). Except for *Chop 20 Wood*, which is a medium-level task with a maximum of 50 steps, the remaining four are easy-level tasks capped at 30 steps. Each task is evaluated over three runs.

4.2 Ablation Studies

To demonstrate the flexible customization capabilities of StarDojo and enable more comprehensive evaluations of agent performance, we benchmark GPT-4.1 under the following three additional settings using five representative basic tasks. 1) **Image Only**: Agents receive only visual observations without textual state information, evaluating their capability to rely solely on visual cues. 2) **Text Only**: Agents receive textual state observations without visual input, restricting the agent's perception to local 7×7 grid-based information. This setting is particularly relevant for LLM agents. 3) **Real-time**: The environment progresses continuously during action generation, simulating real gameplay conditions where agents must plan and respond without game pausing.

As illustrated in Figure 4, removing textual input (*Image Only*) significantly affects agents' performance across all tasks, reflecting the defect of base models' poor visual-based control, emphasizing textual information's importance in grounding detailed action decisions for the current stage of agents. On the other side, eliminating visual input (*Text Only*) substantially reduces success in tasks that require navigation like *Ship 1 Parsnip* and *Go to Bus Stop*, demonstrating the essential contribution of visual cues to spatial reasoning and movement. Disabling the feature to pause the environment (*Real-time*) remarkably affects performance across tasks demanding timely reactions or prolonged action sequences. For example, in combat scenarios like *Kill 1 Bug*, the target (bug) continues moving during model inference, which can take over 10 seconds per request for GPT-4.1 via API. By the time the action is executed, the bug has often moved far from its previous position, rendering the action ineffective. Similarly, in long-horizon tasks like *Chop 20 Wood*, the in-game clock advances continuously during inference. With pausing enabled, the task may be completed in just 2 in-game hours; without pausing, it can take over 12 in-game hours, frequently pushing completion into the night or spanning multiple in-game days. These findings highlight the practical importance of real-time evaluation, an aspect often overlooked in prior benchmarks.

5 Limitations and Conclusion

Limitations. This work has several potential limitations. 1) Although StarDojo is an open-source environment and benchmark, users need an official copy of Stardew Valley to run it. 2) Fishing, an optional gameplay, is currently not included in StarDojo, due to its nature as a complex, real-time mini-game. The operations are independent of other game controls, which is not applicable to evaluate MLLMs. 3) The benchmark primarily focuses on early- and mid-game content within the main valley map. Other advanced areas, such as the Desert and Ginger Island, are not yet supported. 4) Due to budget limitations, our evaluations were conducted mainly on StarDojo-Lite with 7 models. A broader assessment across more diverse tasks and model/agent variants remains an important direction for future work.

Conclusion. Overall, we introduce StarDojo, a novel environment and benchmark designed to evaluate the open-ended behaviors of MLLM agents in Stardew Valley. StarDojo bridges the gap in existing environments by enabling comprehensive assessment of agents across both production and daily living activities within a simulated nature and society. Featuring a set of diverse tasks, StarDojo exposes significant challenges in current agents' visual understanding, multimodal reasoning, long-term planning, and real-time inference, highlighting key areas for future research and development.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Joshua Albrecht, Abraham Fetterman, Bryden Fogelman, Ellie Kitanidis, Bartosz Wróblewski, Nicole Seo, Michael Rosenthal, Maksis Knutins, Zack Polizzi, James Simon, et al. Avalon: A benchmark for rl generalization using procedurally generated worlds. *Advances in Neural Information Processing Systems*, 35:12813–12825, 2022.

[3] Anthropic. Claude 3.7 sonnet and claude code. `https://www.anthropic.com/news/claude-3-7-sonnet`, 2025. Accessed: 2025-05-10.

[4] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[5] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[6] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[7] Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer, 2019.

[8] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362, 2022.

[9] Jingying Fu, Qiang Gao, Dong Jiang, Xiang Li, and Gang Lin. Spatial–temporal distribution of global production–living–ecological space during the period 2000–2020. *Scientific Data*, 10(1):589, 2023.

[10] Google. Gemini 2.5: Our most intelligent ai model. `https://blog.google/technology/google-deepmind/gemini-model-thinking-updates-march-2025/#gemini-2-5-thinking`, 2025. Accessed: 2025-05-10.

[11] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[13] Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

[14] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[15] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

[16] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[17] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021.

[18] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

[19] TP Lillicrap. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[20] Meta. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`, 2025. Accessed: 2025-05-10.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[22] OpenAI. Introducing gpt-4.1 in the api. `https://openai.com/research/gpt-4-1`, 2025. Accessed: 2025-05-10.

[23] Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024.

[24] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22, 2023.

[25] Jesse Plamondon-Willard. SMAPI: Stardew Modding API. `https://smapi.io/`. Accessed: 2025-07-09.

[26] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.

[27] Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander William Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars and robots. *arXiv preprint arXiv:2310.13724*, 2023.

[28] Siyuan Qi, Shuo Chen, Yexin Li, Xiangyu Kong, Junqi Wang, Bangcheng Yang, Pring Wong, Yifan Zhong, Xiaoyuan Zhang, Zhaowei Zhang, et al. Civrealm: A learning and reasoning odyssey in civilization for decision-making agents. *arXiv preprint arXiv:2401.10568*, 2024.

[29] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

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

[31] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pages 3135–3144. PMLR, 2017.

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

[33] Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Gang Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering foundation agents towards general computer control. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

[34] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[35] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[36] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

[37] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

[38] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[39] Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*, 2024.

[40] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[41] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[42] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

[43] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2025.

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[45] Chaoyun Zhang, Liqun Li, Shilin He, Xu Zhang, Bo Qiao, Si Qin, Minghua Ma, Yu Kang, Qingwei Lin, Saravan Rajmohan, et al. UFO: A UI-focused agent for Windows OS interaction. *arXiv preprint arXiv:2402.07939*, 2024.

[46] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *ICLR*, 2024.

[47] Xiangxi Zheng, Linjie Li, Zhengyuan Yang, Ping Yu, Alex Jinpeng Wang, Rui Yan, Yuan Yao, and Lijuan Wang. V-mage: A game evaluation framework for assessing visual-centric capabilities in multimodal large language models. *arXiv preprint arXiv:2504.06148*, 2025.

[48] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A Appendix

B Introduction to Stardew Valley

Stardew Valley is an open-ended simulation RPG game where the player inherits a run-down farm and works to restore it. The game is presented in a 2D top-down perspective, offering a wider field of view compared to 3D games, which helps reduce operational difficulty and makes navigation smoother. The game does not have mandatory main storyline tasks and players can freely explore and live in the open-ended world.

B.1 Realistic Gameplay Mechanism

Stardew Valley is an open-ended simulation RPG game where the player inherits a run-down farm and works to restore it. The game is presented in a 2D top-down perspective, offering a wider field of view compared to 3D games, which helps reduce operational difficulty and makes navigation smoother. The game does not have mandatory main storyline tasks and players can freely explore and live in the open-ended world.

There are many mechanisms in Stardew Valley to simulate the real world, which raises additional challenges for players to plan reasonably according to these rules during the gameplay.

Time and Daily Routine. Each in-game day lasts about 14 minutes in real-time, running from 6 AM to 2 AM. Players must balance production, and social activities before exhaustion sets in. Staying up past 12 AM reduces energy the next day, and if players don't sleep by 2 AM, they will pass out and lose gold or wake up at the clinic. Nightfall typically occurs between 6 and 8 PM, depending on the season, as the outdoors gradually darkens over time.

Energy Management. At the beginning, players have 270 energy points per day. Every action, from farming to mining, consumes energy. Energy can be restored by sleeping or eating food. Strategic time and energy management are essential to maximize productivity and also the main challenge in Stardew Valley.

Seasons and Weather Effects. The game also has four seasons each lasting 28 days and affecting crop growth, fish availability, and town events. Weather varies daily, with rain saving time on watering crops, storms potentially damaging them, and snow limiting farming options. Some crops and activities are only available in specific seasons, requiring careful planning.

B.2 Production

Simulating real-world society, production activities are the main gameplay within the game, where players improve life quality through labor output.

Labor Activities. The game world is vast and diverse, featuring locations like Pelican Town, the Mines, the Beach and the forest, each offering unique activities such as mining, foraging, and fishing. Farming is central to gameplay, requiring players to plant, water, harvest crops and raise animals like cows and chickens for valuable products. Fishing and foraging provide additional resources, with seasonal variations and rare finds. Mining is crucial for gathering ores and materials, with progressively challenging levels and combat against monsters.

Skill Progression and Crafting. As players engage in these activities, they gradually improve their skills in Farming, Mining, Foraging, Fishing, and Combat. Gaining experience in each skill unlocks new crafting recipes, efficiency boosts, and profession choices that provide specialized benefits. Crafting is an essential part of progression, allowing players to create tools, machines, and decorations that enhance farm efficiency and exploration. In addition to upgrading tools and structures, they can fully customize their farm and home, arranging decorations and personalizing interiors to create their ideal living space.

B.3 Society

Besides production, engaging with the community is a core aspect of the game.

Festivals and Quests. Every season features unique festivals and events, such as the Egg Festival, Stardew Valley Fair, and Winter Star Festival. These events provide mini-games, rare items, and opportunities to strengthen relationships with villagers, adding depth to the community experience. Alongside festivals, quests play a crucial role in guiding players through different aspects of the game. NPCs also post daily requests on the town board, offering gold and friendship points for completing specific tasks. By participating in festivals and completing quests, players engage more deeply with the world, fostering a sense of purpose and progression throughout their journey.

Relationship and Friendship System. The game also provides various NPCs with unique personalities, schedules, and heart events. Players can befriend them, give gifts, and even date or marry eligible characters. Higher friendship levels unlock new interactions, cutscenes, and benefits, such as helpful spouses assisting with farm chores.

Economy System. Stardew Valley features a comprehensive economic system where players generate income through various means, including farming, fishing, mining, and crafting. Players must manage their resources, reinvest in production, and make strategic decisions to ensure financial growth. The economy fluctuates based on seasonal demand, production choices, and market conditions. Strategic planning, investment in high-value goods, and efficient time management are essential to achieving long-term prosperity. The economic system provides depth and challenges players to optimize their approach to wealth generation and sustainability.

Stardew Valley serves as an ideal benchmark for decision-making agents in Production–Living Simulation. Its well-integrated systems of time management, resource allocation, economic planning, and social interaction provide a dynamic and complex environment that requires strategic thinking and adaptability. The game's structured yet open-ended nature makes it an excellent testbed for evaluating decision-making capabilities in simulated real-world conditions.

C StarDojo Environment

Our environment is developed based on the game Stardew Valley, which offers comprehensive official mod development documentation. A large community of game enthusiasts has created and open-sourced their own mods. This provides significant convenience, as we can build a completely new mod on top of existing open-source mods to achieve the interaction between the LLM agent, RL agent, and the game.

Stardew Valley is available for purchase on Steam, and users must first buy and install the game to use our environment. Since our mod heavily relies on the official mod framework SMAPI, users will need to install both SMAPI and the mod we developed. This process can be easily carried out on Windows, macOS, and Linux systems with a graphical interface. Additionally, we have ensured compatibility for Linux systems without a graphical interface. After purchasing the game, users can install Stardew Valley through Steamcmd, the command-line tool of the Steam client to install and update dedicated servers for Steam games. We also provide installation commands for SMAPI, allowing users to install it directly. To use our developed mod, users simply need to copy it to the designated mod folder. Since Stardew Valley is a graphical application, it cannot be directly opened on systems without a graphical interface. To address this limitation, we use the X virtual framebuffer (Xvfb), which supports all graphical operations in virtual memory without displaying any screen output. This directly ensures that our environment is compatible with various system and hardware configurations.

The interaction between our algorithm and Stardew Valley is achieved through the mod we developed, rather than simulating keyboard and mouse inputs. This approach allows us to open multiple game instances and control each one independently. Specifically, when launching multiple games, we assign a unique port number to each one. Actions provided by the algorithms being trained or tested are transmitted through this port to the corresponding game, and the returned information is received via the same port. Moreover, we use shared memory for each port to improve communication efficiency when necessary. By ensuring that the actions executed and information transmitted in all game processes do not interfere with one another, we enable parallel training and testing. To help researchers efficiently establish the initial state for each task, we provide the simulator APIs to configure the game environment.

C.1 Observation Space

StarDojo provides a rich and flexible observation space to accommodate the diverse needs of various agents. The observation space includes both visual and textual observations, which can be customized and extended by developers to suit their specific goals.

Visual Observation. StarDojo leverages the game engine to directly retrieve rendered gameplay images, eliminating the need for inefficient screen-capture methods. This approach ensures high-fidelity visual observations that are consistent with the gameplay environment. The resolution of these images can be configured dynamically, ranging from 360P to 4K. Notably, the resolution scaling is not merely a resizing operation; as the image size increases, the field of view also expands, providing agents with a broader perspective of the game world. This feature is particularly useful for tasks requiring detailed spatial awareness or long-term planning.

Textual Observation. While visual observations are essential, recent work [33] highlights the challenges faced by state-of-the-art MLLMs in accurately interpreting the unique art style and precise manipulation requirements of Stardew Valley. To address this limitation, StarDojo provides structured textual observations through built-in APIs. These observations are designed to complement visual data, offering agents a more comprehensive understanding of the game state. The textual observations are organized into four main categories:

- **Character Information**: Includes health, energy, gold, position, inventory, location, facing direction, professions, skills (e.g., Farming, Mining, Combat, Fishing, Foraging), dating partners and spouse.
- **Surrounding Information**: Includes tile information within an N×N grid centered on the player's position. Each tile in the grid contains detailed information about its contents and properties, such as debris, crops, NPCs, exits, buildings, furniture, terrain features, and other properties. The grid size can be adjusted to balance granularity and computational efficiency.
- **Map-level Global Information**: Provides global information within the current location map. Details such as crops, exits, NPCs, buildings, shop counters, furniture in rooms, and animals or pets on the farm are included as part of the map-level global information. By providing a structured overview of the entire map, this information enables the model to locate and reason about specific targets more efficiently, while reducing the need for exhaustive exploration of the environment.
- **Game-level Global Information**: Provides global information across the entire game. Game states such as time, day of month, season, year and weather are included in this part, along with other necessary information like the current menu information if any menu is showing.

As shown in our experiment, we selected a subset of the available information as the general configuration. Specifically, we adopted character information, surrounding information, and game-level global information to construct the observation space for Stardojo. Although map-level global information can significantly boost agent performance in certain tasks, our goal is to evaluate the agent's exploration ability based on visual input. Since map-level global information offers shortcuts for locating specific targets, it is deliberately excluded from our experimental setup. This combination of visual and textual observations ensures that agents have access to both high-level contextual information and fine-grained environmental details, enabling more robust and informed decision-making.

The observation space captures a comprehensive snapshot of the game's state in a JSON-like structure, with an additional screenshot RGB map. All relevant details are organized within the following nested objects. Figure 5 shows an example of the screenshot included in the observation.

Observation Space Format

The observation space consists of the following structured fields:

- **Health**: Integer representing the agent's current health.
- **Energy**: Float indicating the agent's current energy level.
- **Money**: Integer showing the amount of money the agent holds.
- **Current Time**: String formatted as `hh:mm AM/PM`.
- **Day**: Integer indicating the current day in the season.
- **Season**: String, one of `spring`, `summer`, `fall`, or `winter`.

- **Item in Your Hand**: A dictionary with fields:
  - `index`: Integer slot index.
  - `currentitem`: String name of the item.
- **Toolbar**: A list of 36 item slot descriptions in the format: `"slot_index N: [Item Name] (quantity: Q)"` or `"slot_index N: No item"`
- **Current Menu**: A dictionary with keys such as: `type`, `message`, `shopmenudata`, `animalsmenudata`, etc.
- **Surrounding Blocks**: A list of nearby tiles, each with:
  - `position`: A 2D integer coordinate offset relative to the agent.
  - `object`: A list of string attributes (e.g., `Type: Dirt`, `Diggable: True`).
  - (Optional) `npc on this tile`: Information about an NPC, if present.

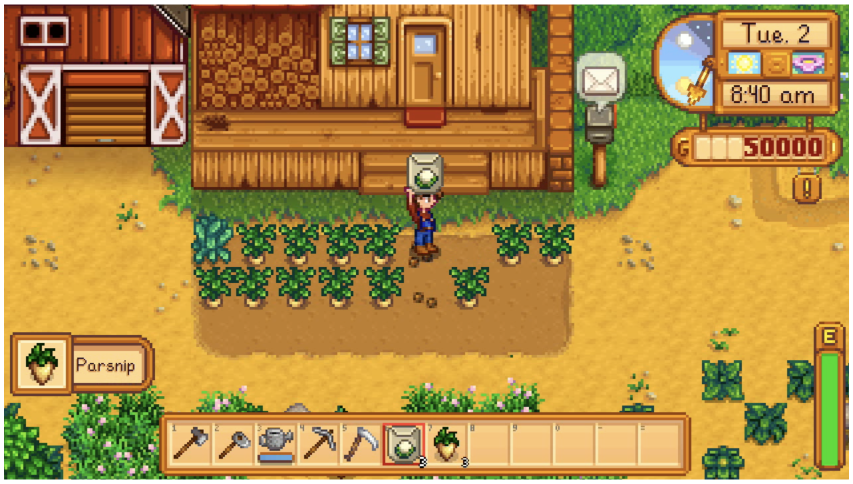

Figure 5: Example of game screenshot as part of the observation space.

C.2 Action Space

We designed an abstracted action space, and further derived a variant specifically for RL agents.

To better align with the actual action space of human players while simplifying redundant operations that do not contribute to the model's decision-making capabilities, we designed a simplified minimal action space.

For simple options which consists of merely a single click or a key press, such as use and interact, we faithfully reproduced the same actions as human players. However, other options require sophisticated drap-and-drop actions or complex menu interactions. For these complex options, we provided a set of finely wrapped abstract actions, which better align with MLLMs' decision making abilities and prevented the models from failing into complecated manipulation traps.

The action space defines the set of skills (or actions) that an agent can perform. Each action is implemented as a function with a specific call template and a thorough comment. The full list of actions is provided in **Table 4**, which details each available action along with its parameters and intended behavior.

Table 4: Complete Action Space with Call Templates and Parameter Descriptions

Action	Description
`move(x, y)`	**Call Template:** `move(x = ..., y = ...)` **Parameters:** `x`, `y` – X and Y coordinates of the destination. Move to the position (x, y).
`craft(item)`	**Call Template:** `craft(item = ...)` **Parameters:** `item` – The name of the item to craft. Craft an item based on its name.
`use(direction)`	**Call Template:** `use(direction = ...)` **Parameters:** `direction` – A string: `up`, `right`, `down`, or `left`. Use an item in a specified direction. Requires proper positioning.
`choose_item(slot_index)`	**Call Template:** `choose_item(slot_index = ...)` **Parameters:** `slot_index` – Inventory index (0–35). Choose the item in the specified inventory slot.
`interact(direction)`	**Call Template:** `interact(direction = ...)` **Parameters:** `direction` – A string: `up`, `right`, `down`, or `left`. Interact with an object or NPC in a specific direction.
`choose_option(option_index, quantity, direction)`	**Call Template:** `choose_option(option_index = ..., quantity = ..., direction = ...)` **Parameters:** `option_index` – Index of the option to choose. `quantity` (optional) – Quantity of items to buy/sell. `direction` (optional) – `"in"` for buy/take, `"out"` for sell/put. Choose from a list of options, with optional quantity and direction.
`attach_item(slot_index)`	**Call Template:** `attach_item(slot_index = ...)` **Parameters:** `slot_index` – Inventory index (0–35). Attach the item in the given inventory slot.
`unattach_item()`	**Call Template:** `unattach_item()` **Parameters:** None. Unattach the currently attached item.
`menu(option, menu_name)`	**Call Template:** `menu(option = ..., menu_name = ...)` **Parameters:** `option` – `"open"` or `"close"` `menu_name` – Menu name (e.g., `"map"`) Open or close a specific menu.
`navigate(name)`	**Call Template:** `navigate(name = ...)` **Parameters:** `name` – Name of the location to navigate to. Navigate to a specified location.

C.3 Task

Our benchmark provides 1000 tasks organized into five categories:

- **Farming**: Farming tasks can be broadly categorized into two types: cultivation (growing crops) and husbandry (raising animals). Easy tasks involve routine agricultural work such as clearing and tilling tiles, fertilizing, sowing seeds, watering plants, harvesting mature crops and animal products, as well as animal care—including feeding, grazing, and interaction. These simple, discrete operations test whether agents possess the most fundamental production capabilities. Medium tasks require agents to independently procure farming resources such as seeds, fertilizer, water, and hay. These resources can be obtained through foraging, crafting, or purchasing. Hard tasks typically span multiple days, requiring agents to perform daily routine farming activities. These activities form a cohesive production chain, following a specific sequence where each step is interdependent. For example, if a task involves growing a plant from seed to harvest, the agent must plant the seed in tilled dirt, water it daily over several days, and finally reap the crop. Any missed step—such as failing to water the plant on a given day—could delay or even prevent maturation. Thus, hard farming tasks demand that agents autonomously allocate time and resources efficiently, presenting significant tests of multi-step reasoning and long-term planning capabilities.

- **Crafting**: Crafting tasks encompass both fabrication and cooking. Most crafting activities simply require adequate raw materials, though some may need equipment like furnace or cookout kit. Typically, easy tasks provide all necessary materials and tools directly in the agent's inventory, requiring only proper execution of crafting procedures. Occasionally, agents might need to gather readily available resources like a few woods or stones. Medium tasks demand greater autonomy - agents must identify, locate and acquire appropriate materials and equipment through careful planning. Hard tasks involve procuring diverse materials through demanding methods (like deep mining for ores), followed by complex, multi-stage processes like crafting intermediary components, testing an agent's comprehensive crafting capabilities.
- **Exploration**: Exploration tasks can be categorized into three types: map navigation/pathfinding, resource gathering in wilderness areas, and completing challenging in-game quests. Easy tasks may involve traveling to an accessible location or collecting specified items within a small nearby area. Medium tasks present greater complexity - some require venturing into more distant and hazardous environments (like the second floor of the mines), while others demand searching expansive zones (such as an entire forest) for randomly spawning resources. Hard tasks challenge agents to locate extremely rare resources with highly randomized spawn locations (like amethysts in mines). Additionally, built-in game quests that chain multiple sub-tasks of varying types - challenging enough to occasionally defeat even experienced human players - also qualify as high-difficulty exploration objectives.
- **Combat**: Agents are tasked with eliminating a specified number of monsters in mines. As difficulty escalates, targets become both more formidable and numerous. Basic adversaries like slimes, bugs, and grubs present minimal threat - their predictable movement patterns, low health pools, and weak attacks make them easy to dispatch or evade. However, advanced creatures employ deadly specialties: flies attack with erratic, lightning-fast strikes; duggies ambush from subterranean positions; rock crabs retreat into impregnable armored stances. Defeating these requires dynamic positioning, tactical strike timing, and adaptive combat strategies.
- **Social**: Social tasks encompass two primary objectives: cultivating relationships with NPCs, and conducting transactional interactions with specialized NPCs (such as carpenter, blacksmith, etc.). Easy tasks require only basic interactions like conversations or gift-giving, and agents are teleported directly to designated transaction locations (e.g., at the counter of Pierre's General Store). Medium tasks demand agents autonomously select and navigate to appropriate venues. Hard tasks challenge agents to build high friendship levels with NPCs. This requires strategic planning to increase rapport through daily greetings, thoughtful gift-giving, and fulfilling requests. Each NPC possesses unique behavioral patterns and preferences. Agents must develop customized engagement strategies, as actions that delight one NPC (e.g., a favored gift) may offend another (e.g., a disliked item). This nuanced system tests agents' adaptive social intelligence.

Each category of tasks is encoded in a YAML file. This dictionary-like file format is both highly readable and convenient for processing by Python programs. The structure of a task is as follows:

Task Format

sow_5_dirt_with_cauliflower_seeds: The key serves as both the name and the description of the task, and will be used as the prompt input to LLM.

- **id**: A unique identifier for the task within its category.
- **object**: The target object of the task—such as item, location, character, or quest—that the player tries to acquire, reach, interact with, or complete in a specific quantity.
- **quantity**: The required number of the target object. For non-quantifiable objects, such as location and NPC, quantity is simply set to 1.
- **tool**: The tool required to complete the task.
- **save**: The initial game save for the task, which has pre-configured some common environmental settings to reduce frequent calls to simulator APIs.
- **init_commands**: This is a list of commands to invoke the simulator APIs. After loading the task, StarDojo will automatically execute these commands one by one, invoking the corresponding API to fine-tune the initial conditions. Combined with the save file, StarDojo achieves efficient standardization for each task.
- **evaluator**: The type of evaluator to assess the task.
- **difficulty**: The difficulty level of the task.

The 100 tasks in StarDojo-Lite are listed in Table 5.

Table 5: Complete list of tasks in StarDojo-Lite.

Task	Category	ID	Object	Quantity	Tool	Save	Init Commands	Evaluator	Difficulty
clear_10_weeds_with_scythe	Farming	0	Weeds	10	Scythe	save_new		clear	easy
clear_5_stone_with_pickaxe	Farming	1	Stone	5	Pickaxe	save_new		clear	easy
clear_30_debris_with_scythe_and_pickaxe_and_axe	Farming	2	Debris	30	Scythe, Pickaxe, Axe	save_new		clear	medium
till_5_tile_with_hoe	Farming	3	Tile	5	Hoe	save_new		till	easy
fertilize_5_dirt_with_basic_retaining_soil	Farming	4	Dirt	5	Basic Retaining Soil	save_farming	add_item_by_name("Basic Retaining Soil", 5)	fertilize	easy
fertilize_1_dirt_with_speed_gro	Farming	5	Dirt	1	Speed-Gro	save_farming		fertilize	easy
sow_5_dirt_with_cauliflower_seeds	Farming	6	Dirt	5	Cauliflower Seeds	save_farming	add_item_by_name("Cauliflower Seeds", 5)	sow	easy
sow_1_dirt_with_potato_seeds	Farming	7	Dirt	1	Potato Seeds	save_new	set_time(time=900)	sow	medium
water_5_crop_with_watering_can	Farming	8	Crop	5	Watering Can	save_farming		water	easy
harvest_5_parsnip	Farming	9	Parsnip	5		save_farming		harvest	easy
cultivate_and_harvest_1_garlic	Farming	10	Garlic	1		save_new	add_item_by_name("Garlic Seeds")	harvest	hard
pet_3_animal	Farming	11	Animal	3		save_farming		pet	easy
pet_8_animal	Farming	12	Animal	8		save_farming		pet	medium
open_1_deluxe_coop	Farming	13	Deluxe Coop	1		save_farming		open	easy
fill_1_pet_bowl_with_watering_can	Farming	14	Pet Bowl	1	Watering Can	save_new		fill	easy
fill_1_feeding_bench_with_hay	Farming	15	Feeding Bench	1	Hay	save_farming	add_item_by_name("Hay", 12)	fill	easy
harvest_1_egg	Farming	16	Egg	1		save_farming		harvest	easy
harvest_1_milk_with_milk_pail	Farming	17	Milk	1	Milk Pail	save_farming	add_item_by_name("Milk Pail")	harvest	easy
harvest_3_milk_with_milk_pail	Farming	18	Milk	3	Milk Pail	save_farming		harvest	hard
incubate_1_chicken_with_incubator	Farming	19	Chicken	1	Incubator	save_farming		incubate	hard
earn_50_friendship_with_1_cat	Farming	20	Cat	50		save_new		friendship	hard
craft_1_cherry_bomb	Crafting	0	Cherry Bomb	1		save_new	set_time(time=900)	craft	medium
craft_1_wood_fence	Crafting	1	Wood Fence	1		save_new		craft	easy
craft_1_sprinkler	Crafting	2	Sprinkler	1		save_new	add_item_by_name("Furnace"), add_item_by_name("Copper Ore", 5), add_item_by_name("Iron Ore", 5), add_item_by_name("Coal", 2)	craft	medium
craft_1_basic_retaining_soil	Crafting	3	Basic Retaining Soil	1		save_new		craft	easy
craft_1_spring_seeds	Crafting	4	Spring Seeds	1		save_new		craft	hard
craft_1_field_snack	Crafting	5	Field Snack	1		save_new	add_item_by_name("Acorn"), add_item_by_name("Maple Seed"), add_item_by_name("Pine Cone")	craft	easy
craft_1_torch	Crafting	6	Torch	1		save_new		craft	easy
craft_1_furnace	Crafting	7	Furnace	1		save_new	set_time(time=900)	craft	medium
craft_1_chest	Crafting	8	Chest	1		save_new	set_time(time=900)	craft	medium
craft_1_scarecrow	Crafting	9	Scarecrow	1		save_new	add_item_by_name("Wood", 50), add_item_by_name("Coal"), add_item_by_name("Fiber", 20)	craft	easy
produce_1_copper_bar_with_furnace	Crafting	10	Copper Bar	1	Furnace	save_new	add_item_by_name("Furnace"), add_item_by_name("Copper Ore", 5), add_item_by_name("Coal")	craft	easy
produce_1_refined_quartz_with_furnace	Crafting	11	Refined Quartz	1	Furnace	save_new		craft	hard
cook_1_fried_egg_with_stove	Crafting	12	Fried Egg	1	Stove	save_new	upgrade_house(1), add_item_by_name("Egg")	craft	easy
cook_1_salad_with_cookout_kit	Crafting	13	Salad	1	Cookout Kit	save_new		craft	hard
go_to_bed	Exploration	0	Bed	1		save_new		sleep	easy
go_to_coop	Exploration	1	Coop	1		save_new		location	easy
go_to_bus_stop	Exploration	2	BusStop	1		save_new		location	easy
go_to_backwoods	Exploration	3	Backwoods	1		save_new		location	easy
go_to_pierre's_general_store	Exploration	4	SeedShop	1		save_new	set_time(time=900)	location	easy
go_to_marnie's_ranch	Exploration	5	AnimalShop	1		save_new	set_time(time=900)	location	easy
go_to_fish_shop	Exploration	6	FishShop	1		save_new	set_time(time=900)	location	easy
go_to_carpenter's_shop	Exploration	7	ScienceHouse	1		save_new	set_time(time=900)	location	easy
go_to_the_mines_2nd_floor	Exploration	8	UndergroundMine2	1		save_new		location	medium
go_to_the_mines_5th_floor_by_elevator	Exploration	9	UndergroundMine5	1	Elevator	save_new		location	medium
go_to_the_mines_10th_floor	Exploration	10	UndergroundMine10	1		save_new		location	hard
chop_10_wood_with_axe	Exploration	11	Wood	10	Axe	save_new		harvest	easy
chop_20_wood_with_axe	Exploration	12	Wood	20	Axe	save_new		harvest	medium
forage_1_wild_horseradish	Exploration	13	Wild Horseradish	1		save_new	warp("forest")	harvest	medium
forage_1_daffodil	Exploration	14	Daffodil	1		save_new	warp("town")	harvest	medium
forage_1_leek	Exploration	15	Leek	1		save_new	warp("mountain")	harvest	medium
forage_1_clam	Exploration	16	Clam	1		save_new	warp("beach")	harvest	easy
forage_10_hay_with_scythe	Exploration	17	Hay	10	Scythe	save_new		silo	easy
forage_1_quartz	Exploration	18	Quartz	1		save_new	warp_mine(1)	harvest	medium
dig_1_cave_carrot_with_hoe	Exploration	19	Cave Carrot	1	Hoe	save_new	warp_mine(13)	harvest	easy
mine_1_amethyst_with_pickaxe	Exploration	20	Amethyst	1	Pickaxe	save_new	warp_mine(1)	harvest	hard
mine_1_copper_ore_with_pickaxe	Exploration	21	Copper Ore	1	Pickaxe	save_new	warp_mine(2)	harvest	easy
mine_1_coal_with_pickaxe	Exploration	22	Coal	1	Pickaxe	save_new	warp_mine(1)	harvest	medium
quit_1_quest	Exploration	23	Quest	1		save_new		quit	easy
take_1_quest_reward	Exploration	24	Quest Reward	1		save_quests		reward	easy
complete_1_help_wanted_quest	Exploration	25	Help Wanted Quest	1		save_new		complete_help	hard
complete_the_story_quest_"introductions"	Exploration	26	9	1		save_new		complete_story	hard
complete_the_story_quest_"getting_started"	Exploration	27	6	1		save_quests		complete_story	hard
kill_1_green_slime_with_rusty_sword	Combat	0	Green Slime	1	Rusty Sword	save_new	warp_mine(2)	kill	easy
kill_5_green_slime_with_rusty_sword	Combat	1	Green Slime	5	Rusty Sword	save_new	warp_mine(2)	kill	medium
kill_10_green_slime_with_rusty_sword	Combat	2	Green Slime	10	Rusty Sword	save_new	warp_mine(2)	kill	hard
kill_1_bug_with_rusty_sword	Combat	3	Bug	1	Rusty Sword	save_new	warp_mine(2)	kill	easy
kill_5_bug_with_rusty_sword	Combat	4	Bug	5	Rusty Sword	save_new	warp_mine(2)	kill	medium
kill_10_bug_with_rusty_sword	Combat	5	Bug	10	Rusty Sword	save_new	warp_mine(2)	kill	hard
kill_1_fly_with_rusty_sword	Combat	6	Fly	1	Rusty Sword	save_new	warp_mine(2)	kill	medium
kill_1_duggy_with_rusty_sword	Combat	7	Duggy	1	Rusty Sword	save_new	warp_mine(6)	kill	medium
kill_1_grub_with_rusty_sword	Combat	8	Grub	1	Rusty Sword	save_new	warp_mine(15)	kill	easy
kill_5_grub_with_rusty_sword	Combat	9	Grub	5	Rusty Sword	save_new	warp_mine(15)	kill	medium
kill_10_grub_with_rusty_sword	Combat	10	Grub	10	Rusty Sword	save_new	warp_mine(15)	kill	hard
kill_1_rock_crab_with_rusty_sword	Combat	11	Rock Crab	1	Rusty Sword	save_new	warp_mine(2)	kill	medium
ship_1_parsnip_with_shipping_bin	Social	0	Parsnip	1	Shipping Bin	save_new	add_item_by_name("Parsnip")	sell	easy
purchase_5_beer	Social	1	Beer	5		save_new	warp_shop("gus")	purchase	easy
purchase_1_muscle_remedy	Social	2	Muscle Remedy	1		save_new	set_time(time=900)	purchase	medium
sell_5_parsnip_to_pierre	Social	3	Parsnip	5		save_new	warp_shop("pierre"), add_item_by_name("Parsnip", 5)	sell	easy
sell_1_parsnip_to_pierre	Social	4	Parsnip	1		save_new	add_item_by_name("Parsnip"), set_time(time=900)	sell	medium
upgrade_to_copper_pickaxe	Social	5	Copper Pickaxe	1		save_new	add_item_by_name("Copper Bar", 5)	upgrade_tool	hard
break_5_geode	Social	6	Geode	5		save_new	warp_shop("clint"), add_item_by_name("Geode", 5)	break	easy
purchase_joja_membership	Social	7	Joja Membership	1		save_new	warp("joja", 21, 26)	jojamart	easy
purchase_minecarts_development_project	Social	8	Minecarts Repaired	1		save_new	joja_membership(), set_time(time=900)	jojamart	medium
upgrade_to_large_pack	Social	9	Large Pack	1		save_new	warp_shop("pierre")	backpack	easy
purchase_1_chicken	Social	10	Chicken	1		save_new	warp_shop("marnie")	purchase_animal	easy
sell_1_chicken	Social	11	Chicken	1		save_new		sell_animal	easy
build_1_big_coop	Social	12	Big Coop	1		save_new	warp_shop("robin"), add_item_by_name("Wood", 400), add_item_by_name("Stone", 150)	build	easy
move_1_coop	Social	13	Coop	1		save_new	set_time(time=900)	move	medium
upgrade_farmhouse	Social	14	Farmhouse	1		save_new	warp_shop("robin"), add_item_by_name("Wood", 450)	upgrade_farmhouse	easy
demolish_1_shipping_bin	Social	15	Shipping Bin	1		save_new	set_time(time=900)	demolish	medium
talk_to_alex	Social	16	Alex	1		save_new	set_time(time=800)	talk	easy
talk_to_sebastian	Social	17	Sebastian	1		save_new	set_time(time=1500)	talk	easy
talk_to_vincent	Social	18	Vincent	1		save_new	set_time(time=900)	talk	easy
give_abigail_1_amethyst	Social	19	Abigail	1	Amethyst	save_new	add_item_by_name("Amethyst", 1, 4), set_time(time=900)	gift	easy
give_haley_1_coconut	Social	20	Haley	1	Coconut	save_new	add_item_by_name("Coconut", 1, 4), set_time(time=1100)	gift	easy
give_jas_1_fairy_rose	Social	21	Jas	1	Fairy Rose	save_new	add_item_by_name("Fairy Rose", 1, 4), set_time(time=900)	gift	easy
give_jodi_1_pancakes	Social	22	Jodi	1	Pancakes	save_new	add_item_by_name("Pancakes", 1, 4), set_time(time=900)	gift	easy
earn_100_friendship_with_elliott	Social	23	Elliott	100		save_new	add_item_by_name("Pomegranate", 1, 4), set_time(time=1100)	friendship	medium
earn_200_friendship_with_harvey	Social	24	Harvey	200		save_new		friendship	hard

C.4 Simulator APIs

To establish the initial state for each task, we develop a set of simulator APIs that allow fine-grained customization of the game environment. Many tasks require strict initial conditions and world settings, such as resources (e.g., sufficient crop seeds in inventory), weather conditions (e.g., a rainy day), and game progress (e.g., unlocking the mines). The simulator APIs can configure these task-specific conditions, endowing each task with a unique environment setup. This mechanism significantly expands the benchmark's diversity and flexibility, allowing varied and nuanced task designs that closely mirror the dynamic production-living settings in human society.

To efficiently standardize our tasks, we also create a series of tailored game save files, which have pre-configured some common environmental settings using simulator APIs. These files reduce the excessively frequent calls to simulator APIs in real time, improving the efficiency of task execution. At the beginning of each task, StarDojo automatically loads the corresponding saved game and invokes specific simulator APIs, ensuring all necessary prerequisites are appropriately configured.

The complete APIs are as follows:

Simulator APIs

- **Player Settings**
  - **set_base_health(amount: int)**: Set the health capacity of the player.
  - **set_health(amount: int)**: Set the current health of the player.
  - **set_base_energy(amount: int)**: Set the energy capacity of the player.
  - **set_energy(amount: int)**: Set the current energy of the player.
  - **set_inventory_size(size: int)**: Set the inventory size.
  - **clear_inventory()**: Clear the inventory.
  - **set_money(amount: int)**: Set the amount of money that the player possesses.
  - **add_item_by_id(id: str, count: int, quality: int)**: Add the specific item of given count and quality (e.g., 0, 1) to inventory by ID.
  - **add_item_by_name(name: str, count: int, quality: int)**: Add the specific item of given count and quality to inventory by name.
  - **lookup(name: str)**: Look up and print the item ID by name.
  - **current_position()**: Print the current position of the player.
  - **add_recipe(type: str, recipe: str)**: Teach the player the specific crafting / cooking recipe.
  - **set_max_luck()**: Set player's luck to the maximum.
  - **print_luck()**: Print player's luck.
- **Surrounding Settings**
  - **world_clear(entity: str, location: str)**: Remove all entities of the given type (e.g., "crops", "trees") from a location.
  - **set_terrain(terrain: str, id: str, x: int, y: int)**: Set the terrain feature of the given tile.
  - **place_item(item: str, type: str, x: int, y: int)**: Place the specific item on the given tile.
  - **remove_item(x: int, y: int)**: Remove the item on the given tile.
  - **place_crop(crop: str, x: int, y: int)**: Place the specific crop on the given tile.
  - **grow_crop(day: int, x: int, y: int)**: Grow the crop on the given tile for a specific number of days.
  - **grow_tree(day: int, x: int, y: int)**: Grow the tree on the given tile for a specific number of days.
  - **build(type: str, force: bool, x: int, y: int)**: Build the specific building at the given coordinate.
  - **build_stable(x: int, y: int)**: Build a stable at the given coordinate.
  - **move_building(x_source: int, y_source: int, x_dest: int, y_dest: int)**: Move the building from the source coordinate to the destination coordinate.
  - **remove_building(x: int, y: int)**: Remove the building at the given coordinate.
  - **upgrade_house(level: int)**: Upgrade the farmhouse to the given level.
- **Character Settings**
  - **spawn_pet(type: str, breed: str, name: str, x: int, y: int)**: Spawn a pet of given type (e.g., "cat", "dog"), breed (e.g., "0", "1"), and name on a tile.
  - **spawn_animal(type: str, name: str)**: Spawn an animal of given type and name in the animal house.

- **grow_animal(name: str)**: Set the specific animal in current location to day 1 of adulthood, unless already adult.
- **animal_friendship(name: str, friendship: int)**: Set the friendship with the specific animal.
- **npc_friendship(npc: str, friendship: int)**: Set the friendship with the specific NPC.
- **all_npc_friendship(friendship: int)**: Set the friendship with all NPCs.
- **dating(npc: str)**: Make the specific NPC be the player's boyfriend / girlfriend.

- **Location Settings**
  - **warp(location: str, x: int, y: int)**: Warp the player to given location and position.
  - **warp_mine(level: int)**: Warp the player to given mine level.
  - **warp_volcano(level: int)**: Warp the player to given volcano level.
  - **warp_home()**: Warp the player back home.
  - **warp_shop(npc: str)**: Warp the player to the shop run by the given NPC.
  - **warp_character(npc: str, location: str, x: int, y: int)**: Warp the specific NPC to given location and position.
- **World Settings**
  - **set_date(year: int, season: str, day: int)**: Set the date.
  - **set_time(time: int)**: Set the current time.
  - **rain()**: Set the weather to rainy.
- **Progression Settings**
  - **set_deepest_mine_level(level: int)**: Set the deepest mine level reached by the player.
  - **set_monster_stats(monster: str, kills: int**: Set the kill stats for the specific monster to the given value.
  - **print_monster_stats(monster: str)**: Print the kill stats for the specific monster.
  - **start_quest(id: str)**: Start the quest of given ID.
  - **start_help_quest(type: str)**: Start a random help wanted quest of given type.
  - **complete_quest(id: str)**: Complete the quest of given ID.
  - **joja_membership()**: Give the player JojaMart membership.
  - **spawn_junimo_note(id: str)**: Spawn the junimo note of given ID in the Community Center.
  - **mark_bundle(id: str)**: Mark the completion of the specific bundle.
  - **complete_room_bundles(id: str)**: Complete all bundles in the specific room of the Community Center.
  - **community_development(id: str)**: Complete the community development project of given ID.
  - **receive_mail(mail: str)**: Add the specific mail to mailbox.
  - **trigger_event(id: str)**: Trigger the event of given ID.
  - **seen_event(id: str, see_or_forget: bool)**: Mark the viewed / unviewed status of the specific event.
  - **load_save(save: str)**: Load the game save of given name.

C.5 Evaluation

Given the scale of our extensive task set, it is imperative to design an efficient and reusable evaluation mechanism that not only accurately monitors task progression but also delivers immediate reward feedback to agents. Therefore, we implement a unified evaluation system based on textual observation comparison. The evaluation workflow follows a consistent pattern across all tasks, as outlined in the following steps and Algorithm 1:

- **Maintain the previous observation**: Store the agent's prior textual observation to enable temporal comparison.
- **Capture the current observation**: Acquire the latest textual observation.
- **Compare two observations**: Based on the task's evaluator type (e.g., harvest, sell) and target object (e.g., item, NPC), the system detects related game state changes, such as items in the inventory and surrounding tiles, to quantify task progress.
- **Accumulate incremental progress:** Accumulate incremental changes captured per step into a sum.
- **Check completion criterion**: Validate whether predefined success conditions (e.g., quantity thresholds, event triggers) are met.

Algorithm 1 Task Evaluation

```
1:  function EVALUATE($obs$)
2:      if $last\_obs$ is null then
3:          $last\_obs \leftarrow obs$
4:          return $\{$completed : False, current_quantity : $0\}$
5:      end if
6:      $quantity\_change \leftarrow$ COMPARE($evaluator, object, obs, last\_obs$)
7:      $last\_obs \leftarrow obs$
8:      $current\_quantity \leftarrow current\_quantity + quantity\_change$
9:      if $current\_quantity \geq quantity$ then
10:         $completed \leftarrow$ True
11:     else
12:         $completed \leftarrow$ False
13:     end if
14:     return $\{$completed : $completed$, current_quantity : $current\_quantity\}$
15: end function
```

The evaluation system relies on the comprehensive observation space, allowing for step-by-step tracking of progress toward the task goal. This incremental inspection approach avoids differences caused by varying initial conditions, making the proposed evaluation mechanism more generalizable. The evaluation output provides standardized metrics, including completion status and current progress. These metrics also serve as reward feedback and, along with observations, are passed to the agents to prompt them to adjust their behavior.

D Experiment Settings

If not mentioned explicitly, all experiments are conducted under the following settings: Agents have access to both visual and textual observations for their decision-making process. Visual observations are provided at a resolution of 720p (1280×720 pixels). Textual observations include detailed information about current state. Specifically, there are information about the player itself, including health, energy, gold, chosen item, and inventory information. There are also global information, which are current time, current day, season, and current open menu. Finally, we provide 7×7 agent-centered surrounding information, containing details about each tile, ranging from the terrain, debris, buildings, object, exits, NPCs, and furniture information, to other tile properties encoded in the game. In addition to receiving observations from the current timestep, agents are also provided with action and visual information from the previous timestep. Incorporating previous timestep information enables agents to reflect on past states and facilitate consistency in decision-making. All the agents can output at most two skills as an action to be executed sequentially. After executing all the actions, the environment remains paused until the agent outputs the next action. All experiments are repeated three times to ensure reliability.

Prompt Used in Main Experiment

You are a helpful AI assistant integrated with 'Stardew Valley' on the PC, equipped to handle various tasks in the game. Your advanced capabilities enable you to process and interpret gameplay screenshots and other relevant information. By analyzing these inputs, you gain a comprehensive understanding of the current context and situation within the game. Utilizing this insight, you are tasked with identifying the most suitable in-game action to take next, given the current task. You control the game character and can execute actions from the available action set. Upon evaluating the provided information, your role is to articulate the precise action you would deploy, considering the game's present circumstances, and specify any necessary parameters for implementing that action.
Here is some helpful information to help you make the decision.
Your Current task is: <$task_description$>

Basic knowledge about the environment:

1. Hoe is used to till the soil, Watering Can is used to water the soil, Pickaxe is used to break rocks, Axe is used to chop trees, Scythe is used to harvest crops.
2. When you want to go through a door, move in front of it by 1 tile, and interact towards it.
3. Please go to bed at night (after 18:00) even if your task is not yet complete!

4. Call interact(direction) with a box, a shipping bin or anything else. Call use(direction) to use an item or tool in your inventory.

Health: <$health$>

Energy: <$energy$>

Money: <$money$>

Current Time: <$time$>

Day: <day>

Season: <$season$>

Item in your hand: <$chosen_item$>

Toolbar of items which you can choose from: <$toolbar_information$>

Current menu: <$current_menu$>

Surrounding blocks (Objects surrounds you): <$surroundings$>

Valid action set in Python format to select the next action: <$skill_library$>

Last executed action: <pre_action>

<$image_introduction$>

Based on the above information, analyze the current situation and provide the reasoning for what you should do for the next step to complete the task. Then, you should output the exact action you want to execute in the game. You should respond to me with:

Reasoning: You should think step by step and provide detailed reasoning to determine the next action executed on the current state of the task. You need to answer the following questions step by step. You cannot miss the last question:

1. Is there an open menu? What is the current menu saying? What are the options? Which one should you choose or should you exit the menu? Use choose_option to make a choice.
2. What is the current map? Do you need to move to another map?
3. Refer to surroundings, what are the important tiles? What are their positions?
4. You are always at (0, 0). You can only affect points adjacent to you, such as (0,1), (0,-1), (1,0), (-1,0). Is your target at these positions? If not, move next to it first.
5. Analyze the information in the toolbar. Does it contain all the necessary items for completing the task? What is the current item?
6. When calling use or interact, you need to decide the direction. For example, if you are at (x,y):
   - call interact("up") or use("up") to interact with or use against (0,-1)
   - call interact("right") or use("right") to interact with or use against (1,0)
   - call interact("down") or use("down") to interact with or use against (0,1)
   - call interact("left") or use("left") to interact with or use against (-1,0)
7. What is the current image showing? What additional information can be learned from the image?
8. Do all the selected actions exist in the valid action set? If no, regenerate the actions and give the reasons.

Actions: The requirements that the generated action needs to follow. The best action, or short sequence of actions without gaps, to execute next to progress in achieving the goal. Pay attention to the names of the available skills and to the previous skills already executed, if any. You should also pay more attention to the following action rules:

1. You should output actions in Python code format and specify any necessary parameters to execute that action. If the function has parameters, you should also include their names and

decide their values, like move(x=0, y=1). If it does not have a parameter, just output the action, like unattch_item().

2. You can only output at most 2 actions in the output.
3. If you want to interact with the objects in the toolbar, you need to make sure that the target object is already selected. You need to use choose_item() to select them before executing use().
4. If you want to plant a seed or harvest a mature crop, please use interact(). If you want to use tools, like axe, hoe, watering can, pickaxe and scythe, please use use().
5. Your action should strictly follow the analysis in the reasoning. Do not output any additional action not mentioned in the reasoning.

You should only respond in the format described below, and you should not output comments or other information.
Reasoning:
1. ...
2. ...
3. ...
Actions:

```
‘‘‘
python
    action(args1=x,args2=y)
‘‘‘
```

Conclusion

The above cases collectively reveal key limitations of current large language models when deployed in grounded, spatially structured environments like Stardew Valley. Despite having access to both visual and textual information, the model consistently exhibits hallucinations in spatial understanding — including confusion about direction, misjudgment of proximity, and failure to plan indirect paths around obstacles.

These behaviors suggest that LLMs, while capable in language-based reasoning, still lack robust internal representations of space and geometry. Moreover, the persistence of such errors even with explicit instructions points to fundamental weaknesses in grounding language to actionable physical reasoning. Bridging this gap will require future work in multimodal integration, spatial memory, and instruction-following mechanisms tailored to embodied agents. Our findings underscore the importance of evaluating LLMs not just on language tasks, but within interactive environments where spatial and physical reasoning are essential.

E Financial Cost

The financial costs of evaluating different closed-source MLLMs on StarDojo-Lite for one round are summarized in Table 6.

Table 6: Financial costs of running one round on StarDojo-Lite with different closed-source MLLMs.

MLLM	Model Name	Cost
GPT-4.1	gpt-4.1-2025-04-14	$20.6
Gemini 2.5 Pro	gemini-2.5-pro-preview-03-25	$25.8
Claude 3.7 Sonnet	claude-3-7-sonnet-20250219	$38.7
GPT-4.1 mini	gpt-4.1-mini-2025-04-14	$4.3