VistaWise: Building Cost-Effective Agent with Cross-Modal Knowledge Graph for Minecraft

Honghao Fu[1,2‡*], **Junlong Ren**[1‡], **Qi Chai**[1], **Deheng Ye**[3], **Yujun Cai**[2], **Hao Wang**[1†]
[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]The University of Queensland, [3]Tencent
honghao.fu@uq.edu.au, {jren686, qchai315}@connect.hkust-gz.edu.cn
dericye@tencent.com, yujun.cai@uq.edu.au, haowang@hkust-gz.edu.cn

Abstract

Large language models (LLMs) have shown significant promise in embodied decision-making tasks within virtual open-world environments. Nonetheless, their performance is hindered by the absence of domain-specific knowledge. Methods that finetune on large-scale domain-specific data entail prohibitive development costs. This paper introduces VistaWise, a cost-effective agent framework that integrates cross-modal domain knowledge and finetunes a dedicated object detection model for visual analysis. It reduces the requirement for domain-specific training data from millions of samples to a few hundred. VistaWise integrates visual information and textual dependencies into a cross-modal knowledge graph (KG), enabling a comprehensive and accurate understanding of multimodal environments. We also equip the agent with a retrieval-based pooling strategy to extract task-related information from the KG, and a desktop-level skill library to support direct operation of the Minecraft desktop client via mouse and keyboard inputs. Experimental results demonstrate that VistaWise achieves state-of-the-art performance across various open-world tasks, highlighting its effectiveness in reducing development costs while enhancing agent performance.

1 Introduction

The development of agents in open-world environments is widely regarded as a promising avenue for advancing artificial general intelligence (Liu et al., 2024; Reed et al., 2022). These agents have to navigate intricate environments and make decisions under conditions of uncertainty. Among the various platforms for developing such agents, Minecraft has emerged as a prominent virtual environment, owing to its open-ended design and the extensive range of task possibilities (Wang et al., 2024b). Early efforts (Baker et al., 2022) within Minecraft focused on training visual models using reinforcement learning in simulators (Guss et al., 2019; Fan et al., 2022). However, these approaches face challenges related to systematic exploration (Wang et al., 2020), interpretability (Zhao et al., 2021), and generalization (Wang et al., 2024b).

With the advancement of Large Language Models (LLMs), applying LLM agents on Minecraft has become a prominent trend (Zhu et al., 2023; Wang et al., 2023b). However, due to limitations in environmental perception and grounding, LLM-based agents in Minecraft often rely on environmental APIs (e.g., MineFlayer[1]) to obtain accurate textual descriptions of the environment and execute actions (Wang et al., 2024b; Zhao et al., 2025). This reliance may hinder generalization, as not all virtual environments provide such APIs. Additionally, directly utilizing APIs for high-level actions (e.g., chopping logs) restricts the autonomy and potential of LLM-based agents (Cai et al., 2024b). In contrast, an ideal agent should rely solely on visual cues for reasoning and perform tasks using low-level and hybrid action spaces, which involve mouse and keyboard (MNK) operations, more closely mimicking human behavior.

Building on this objective, several studies (Wang et al., 2024c; Cai et al., 2024a; Wang et al., 2024d) propose the use of Multimodal LLMs (MLLMs) for reasoning based on visual information, subsequently enhancing visual policies with the reasoning outcomes. In parallel, to mitigate performance degradation or hallucinations resulting from the LLM's lack of domain-specific knowledge in virtual environments, finetuning the LLM with knowledge tailored to the target virtual world has become a widely adopted strategy (Wang et al., 2023b, 2024d,c; Zhao et al., 2025; Qin et al., 2024). However, several challenges exist: (1) The re-

*The work was done during an internship at HKUST(GZ).
†Corresponding Author. ‡Equal Contribution.

[1]https://github.com/PrismarineJS/mineflayer

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21896–21910
November 4-9, 2025

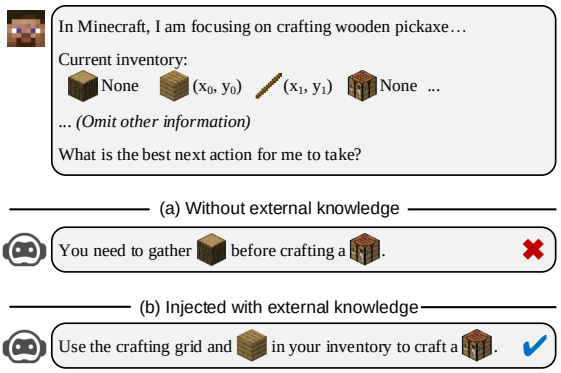

Figure 1: **The motivation of injecting external knowledge.** (a) LLMs may generate incorrect dependencies due to a lack of domain-specific knowledge in the virtual world; (b) Injecting external knowledge enables LLMs to generate more accurate responses.

liance on additional visual policies to predict actions diminishes the LLM's decision-making potential. Moreover, this approach usually models visual input globally, incorporating irrelevant noise that results in unstable predictions for MNK operations (Cai et al., 2024a). (2) General visual policies or LLMs may underperform due to the lack of domain-specific knowledge. However, finetuning them with large-scale domain-specific data needs prohibitive development costs.

To address these problems, we propose VistaWise, a cost-effective agent with a cross-modal knowledge graph for Minecraft. VistaWise incorporates an object detection model to identify visual entities, effectively minimizing irrelevant information and noise. This model is the sole component in VistaWise that requires finetuning on domain-specific data. It is efficiently finetuned with fewer than 500 annotated frames extracted from gameplay videos, thereby significantly reducing the training cost for embedding domain knowledge into the visual modality.

VistaWise utilizes an LLM as the policy to directly predict actions, harnessing the LLM's reasoning and decision-making capabilities. To mitigate hallucinations (e.g., as shown in Figure 1), we aim to integrate domain-specific knowledge into the textual modality through external knowledge retrieval (Lewis et al., 2020), thus eliminating the need for additional finetuning of the LLMs. Inspired by the efficacy of knowledge graphs (KGs) in reducing hallucinations (Edge et al., 2024), we construct a textual KG using online textual information as the external knowledge base to provide factual dependencies. Given the multimodal nature of tasks in Minecraft, the visual information extracted by the object detection model is further embedded into the graph, enabling the construction of the cross-modal knowledge graph that enhances the agent's comprehensive and accurate understanding of the environment. To mitigate the inference cost of LLMs due to information redundancy, we further design a retrieval-based graph-pooling strategy for efficiently accessing the cross-modal information stored within the graph structure.

To enhance the agent's dynamic decision-making capabilities, we integrate a concise task description, Chain-of-Thought (CoT) reasoning (Wei et al., 2022), and a memory module. To facilitate action execution, we develop a skill library that references human players' MNK behaviors using PyAutoGUI, from which the LLM autonomously generates parameters based on visual cues. This approach empowers the agent to directly control the game on the desktop, eliminating the need for simulation environments or environmental APIs. Our key contributions are summarized as follows:

- We propose a cost-effective agent framework that incorporates multimodal domain-specific knowledge by finetuning an object detection model and externally retrieving textual knowledge, reducing training data requirements from millions to only a few hundred samples.
- We integrate visual and textual information via a KG to construct cross-modal representations, enhancing the agent's understanding of multimodal tasks. We also design a pooling strategy termed retrieval-based pooling to extract information from the introduced KG.
- By directly leveraging the LLM's reasoning and decision-making capabilities, VistaWise outperforms other non-API-based baselines in complex tasks, achieving a success rate of 33% in obtaining diamonds, surpassing the previous state-of-the-art rate of 25%.

2 Related Work

2.1 Agents in Minecraft

Agents based on vision models. MineRL (Guss et al., 2019) and MineDOJO (Fan et al., 2022) offer simulation platforms for developing virtual agents in Minecraft. Early agents utilize visual models trained through reinforcement or imitation learning as policies. VPT (Baker et al., 2022) employs internet-scale pretraining for sequential decision-making via semi-supervised imitation learning.

STEVE-1 (Lifshitz et al., 2023) extends VPT by incorporating additional prior knowledge derived from task descriptions and task execution videos. However, these methods face challenges related to systematic exploration, interpretability, and generalization (Wang et al., 2024b).

Agents based on LLMs. Compared to traditional visual models, LLMs excel in complex cognitive tasks and causal reasoning, driving the development of Minecraft agents based on LLM policies (Zhu et al., 2023; Chai et al., 2025). To address limitations of LLMs in visual capabilities, Voyager (Wang et al., 2024b) proposes to leverage environmental APIs to obtain precise data and execute high-level actions. With the emergence of Multimodal LLMs (MLLMs), subsequent works such as STEVE (Zhao et al., 2025), LARM (Li et al., 2024), MP5 (Qin et al., 2024), LLaMA-Rider (Feng et al., 2024), and Odyssey (Liu et al., 2024) further enhance agent performance by incorporating visual inputs and finetuning MLLMs with domain-specific data. However, these methods rely on certain APIs to gather information and execute actions, facing challenges in generalization, as not all environments provide such APIs.

Agents integrating LLMs and vision models. To enable LLM-based agents to execute actions solely based on visual input, without reliance on APIs, JARVIS-1 (Wang et al., 2024c) introduces the use of MLLMs for long-term planning. ROCKET-1 (Cai et al., 2024a) further incorporates priors from MLLMs and SAM (Kirillov et al., 2023) to train a visual policy. However, these approaches incur substantial costs in data collection and training. In this paper, we offer a more efficient framework by minimizing the complexity of these processes.

2.2 LLMs with Knowledge Graph

LLMs have demonstrated remarkable language understanding and zero-shot transfer abilities across various natural language processing tasks. However, they still lack up-to-date or domain-specific knowledge (Wang et al., 2023a), which hinders their generalization. Recent works have proposed to adopt the knowledge graph (KG) to provide up-to-date and structured domain-specific knowledge to improve LLM reasoning on knowledge-intensive tasks (Pan et al., 2024). Many methods (Zhang et al., 2022; Jiang et al., 2023b; Baek et al., 2023; Jiang et al., 2023a) combine LLMs with KGs through converting retrieved relevant knowledge from KGs to textual prompts for LLMs. Nevertheless, the effectiveness of these methods mainly depends on the quality of the retrieved knowledge. To retrieve more accurate and reliable knowledge from KGs, recent works (Jiang et al., 2023c; Sun et al., 2024; Jiang et al., 2024; Wang et al., 2025b) utilize LLMs to enhance reasoning on KGs.

3 Proposed Method

3.1 Overview

Figure 2 shows the framework of VistaWise, which leverages visual inputs and desktop-level control to interact with Minecraft. VistaWise is built around an LLM and consists of three graph-based processes: (1) text-modal graph construction, (2) cross-modal graph construction, (3) task-specific information retrieval, and two interaction modules: (i) a desktop-level skill library and (ii) a memory stack.

3.2 Cross-modal Information Integration

Text-modal graph construction. As shown in Figure 1(a), the agent may misinterpret entity dependencies and suffer from hallucinations due to a lack of domain-specific knowledge, resulting in incorrect reasoning during crafting tasks. While finetuning LLMs to incorporate new knowledge is a viable approach, it suffers from significant costs related to data collection and additional training. An alternative and widely recognized solution is to provide factual support via an external knowledge base (Lewis et al., 2020; Jiang et al., 2023d), where a structured KG is an effective way to provide factual dependencies (Wang et al., 2025a; Edge et al., 2024; Hu et al., 2024; Han et al., 2024). We structure online textual knowledge into a KG $\mathcal{G}_{\text{init}} = (\mathcal{V}_{\text{init}}, \mathcal{E}_{\text{init}})$, where entity nodes $\mathcal{V}_{\text{init}}$ (e.g., "Player," "Tools," "Iron Ingot") and their relationships $\mathcal{E}$ (e.g., "includes," "can be used to mine," "is used to craft") capture factual dependencies. In practice, we find that entity names alone are sufficient for the LLM to understand these dependencies. Therefore, we remove other information (e.g., background knowledge) of the nodes, creating a more lightweight representation. It reduces the external knowledge injected, thus decreasing the computational costs while maintaining comparable performance as indicated in Table 2 and Sec. 4.4.

Visual perception. To predict the next action, the agent must not only comprehend the factual dependencies within the virtual world but also perceive entities in the observable space. In order to accurately perceive the environment and mit-

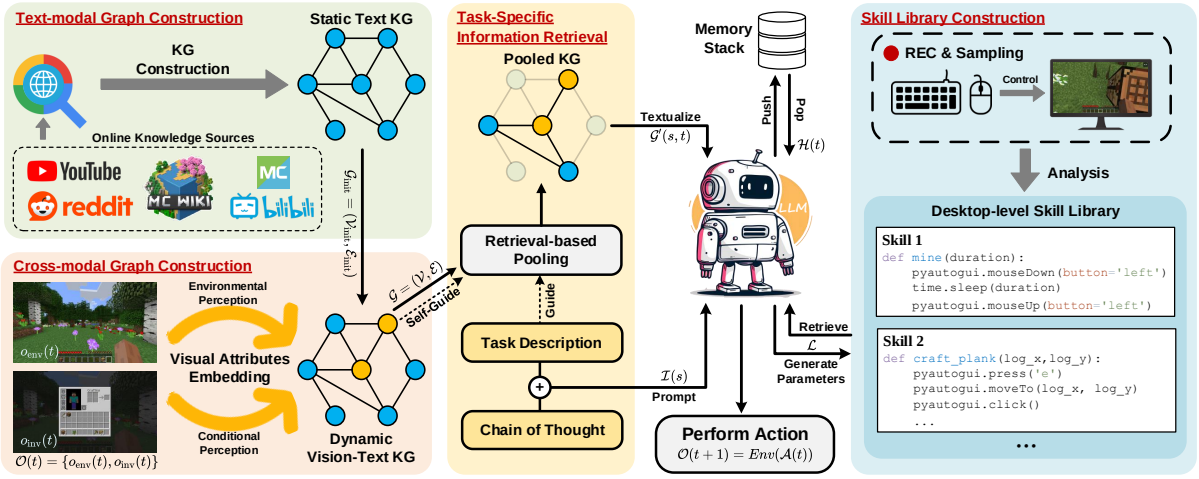

Figure 2: **The overview of VistaWise.** VistaWise is based on an LLM and incorporates three graph-based processes: (1) **text-modal graph construction**, integrating external textual domain knowledge via a lightweight KG to establish factual dependencies and mitigate hallucinations; (2) **cross-modal graph construction**, embedding real-time visual information from a dedicated object detection model into the text-modal graph, forming a vision-text graph with dynamic visual attributes; (3) **task-specific information retrieval**, utilizing a retrieval-based pooling strategy to extract task-related information from the vision-text graph, guided by both the task-specific prompt and the real-time visual attributes of the graph. Furthermore, VistaWise comprises two interaction modules: (i) a **desktop-level skill library**, allowing the agent to act in the Minecraft desktop client via MNK operations, with action parameters generated autonomously by the LLM; (ii) a **memory stack**, storing and querying decision history to support reasoning. At each timestep, the agent makes decisions and executes actions based on the prompt, retrieved information, memory, and skill library, altering the game environment to advance the task.

igate token overhead from visual inputs, we utilize a dedicated object detection model to replace the object grounding and detection functions of MLLMs, extracting real-time entity information (e.g., coordinates, bounding boxes) from the observable space. We classify the entities into two categories: environmental entities, which represent the spatial distribution of the environment, and conditional entities, which can be used to track the progress of the agent's task. We define the observable space for the agent at timestep t as $\mathcal{O}(t) = \{o_{\text{env}}(t), o_{\text{inv}}(t)\}$, where o_{env} and o_{inv} represent the environment and inventory spaces in Minecraft. The goal is to enable the agent to accurately perceive the spatial locations of interactive or semantically significant visual entities within $\mathcal{O}$. The agent's perception is based on two parallel processes: one for environmental entities and another for conditional entities, both supported by an object detection model D. To model the virtual environment, we define the environmental entities in $o_{\text{env}}(t)$ as $E(t) = \{e_{\text{t,1}}, e_{\text{t,2}}, \ldots, e_{\text{t,n}}\}$ (e.g., trunks, lava, ores). The agent interacts with these entities to gather resources and modify the environment to achieve the goal task. For any observable entity e at any timestep, the agent obtains its desktop-view information $D(o_{\text{env}}, e) = \{x_e, y_e, w_e, h_e\}$, where (x_e, y_e) represents the center coordinates of e, and w_e and h_e denote the width and height of its bounding box. Let V_{env} represent all the information related to E in o_{env} accessible through D. The V_{env} at timestep t is expressed as:

$$V_{\text{env}}(t) = \{D(o_{\text{env}}(t), e_{\text{t,i}}) \mid e_{\text{t,i}} \in E(t), i = 1, 2, \ldots, n\}. \quad (1)$$

Moreover, we define the visual entities in $o_{\text{inv}}(t)$ (e.g. item icons) as conditional entities $C(t) = \{c_{\text{t,1}}, c_{\text{t,2}}, \ldots, c_{\text{t,m}}\}$. For any c at any timestep, the information the agent can observe is $D(o_{\text{inv}}, c) = \{x_c, y_c\}$, where (x_c, y_c) are the center coordinates of c. Agent tracks task progress and assesses whether conditions for specific actions (e.g., crafting) are met based on C. Let V_{inv} denote all information related to C in o_{inv} accessible through D. The V_{inv} at timestep t is expressed as:

$$V_{\text{inv}}(t) = \{D(o_{\text{inv}}(t), c_{\text{t,i}}) \mid c_{\text{t,i}} \in C(t), i = 1, 2, \ldots, m\}. \quad (2)$$

In addition, a significant challenge is assessing the distance between the agent and e based on visual information. It is primarily raised by the distribution gap between real-world and virtual environment visuals, which renders deep estimation models trained on real-world data ineffective in capturing the spatial distribution of o_{env}. Drawing inspiration from how human players empirically

estimate distances based on the size of visual entities, we use w_e and h_e to determine whether the agent is within the interaction range of e, employing empirical thresholds k_w and k_h, respectively. This enables the agent to perceive spatial distance in a coarse-grained manner.

Cross-modal graph construction. The real-time visual information is embedded as attributes of the corresponding entity nodes in the static KG that describes factual relationships. This visual attributes embedding process updates the text KG with static dependencies into a cross-modal vision-text KG with dynamic visual attributes. At timestep t, we embed the results of the visual perception process as attributes to the corresponding entity nodes in $\mathcal{G}_{\text{init}}$, which include the spatial and conditional information ($V_{\text{env}}(t)$ and $V_{\text{inv}}(t)$) of the observed environment. This approach cross-modally integrates the static text-modal KG with dynamic spatial and conditional visual context, enhancing the agent's understanding of the environment by linking real-time visual perception to the semantic entities represented in the KG. The resulted cross-modal vision-text graph is denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

3.3 Graph-based Information Retrieval

We retrieve task-related cross-modal information from the cross-modal KG to support the reasoning and decision-making processes of the agent and reduce the computational resources occupied by redundant information. However, the lightweight design of the KG results in insufficient descriptive text, weakening traditional semantic similarity-based retrieval. Therefore, we propose a retrieval-based graph-pooling strategy for accessing the information stored within the introduced KG.

Prompt synthesis. For a task s, we define a task description $T_{\text{td}}(s)$ to guide the agent's focus on task-relevant information, such as "In Minecraft, I am focusing on chopping logs and need to make reasonable action choices based on the position of logs on the screen and my crosshair position." To support reasoning, we include CoT $T_{\text{cot}}(s)$ (Wei et al., 2022), like "Is the crosshair close enough to the target?", "Does the crosshair need further alignment?", or "Do you need to move closer to the target?". The synthetic prompt for task s can be defined as $\mathcal{I}(s) = \{T_{\text{td}}(s), T_{\text{cot}}(s)\}$, which guides the information retrieval process and prompts the agent's next action.

Retrieval-based pooling. Due to the limited textual attributes in the constructed lightweight graph

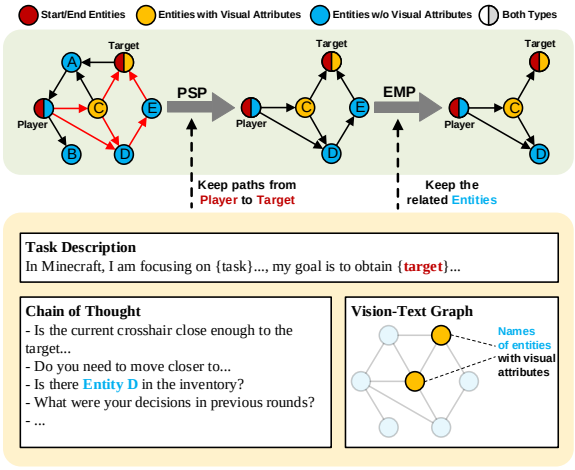

Figure 3: **Retrieval-based pooling.** It first employs path searching pooling (PSP) to retain paths from the "Player" node to the task-specific "Target" node in the KG. Subsequently, entity matching pooling (EMP) preserves entities referenced in the task prompt and those with visual attributes in the dynamic vision-text graph. The pooled graph is textualized and input to the LLM, providing the agent with factual dependencies and real-time visual information.

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$, traditional information retrieval methods relying on semantic similarity (Ren et al., 2025) face robustness issues (Edge et al., 2024). To address this, we propose a retrieval-based pooling mechanism for the graph, as shown in Figure 3, which includes path searching pooling (PSP) and entity matching pooling (EMP). Let p represent a path between two nodes in $\mathcal{V}$. The PSP retains $p \in \mathcal{P}_{\text{P-T}}(s)$, where $\mathcal{P}_{\text{P-T}}(s)$ denotes the set of paths from the player node v_{player} to the task target node $v_{\text{target}}(s)$, capturing global dependencies for task s. Each path $p \in \mathcal{P}_{\text{P-T}}(s)$ consists of a node set $\mathcal{V}(p)$ and an edge set $\mathcal{E}(p)$. The nodes $\mathcal{V}_{\text{global}}(s)$ and edges $\mathcal{E}_{\text{global}}(s)$ along all related paths for task s can be expressed as:

$$\mathcal{V}_{\text{global}}(s) = \bigcup_{p \in \mathcal{P}_{\text{P-T}}(s)} \mathcal{V}(p), \tag{3}$$

$$\mathcal{E}_{\text{global}}(s) = \bigcup_{p \in \mathcal{P}_{\text{P-T}}(s)} \mathcal{E}(p). \tag{4}$$

Following PSP, the EMP checks whether each node in $\mathcal{V}_{\text{global}}(s)$ appears in the synthetic prompt $\mathcal{I}(s)$ and visual attributes at timestep t, and retains the corresponding nodes $\mathcal{V}_{\text{local}}(s,t) \in \mathcal{V}_{\text{global}}(s)$ and their edges $\mathcal{E}_{\text{local}}(s,t) \in \mathcal{E}_{\text{global}}(s)$ to retrieve dynamic local dependencies that adapt to the real-time task progress. After pooling, the KG $\mathcal{G}'(s,t) = (\mathcal{V}_{\text{local}}(s,t), \mathcal{E}_{\text{local}}(s,t))$ contains information that most relevant to the agent's state at timestep t, which is then textualized as input to the LLM.

3.4 Desktop-level Skill Library

Previous works that employ LLMs as policies typically utilize APIs (e.g., MineFlayer) to construct skill libraries for agents, with each agent interacting with the environment in high-level action spaces to accomplish complex tasks (Wang et al., 2024b). However, it limits the agent's ability to generalize, as not all environments provide APIs (Cai et al., 2024b). A potential improvement is enabling interaction through MNK operations, yet existing works that adopt this method rely on action libraries tailored to specific simulators, restricting their transferability (Wang et al., 2024c; Cai et al., 2024b,a).

To address this limitation, we develop a desktop-level skill library $\mathcal{L}$ based on the concept of general computer control (Tan et al., 2024), utilizing PyAutoGUI in hybrid action spaces (Wang et al., 2024c). This library encompasses low-level MNK operations (e.g., pressing a key or dragging the mouse) and their combinations (e.g., crafting items). By enabling the agent to act directly within the Minecraft desktop client through MNK inputs, the library removes the dependency on simulators or environment-specific APIs. Examples of skill functions within the library are shown on the right side of Figure 2. The agent can access the library, retrieve the necessary skill functions, and autonomously generate input parameters based on the observable visual information.

3.5 Memory Stack

Given the recency sensitivity of decision-making (i.e., recent decisions matter more) and the causality of continuous decisions in virtual games, we design a memory module based on the Last-In-First-Out (LIFO) stack storage concept in physical computer systems, termed the "memory stack". The LIFO property prioritizes the recent memory and ensures the agent can query the decision history continuously. At each timestep, the agent pushes its latest decision to the top of the memory stack. At timestep t, a query $q(t)$ with specified recall steps prompts the stack to pop decision histories $\mathcal{H}(t) = M(q(t))$ from top to bottom, allowing the agent to recall decisions from the most recent to earlier ones with controllable recall steps.

3.6 Agent Workflow Modeling

Let π_θ denote a parameterized LLM. At timestep t for task s, the agent uses π_θ as policy to predict an executable action $\mathcal{A}(t)$ (a skill function and its parameters) based on the synthetic prompt $\mathcal{P}(s,t)$, the pooled knowledge graph $\mathcal{G}'(s,t)$, decision histories $\mathcal{H}(t)$, and the accessible skill library $\mathcal{L}$. This workflow can be modeled as:

$$\mathcal{A}(t) = \pi_\theta(\mathcal{I}(s), \mathcal{G}'(s,t), \mathcal{H}(t), \mathcal{L}). \quad (5)$$

After $\mathcal{A}_t$ is executed in the Minecraft environment Env, the observable space at the next timestep can be expressed as:

$$\mathcal{O}(t+1) = Env(\mathcal{A}(t)). \quad (6)$$

4 Experiment

4.1 Dataset and Implementation

Dataset. We collect a small-scale dataset[2] to train an object detection model for Minecraft, defining 23 types of visual entities. From Minecraft gameplay videos, we extract 471 frames and annotate 3,304 instances using X-AnyLabeling. This dataset is used to finetune a pretrained object detection model for application in Minecraft environments. Additional details are provided in Appendix B.

Implementation details. We use the pretrained YOLOv10-L (Wang et al., 2024a) object detection model for entity perception, finetuned with Minecraft data on an L4 GPU with 24G VRAM. The dataset is split into training and testing sets in a 9:1 ratio, with finetuning conducted over 150 epochs, a batch size of 16, and an input image size of 768. Other settings follow YOLO's default configurations in the Ultralytics package. The empirical thresholds in Sec.3.2 are $k_w = 110$ and $k_h = 275$. We use GPT-4o (Achiam et al., 2023) as the LLM policy. The agent acts in the Vanilla 1.11.2 Minecraft client on a Windows 10 machine, equipped with a 1080p resolution monitor, a 3060 Ti GPU with 8G VRAM, an i7-12700F CPU, and 32GB of RAM. Unless otherwise specified, experimental results are based on 15 repeated trials. The setup of the game environment and details of the skill library are provided in Appendix C and D.

4.2 Comparison against other methods

We compare our method with 7 non-API-based baselines[3]. These methods includes: STEVE-1 (Lifshitz et al., 2023), GROOT (Cai et al., 2024b), DEPS (Wang et al., 2023b), Omni JARVIS (Wang

[2]The dataset is available at https://drive.google.com/file/d/1QXGtSJJWw4emKB8RLGYUvq_fNEDCxhP_/view?usp=sharing

[3]We exclude comparisons with works using environmental APIs (e.g., MineFlayer) for perception and action.

Table 1: **Comparison with other non-API-based baselines.** The comparison includes the scale of the training dataset and the GPU VRAM threshold required for integrating domain knowledge, and the success rate of milestone goals for "obtain diamond". '*' indicates that base models are finetuned on domain-specific text, but the dataset and GPU resources scale were not disclosed. '-' denotes not available publicly.

Method	Venue	Dataset Scale	GPU VRAM									
STEVE-1	NeurIPS'23	160M frames	192 GB	0.04	0.04	-	0.00	0.00	0.00	0.00	0.00	0.00
GROOT	ICLR'24	1.6B frames	384 GB	0.05	0.05	-	0.00	0.00	0.00	0.00	0.00	0.00
DEPS	NeurIPS'23	*	*	0.90	0.80	-	0.80	0.70	-	-	0.10	0.01
Omni JARVIS	NeurIPS'24	990M tokens	640 GB	0.95	0.95	0.82	-	-	0.82	0.32	0.32	0.08
JARVIS-1	TPAMI'24	*	*	0.97	0.96	0.95	0.94	0.94	0.57	0.55	0.34	0.09
VPT	NeurIPS'22	140B frames	23040 GB	0.99	0.99	0.99	0.98	0.78	**0.84**	0.60	0.57	0.15
ROCKET-1	CVPR'25	1.6B frames	*	**1.00**	**1.00**	**1.00**	-	**1.00**	-	-	-	0.25
VistaWise (Ours)	**EMNLP'25**	**471 frames**	**24 GB**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.73	**0.73**	**0.73**	**0.33**

Table 2: **Ablation study on textual entity attributes** in the KG on the success rate of milestone goals.

Attributes						
Full information	1.00	1.00	1.00	0.80	0.80	0.27
Names only	1.00	1.00	1.00	0.73	0.73	0.33

et al., 2024d), JARVIS-1 (Wang et al., 2024c), VPT (Baker et al., 2022), and ROCKET-1 (Cai et al., 2024a). Results for STEVE-1 and GROOT are taken from Omni JARVIS, while other results are sourced from the respective original papers.

Performance on "obtain diamond". We test our method with the ultimate goal of "obtaining diamond," a classic challenge for agents in Minecraft. Table 1 compares our performance with others across 9 sequential sub-goals leading to this objective: obtain crafting table → obtain wooden pickaxe → ... → obtain diamond. These sub-goals must be executed sequentially, with failure at any stage resulting in the failure of subsequent goals. The results demonstrate that our method achieves state-of-the-art (SOTA) performance on the ultimate goal and most of the sub-goals.

Efficiency. As shown in Table 1, we finetune the object detection model using only 471 frames extracted from game videos and 24 GB GPU VRAM, whereas other methods typically rely on at least 160M frames with 192 GB VRAM or 990M tokens of text with 64 GB VRAM. Despite the substantially smaller dataset scale and VRAM usage, we achieve SOTA performance. It demonstrates the effectiveness of our approach in reducing development costs related to data collection and training.

4.3 Ablation Study

Entity attributes. In Sec.3.3, we construct a KG to integrate external textual knowledge for the agent, reducing hallucinations in complex crafting tasks. To minimize token overhead, we retain only entity names and exclude other information (e.g., background knowledge). As shown in Table 2, the agent successfully understands entity dependencies despite the omission of full information, with no crafting failures. Variations in success rates between the two ablation groups stem from the random distribution of in-game resources.

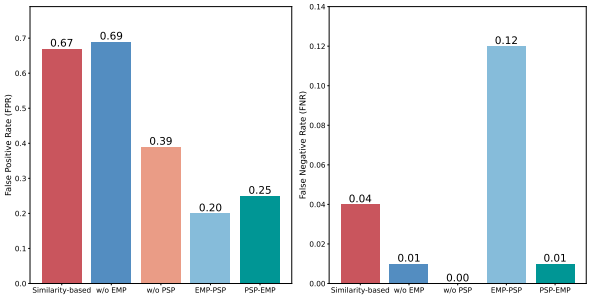

Figure 4: **Ablation study on information retrieval strategies. (Left)** False Positive Rate (FPR), the proportion of redundant information in the retrieved results. **(Right)** False Negative Rate (FNR), the proportion of missed information to all that should have been retrieved. Lower FPR and FNR indicate better retrieval. "Similarity-based", "EMP", and "PSP" refer to the similarity-based strategy, entity matching pooling, and path searching pooling, respectively, while "EMP-PSP" and "PSP-EMP" denote their execution order.

Knowledge retrieval strategies. Figure 4 compares similarity-based and pooling-based retrieval strategies in our lightweight KG using False Positive Rate (FPR) and False Negative Rate (FNR). The similarity-based strategy employs a BERT model (all-MiniLM-L6-v2) to embed prompts and graph attributes, and the cosine similarity between them to retrieve relevant entities. We also examine the roles of path pooling, entity pooling, and their order. The results indicate that the pooling strategy outperforms similarity-based retrieval in the introduced KG. Moreover, performing entity pooling first disrupts the node-edge structure, potentially losing critical paths. Thus, considering both FPR and FNR, the optimal strategy is path

Table 3: **Ablation study on LLM policies and visual perception methods.** 'V' denotes leveraging the visual capabilities of the MLLM for image understanding, while 'OD' refers to using an object detection model. "a/b" indicates that the agent successfully achieved the goal a times out of b consecutive attempts.

LLM Policy	Method					
Gemini-1.5-pro	V	2/3	0/3	0/3	0/3	0/3
	OD	**3/3**	**3/3**	**3/3**	**2/3**	**1/3**
	V+OD	**3/3**	**3/3**	**3/3**	**2/3**	**1/3**
Qwen-VL-Max	V	0/3	0/3	0/3	0/3	0/3
	OD	**3/3**	0/3	0/3	0/3	0/3
	V+OD	**3/3**	0/3	0/3	0/3	0/3
GPT-4o	V	**3/3**	0/3	0/3	0/3	0/3
	OD	**3/3**	**3/3**	**3/3**	**2/3**	**1/3**
	V+OD	**3/3**	**3/3**	**3/3**	**2/3**	**1/3**

Table 4: **Ablation study on memory stack, CoT and KG.** "a/b" indicates that the agent successfully achieved the goal a times out of b consecutive attempts.

Method					
w/o Memory Stack	**3/3**	**3/3**	**3/3**	0/3	0/3
w/o CoT	2/3	0/3	0/3	0/3	0/3
w/o KG	**3/3**	2/3	2/3	1/3	0/3
Full	**3/3**	**3/3**	**3/3**	**2/3**	**1/3**

pooling followed by entity pooling.

LLM policies and visual perception. Table 3 evaluates several common MLLMs (Team et al., 2023; Bai et al., 2023; Achiam et al., 2023) within our framework, assessing agent performance with and without object detection models or MLLM visual capabilities. Results show that without accurate visual entity information from object detection, relying solely on MLLM's visual capability is inadequate for complex tasks. Besides, as object detection provides sufficient information, additional visual input does not significantly affect performance. It indicates that using an object detection model can effectively replace MLLM's visual capability and enhance the agent's spatial perception.

Memory stack, CoT and KG. We examine the effectiveness of several components in Table 4. The results demonstrate that the memory stack is crucial for handling complex tasks, CoT serves as the foundation for decision-making, and KG contributes to enhancing stability in crafting tasks.

4.4 Inference Costs

In addition to the costs associated with model training, managing inference overhead is crucial to enhancing the cost-efficiency of LLM-based agents. As illustrated in Figure 5, we quantify this overhead

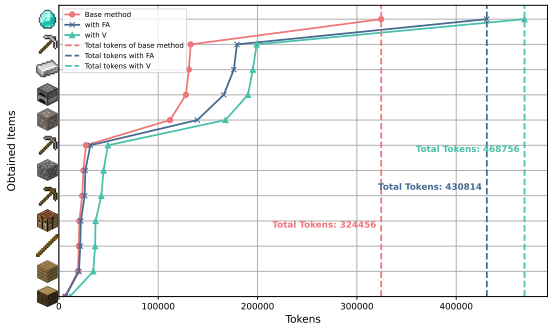

Figure 5: **The tokens consumed by our proposed agent to successfully achieve various goals.** "VistaWise" is our standard framework proposed in Sec.3, while "with FA" and "with V" indicate the addition of full graph attributes (Table 2) and the use of visual input (Table 3) to the standard framework, respectively.

through the token consumption required to achieve various goals. For the ultimate goal of "obtain diamond," omitting full entity attributes during external knowledge injection and disabling the MLLM's visual capabilities leads to a reduction in token consumption by 30.6% and 41.0% respectively, while the agent performance shows no significant degradation as indicated in Tables 2 and 3. Moreover, by leveraging the lightweight graph-based representations of cross-modal information, our framework achieves budget-friendly inference. Based on GPT-4o pricing for 160 iterations, our framework incurs a cost of $\sim$ \$1.28. In comparison, the earlier cost-transparent[4] LLM policy-based agent Voyager, under the same conditions, costs around $\sim$ \$25, achieving a 94.9% reduction in cost. These results validate the cost-effective nature of our method.

5 Conclusion

We present VistaWise, a cost-effective agent framework for open-world embodied decision-making in Minecraft. With cross-modal domain-specific knowledge through retrieving external textual dependencies and training a dedicated object detection model, VistaWise minimizes the need for extensive domain-specific training data. It constructs a cross-modal KG combining visual information and textual dependencies for better task understanding. It also includes a retrieval-based pooling strategy to extract task-relevant data from the KG, and a desktop-level skill library to control the Minecraft desktop client directly via MNK inputs. VistaWise effectively reduces development costs while maintaining high performance, offering a novel and efficient solution for virtual open-world agents.

[4]https://github.com/MineDojo/Voyager/blob/main/FAQ.md

Limitations

Utilizing an LLM as the policy within VistaWise impacts the agent's real-time performance, as each action iteration necessitates waiting for LLM inference and, if applicable, a server response. While pausing the virtual environment's processes synchronizes the agent's decision-making with the environment's timescale, it results in increased time for the agent to complete tasks.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (No. 62406267), Tencent Rhino-Bird Focused Research Program and the Guangzhou Municipal Science and Technology Project (No. 2025A04J4070).

References

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

Jinheon Baek, Alham Aji, and Amir Saffari. 2023. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 1(2):3.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

Shaofei Cai, Zihao Wang, Kewei Lian, Zhancun Mu, Xiaojian Ma, Anji Liu, and Yitao Liang. 2024a. Rocket-1: Master open-world interaction with visual-temporal context prompting. *arXiv preprint arXiv:2410.17856*.

Shaofei Cai, Bowei Zhang, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. 2024b. Groot: Learning to follow instructions by watching gameplay videos. In *The Twelfth International Conference on Learning Representations*.

Qi Chai, Zhang Zheng, Junlong Ren, Deheng Ye, Zichuan Lin, and Hao Wang. 2025. Causalmace: Causality empowered multi-agents in minecraft cooperative tasks. *Preprint*, arXiv:2508.18797.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

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

Yicheng Feng, Yuxuan Wang, Jiazheng Liu, Sipeng Zheng, and Zongqing Lu. 2024. Llama-rider: Spurring large language models to explore the open world. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4705–4724.

William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. 2019. Minerl: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 2442–2448.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A Rossi, Subhabrata Mukherjee, Xianfeng Tang, and 1 others. 2024. Retrieval-augmented generation with graphs (graphrag). *arXiv preprint arXiv:2501.00309*.

Yuntong Hu, Zhihan Lei, Zheng Zhang, Bo Pan, Chen Ling, and Liang Zhao. 2024. Grag: Graph retrieval-augmented generation. *arXiv preprint arXiv:2405.16506*.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yaliang Li, and Ji-Rong Wen. 2023a. Reasoninglm: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3721–3735.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2024. Kg-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph. *arXiv preprint arXiv:2402.11163*.

Jinhao Jiang, Kun Zhou, Xin Zhao, and Ji-Rong Wen. 2023b. UniKGQA: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. In *The Eleventh International Conference on Learning Representations*.

Jinhao Jiang, Kun Zhou, zican Dong, KeMing Ye, Xin Zhao, and Ji-Rong Wen. 2023c. StructGPT: A general framework for large language model to reason over structured data. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023d. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, and 1 others. 2023. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Zhuoling Li, Xiaogang Xu, Zhenhua Xu, SerNam Lim, and Hengshuang Zhao. 2024. Larm: Large auto-regressive model for long-horizon embodied intelligence. *arXiv preprint arXiv:2405.17424*.

Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. 2023. Steve-1: A generative model for text-to-behavior in minecraft. *Advances in Neural Information Processing Systems*, 36:69900–69929.

Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. 2024. Odyssey: Empowering minecraft agents with open-world skills. *arXiv preprint arXiv:2407.15325*.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Yiran Qin, Enshen Zhou, Qichang Liu, Zhenfei Yin, Lu Sheng, Ruimao Zhang, Yu Qiao, and Jing Shao. 2024. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16307–16316. IEEE.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, and 1 others. 2022. A generalist agent. *Transactions on Machine Learning Research*.

Junlong Ren, Hao Wu, Hui Xiong, and Hao Wang. 2025. Sca3d: Enhancing cross-modal 3d retrieval via 3d shape and caption paired data augmentation. *arXiv preprint arXiv:2502.19128*.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*.

Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Gang Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, and 1 others. 2024. Cradle: Empowering foundation agents towards general computer control. In *NeurIPS 2024 Workshop on Open-World Agents*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and 1 others. 2024a. Yolov10: Real-time end-to-end object detection. *Advances in Neural Information Processing Systems*, 37:107984–108011.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024b. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. 2020. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International conference on machine learning*, pages 9940–9951. PMLR.

Shijie Wang, Wenqi Fan, Yue Feng, Xinyu Ma, Shuaiqiang Wang, and Dawei Yin. 2025a. Knowledge graph retrieval-augmented generation for llm-based recommendation. *arXiv preprint arXiv:2501.02226*.

Song Wang, Junhong Lin, Xiaojie Guo, Julian Shun, Jundong Li, and Yada Zhu. 2025b. Reasoning of large language models over knowledge graphs with super-relations. In *The Thirteenth International Conference on Learning Representations*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023a. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023b. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 34153–34189.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, and 1 others. 2024c. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zihao Wang, Shaofei Cai, Zhancun Mu, Haowei Lin, Ceyao Zhang, Xuejie Liu, Qing Li, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. 2024d. Omnijarvis: Unified vision-language-action tokenization enables open-world instruction following agents. *Advances in Neural Information Processing Systems*, 37:73278–73308.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5773–5784.

Zelin Zhao, Karan Samel, Binghong Chen, and 1 others. 2021. Proto: Program-guided transformer for program-guided tasks. *Advances in neural information processing systems*, 34:17021–17036.

Zhonghan Zhao, Wenhao Chai, Xuan Wang, Boyi Li, Shengyu Hao, Shidong Cao, Tian Ye, and Gaoang Wang. 2025. See and think: Embodied agent in virtual environment. In *European Conference on Computer Vision*, pages 187–204. Springer.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, and 1 others. 2023. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

A The Explanation of Icons

We explain the goals represented by the icons mentioned in the main paper in Table 5.

Table 5: The explanation of icons mentioned in the main paper.

Icons (Goals)	Explanations
	Obtain log. Chop a tree using the hands or an axe to collect wood logs.
	Obtain wooden plank. Craft wooden planks from logs.
	Obtain stick. Craft sticks from wooden planks.
	Obtain crafting table. Craft a crafting table from wooden planks.
	Obtain wooden pickaxe. Use wooden planks and sticks to craft a wooden pickaxe at the crafting table.
	Obtain cobblestone. Mine stone blocks with wooden pickaxe to collect cobblestone.
	Obtain stone pickaxe. Craft a stone pickaxe using cobblestones and sticks at the crafting table.
	Obtain iron ore. Mine iron ore blocks (found underground) with stone pickaxe.
	Obtain furnace. Craft a furnace using cobblestones at the crafting table.
	Obtain iron ingot. Smelt the iron ore in a furnace using fuel.
	Obtain iron pickaxe. Craft an iron pickaxe using iron ingots and sticks at the crafting table.
	Obtain diamond, the rarest mineral in our environment. Mine diamond ore blocks (found deep underground) with an iron pickaxe.

B Dataset

We use X-AnyLabeling to annotate a total of 3,304 visual entities across 471 frames extracted from the gameplay video. The entity details and their corresponding annotation counts are provided in Table 6. Entities such as trunk, plank_icon, and log_icon exhibit higher annotation counts, since each of them has different types of textures (such as oak, birch, etc.).

Table 6: The defined visual entity types in the dataset and their annotation counts in the 471 frames of the game screen.

Entities	Annotations
bedrock	59
coal_icon	110
coal_ore	101
cobblestone_icon	249
crafting_table_icon	106
diamond_icon	79
diamond_ore	50
diamond_pickaxe_icon	15
dirt_icon	115
furnace_icon	67
iron_icon	103
iron_ore	141
iron_ore_icon	76
iron_pickaxe_icon	38
lava	15
log_icon	361
non_cobblestone_icon	152
plank_icon	446
stick_icon	181
stone_pickaxe_icon	72
trunk	608
water	37
wood_pickaxe_icon	123
Total	3304

C Game Environment

We use the Vanilla 1.11.2 Minecraft desktop client as the virtual environment for the agent, with the specific game world settings detailed in Table 7.

D Skill Library

We present some important skill functions from the skill library and their explanations in Table 8.

The construction of the skill library is semi-automated. We first manually design a few simple template functions, then define the requirements and pass them to an LLM to generate the corresponding functions. Specifically, the human effort involved includes: (i) Designing a small number of template functions intended to cover all relevant keyboard and mouse operations in the game; (ii) Defining the requirements, including the name of each skill function, its input parameters, and brief descriptions of both the function and its parameters; (iii) Manually refining the generated functions when issues are present. Using GPT-4o mini, we successfully generate all skill functions, with fewer than half requiring minor manual adjustments.

Table 7: Game world settings.

Game Options	Settings
Game Mode	Survival
Generate Structures	ON
Bonus Chest	OFF
Allow Cheats	ON
Sea Level	63
Caves	Yes
Strongholds	Yes
Villages	Yes
Mineshafts	Yes
Temples	Yes
Ocean Monuments	Yes
Woodland Mansions	Yes
Ravines	Yes
Dungeons	Yes
Dungeon Count	7
Water Lakes	Yes
Water Lake Rarity	4
Lava Lakes	No
Lava Lake Rarity	80
Lava Oceans	No
Biome	Plains
Biome Size	4
River Size	4

E Knowledge Graph

Entity and relationship extraction is automated using a multi-stage process. We employ an LLM (GPT-4o mini) with a web plugin to construct the knowledge graph. Selected websites are provided to the LLM, which summarizes the knowledge (entities and their dependencies) from these sites. Due to context length limitations, each site is processed individually, and the outputs are merged into a rough draft document. The LLM is then prompted to refine this document, remove redundancy, and generate a Python-compatible knowledge graph using the NetworkX package.

While this approach produces detailed graphs, it may also introduce task-irrelevant relations, resulting in unnecessary complexity. To mitigate this, we restrict edge types to nine specific categories ("can use", "can mine", "is used to craft", "is used to produce", "can be put in/on", "is the fuel of",

"includes", "can be used to mine", and "outputs") to ensure relevance and clarity. Then, a brief manual review is conducted to remove redundant or irrelevant nodes, further improving the quality of the graph. Human involvement is primarily limited to (i) selecting source websites, (ii) defining allowed relation types, and (iii) performing minor refinements.

F Prompt Template

As an example, we report the prompt template for action prediction in Table 9.

Table 8: Examples from the desktop-level skill library.

Skill Function	Explanation
craft_furnace(c_x, c_y)	Use cobblestone to craft a furnace. 'c_x' and 'c_y' are the x and y coordinates of the cobblestone in the player's inventory.
craft_iron_pickaxe(i_x, i_y, s_x, s_y)	Use iron ingots and sticks to craft an iron pickaxe. 'i_x' and 'i_y' are the x and y coordinates of the iron ingots in the player's inventory, 's_x' and 's_y' are the x and y coordinates of the sticks.
craft_plank(l_x, l_y)	Use logs to craft planks. 'l_x' and 'l_y' are the x and y coordinates of the logs in the player's inventory.
craft_stick(p_x, p_y)	Use planks to craft sticks. 'p_x' and 'p_y' are the x and y coordinates of the planks in the player's inventory.
craft_stone_pickaxe(c_x, c_y, s_x, s_y)	Use cobblestone and sticks to craft a stone pickaxe. 'c_x' and 'c_y' are the x and y coordinates of the cobblestone in the player's inventory, 's_x' and 's_y' are the x and y coordinates of the sticks.
craft_wood_pickaxe(p_x, p_y, s_x, s_y)	Use planks and sticks to craft a wooden pickaxe. 'p_x' and 'p_y' are the x and y coordinates of the planks in the player's inventory, 's_x' and 's_y' are the x and y coordinates of the sticks.
dig_horizontal_mine_tunnels(k)	Select a pickaxe in the hotbar and use it to dig horizontal tunnels to explore ores. 'k' is a number from 1 to 9 corresponding to the key for the pickaxe in the hotbar.
dig_vertical_mine_tunnels(k, d)	Select a pickaxe from the hotbar and use it to dig a vertical mine tunnel to reach (deeper) underground. 'k' is a number from 1 to 9 corresponding to the key for the pickaxe in the hotbar. 'd' is mouse press duration.
mine_log(d)	When log is within 5 meters and the crosshair is correctly aligned with the target, mine the log. 'd' is mouse press duration.
mine_diamond_ore(k, d)	When diamond ore is within 5 meters and the crosshair is correctly aligned with the target, mine the diamond ore. 'k' is the key for the iron pickaxe in the hotbar, and 'd' is mouse press duration.
mine_iron_ore(k, d)	When iron ore is within 5 meters and the crosshair is correctly aligned with the target, mine the iron ore. 'k' is the key for the stone pickaxe in the hotbar, and 'd' is mouse press duration.
move_forward(d)	Move forward for a certain duration. 'd' is the press duration of 'w'.
move_item_to_hotbar(t_x, t_y)	Move a specific item from the inventory to the hotbar. 't_x' and 't_y' are the x and y coordinates of the target item in the player's inventory.
place_blocks_underfoot(k, n)	Place cobblestone blocks underfoot to raise the player from lower to higher position. 'k' is a number from 1 to 9 corresponding to the key for the cobblestone in the hotbar. 'n' is the number of placed blocks.
put_functional_block(k, d)	Select a functional block such as a crafting table or furnace from the hotbar and place it on the ground. 'k' is a number from 1 to 9 corresponding to the key for the functional block in the hotbar. 'd' is mouse press duration.
smelt_iron_ore(i_o_x, i_o_y, p_x, p_y)	Use a furnace to smelt iron ore and obtain iron ingots. 'i_o_x' and 'i_o_y' are the x and y coordinates of the iron ore in the player's inventory, 'p_x' and 'p_y' are the x and y coordinates of the planks.
turn(x, y)	Turn the view to the target. 'x' and 'y' are the horizontal and vertical pixel offset of the crosshair from the target, with positive or negative values.
turn_and_move_forward(d, x, y)	Firstly turn to the target, then move forward. 'x' and 'y' are the pixel offset of the crosshair from the target, corresponding to the horizontal and vertical directions, with positive or negative values. 'd' is the press duration of the key 'w'.

General Prompt Template

In Minecraft, player is focusing on {task_description}, requiring strategic action choices based on the environment and inventory status The crosshair position is at {crosshair_position} in the screen. If applicable, the distance between the player and the {target} is {greater than/close to/less than} the interactable range.

Here is some knowledge related to the current status that may help clarify item-behavior dependencies: {retrieved_knowledge_graph}.

Current inventory status: {retrieved_attributes_of_observable_inventory}.

Current environment status: {retrieved_attributes_of_observable_environment}.

Available actions are defined as functions with the following formats and descriptions: {skill_library}.

Previous round(s) of action decision(s): {retrieved_memory}.

Please address these questions to determine the next optimal action: {chain_of_thought}.

Based on this reasoning, decide the best next action and calculate the required parameter values. The output format must be: "Action: skill_function(*params)".

Table 9: General prompt template for action prediction.