DeepHA: Scaling Action Chains Elicits Deep Hierarchical Agents

Anonymous authors
Paper under double-blind review

Abstract

Prevailing autonomous agents are often constrained by a single, predefined action space, which limits their generalization capabilities across diverse tasks and can introduce compounding errors through decoupled policy execution. To address these limitations, we introduce the Deep Hierarchical Agent (DeepHA), a unified architecture that operates across a mixture of heterogeneous action spaces, flexibly generating actions ranging from high-level semantic skills to low-level motor controls. We further propose a Chain-of-Action (CoA) reasoning framework, which enables the agent to use higher-level abstract actions as structured ‘thoughts’ to guide the generation of more granular, subsequent actions. To manage the computational demands of this deep reasoning in long-horizon tasks, we develop a memory-efficient mechanism that dynamically compresses historical context and leverages Key-Value (KV) caching, reducing context length by approximately 75% without sacrificing performance. We conduct extensive evaluations on a new, large-scale benchmark of over 800 diverse Minecraft tasks. Results show that DHA significantly outperforms prior methods, establishing a new state-of-the-art and demonstrating superior generalization, particularly in complex, multi-step planning tasks. Our work presents a novel, unified framework for building more capable and efficient autonomous agents.

1 Introduction

Prevailing architectures for autonomous agents often adopt a hierarchical design where a high-level Large Language Model (LLM) generates an abstract action from a manually predefined action space (Driess et al., 2023; Belkhale et al., 2024b; Gu et al., 2023; Liang et al., 2022; Brohan et al., 2022b). This abstract action is then translated into low-level, executable environment actions by a separate, independently trained policy or a fixed API decoder (Qin et al., 2025; Chen et al., 2021; Wang et al., 2024a). However, this paradigm faces two critical limitations. First, it relies on a single, monolithic action space (e.g., language skills, motion primitives), yet research has shown that the optimal choice of action abstraction is highly task-dependent; an action space that excels at navigation may be ill-suited for precise object manipulation. Consequently, no single action space can enable an agent to achieve state-of-the-art performance across a diverse range of benchmarks (Wang et al., 2025; Team, 2025b). Second, the decoupled nature of the high-level LLM and the low-level policy can introduce compounding errors, where inaccuracies in the generated abstract action are magnified by an imperfect decoder (Wang et al., 2024c; Driess et al., 2023; Brohan et al., 2022b).

To overcome these constraints, we propose a paradigm shift from an agent reliant on a single, predefined action space to one that can dynamically reason and operate across multiple action spaces during inference. We introduce the **Deep Hierarchical Agent** (DeepHA), which features a high-level VLM capable of generating actions from a *mixture of heterogeneous action spaces*—spanning high-level language skills, coordinate-based grounding actions, mid-level motion commands, and low-level raw action sequences. These are interpreted by corresponding specialized policies within a Mixture-of-Policies (MoP) framework, allowing DeepHA to autonomously select and execute the most appropriate action abstraction for any given task or sub-task. This design effectively unifies the strengths of direct VLM models and structured hierarchical agents within a flexible architecture.

Furthermore, we observe that complex tasks naturally decompose into a hierarchy of actions, which we term an **action pyramid**. For instance, the instruction to `obtain a diamond` can be broken down into skills like `gather wood` and `craft tools`, each further decomposing into grounding, motion, and raw actions. Inspired by this, we propose **long action-chain reasoning through action pyramid**, a framework that enables DeepHA to explicitly generate a sequence of linked abstract actions. In this process, higher-level actions serve as guiding thoughts for the generation of subsequent, more granular actions. This structured reasoning enhances decision-making; for example, generating `press(w)` is more reliably guided by an intermediate `move forward` thought than by a distant high-level objective alone. This framework also allows for dynamically scaling the depth of reasoning, and thus the computational effort, at inference time.

While powerful, the detailed action hierarchical process can impose substantial memory demands in long-horizon tasks, potentially leading to context lengths exceeding 600k tokens. To mitigate this, we develop a **memory-efficient hierarchical mechanism**. Recognizing that high-level abstracted actions persist across many low-level steps, this mechanism dynamically condenses the history of past low-level actions and redundant observations while preserving the active, high-level actions. By synergizing with the VLM's Key-Value (KV) caching, this approach achieves approximately a 75% reduction in context memory requirements without compromising performance.

We conduct extensive experiments in Minecraft, scaling existing benchmarks from under 100 to over 800 diverse tasks to rigorously evaluate generalization. Our results demonstrate that DeepHA, particularly when augmented with CoA reasoning, establishes new state-of-the-art performance. Our main contributions are: (1) A unified Deep Hierarchical Agent (DeepHA) architecture that flexibly generates and executes multi-level, multi-modal abstract actions via a Mixture-of-Policies framework, bridging the VLA and hierarchical agent paradigms. (2) A Chain-of-Action (CoA) reasoning framework that enables explicit, structured, and hierarchical decision-making by using higher-level actions as 'thoughts' for lower-level action prediction. (3) A memory-efficient CoA mechanism that significantly reduces (by 75%) the context memory footprint for long-horizon tasks by dynamically compressing history and leveraging KV caching. (4) The establishment of new state-of-the-art performance by DeepHA on an extensive benchmark of over 800 diverse Minecraft tasks, showcasing superior generalization performance.

2 Problem Formulation and Related Works

2.1 Action Spaces in LLM-based Agents

LLM-based agents typically generate actions in a two-stage process. First, an abstracted high-level description of the action is generated based on the task instruction and the current observation. We denote such descriptions as abstract actions A. Second, this abstract action is translated to a low-level, executable action of the environment. This process can be expressed as:

$$A \sim \pi_{LM}(\cdot \mid obs, ins), a \sim \pi_{policy}(\cdot \mid obs, A) \tag{1}$$

Here, ins denotes the human-provided textual instruction, and $obs \in \mathbb{R}^{H\times W\times 3}$ represents the current visual observation. $A \sim \mathcal{A}$ signifies an *abstracted action* and is sampled from the predefined action space for LLM-based agents (Zhong et al., 2025a). The action space $\mathcal{A}$ ranges from language skills or sub-tasks, code, affordance, trajectory, goal state, latent representation, raw action, and reasoning (Liang et al., 2022; Driess et al., 2023; Belkhale et al., 2024a; Brohan et al., 2023; Zhen et al., 2024; Gu et al., 2023; Cai et al., 2023; Wang et al., 2024d; Bjorck et al., 2025). A is generated by an autoregressive model, denoted π_{LM}, conditioned on both the ins and the current obs, as shown in Eq. 1. Subsequently, a represents a *low-level, discrete action* that the agent executes in the environment. These are often considered *raw actions*; a could correspond to discrete keyboard inputs or mouse movements in computer interaction domains (Seed, 2025) and end controller position and forces in embodied robotics (Kim et al., 2024). The policy π_{policy} (Eq. 1) is responsible for generating the action a, conditioned on the abstracted action A and the current observation obs, which usually can be categorized into predefined MCP APIs (Xu and Peng, 2025), learning-based policies (Driess et al., 2023), or a rule-based parser (Team, 2025b).

This hierarchical formulation is general enough to encompass many recent Vision-Language-Action (VLA) models (Chi et al., 2023; Team et al., 2024b; Brohan et al., 2022a; 2023; Kim et al., 2024; Zheng et al., 2024; Zhang et al., 2024; Zhong et al., 2025b; Li et al., 2024b; Zhang et al., 2025; Zhu et al., 2025; Zhou et al., 2025; Chen et al., 2025). We conduct the experiments on the open-world Minecraft simulator, which has a series of hierarchical LLM-based agents and VLA models (Wang

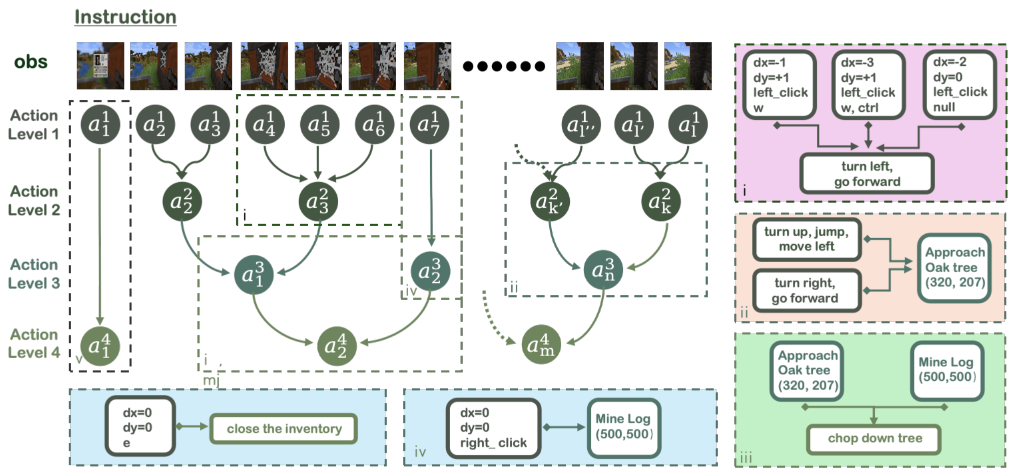


Figure 1: An illustration of the multi-level action hierarchy for the instruction "BUILD A HOUSE" in Minecraft. A sequence of visual observations (o_t) is temporally aligned with a four-tiered action space. This hierarchy ranges from **Level 4** high-level semantic skills (e.g., `Mine Log`) and **Level 3** grounding actions that specify object targets (e.g., `Approach Oak tree (320,207)`), down to **Level 2** mid-level motion policies (e.g., `turn left, go forward`), and finally to **Level 1** primitive keyboard and mouse operations (e.g., `mouseMove(-1,2)`, `press(w)`). The figure exemplifies how a single, complex goal is decomposed into progressively finer-grained sub-actions, highlighting the temporal and semantic abstraction inherent in the action space.

et al., 2023; Zhou et al., 2024; Fan et al., 2022; Deng et al., 2025; Zhao et al., 2024; Cai et al., 2023; Wang et al., 2023; 2024c;a; Li et al., 2024c; Cai et al., 2024c;b; Wang et al., 2024d; Zhou et al., 2024; Zheng et al., 2023).

2.2 HIERARCHY BETWEEN ACTION SPACES

We identify a distinct hierarchical structure within the action spaces of autonomous agents. This hierarchy is evident along two primary dimensions: temporal and semantic. Temporally, a single high-level action corresponds to a sequence of multiple low-level actions executed over time. Semantically, a high-level action functions as a conceptual abstraction, integrating a series of consecutive low-level actions into a single, meaningful command.

For example, in a gaming environment, the high-level motion command `go forward` is realized through a sequence of primitive, low-level actions, such as multiple `press(w)` keystrokes. This hierarchical principle extends beyond gaming to a wide range of applications for LLM-based agents. Consider a high-level command such as `search(query)`. Its execution is decomposed into a series of fundamental GUI operations: 1) opening a web browser, 2) typing the query into the search bar, and 3) pressing `Enter` to initiate the search.

It is crucial to distinguish this hierarchical action structure from planning paradigms such as the Tree of Thought (ToT) (Yao et al., 2023). Our action hierarchy is fundamentally a model of execution and abstraction, defining how a selected high-level intention is translated into a concrete sequence of low-level motor controls. In contrast, the Tree of Thought is a mechanism for deliberation and planning, which explores a branching tree of potential reasoning paths and action sequences to decide what the best overall strategy is. In essence, while ToT helps an agent decide which high-level goal to pursue from among multiple alternatives (e.g., "should I mine wood or find food?"), our action hierarchy defines the sub-steps required to accomplish the single goal once it has been chosen (e.g., "to mine wood, I must approach the tree, then swing the axe...").

The formation of this action hierarchy can be approached from two perspectives. A **top-down** approach involves predefining the action layers and employing a Large Language Model (LLM) for task decomposition and planning. In this paradigm, a complex, high-level action is systematically broken down into a sequence of simpler, executable sub-actions. Conversely, a **bottom-up** approach allows for the emergent formation of the hierarchy. Using algorithms inspired by Byte Pair Encoding (Sennrich et al., 2015) or other LLM-driven heuristics (Deng et al., 2025), the agent can

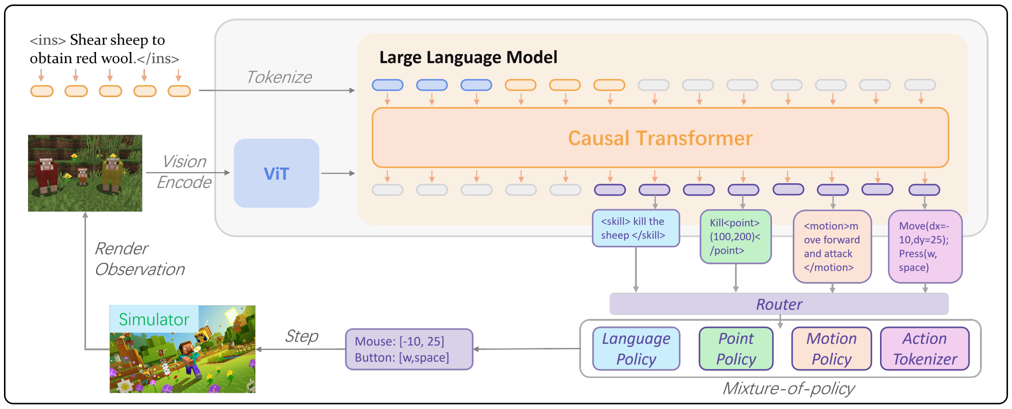

<ins> Shear sheep to obtain red wool.</ins>
Tokenize
Large Language Model
Causal Transformer
Vision Encode
ViT
Render Observation
<skill> kill the sheep </skill>
Kill<point>(100,200)</point>
<motion>move forward and attack</motion>
Move(dx=-10,dy=25); Press(w, space)
Router
Simulator
Step
Mouse: [-10, 25] Button: [w,space]
Language Policy
Point Policy
Motion Policy
Action Tokenizer
Mixture-of-policy

Figure 2: **Architecture of the Deep Hierarchical Agent (DeepHA).** Fed with the visual observations and language instructions, the vision language model autoregressively generates a hierarchical action chain, which can range from a high-level skill (e.g., `<skill>chop down a tree</skill>`) to a mid-level coordinate-based command (e.g., `<point>mine log (500, 500)</point>`) or a low-level motor action (e.g., `<motion>turn left</motion>`). A router dispatches the chosen action to a corresponding specialized policy within a Mixture-of-Policies module for execution in the simulator, thereby closing the agent-environment loop.

automatically aggregate sequences of frequently co-occurring low-level actions to form new, more abstract, and composite actions.

We refer to this structured representation as an **action pyramid**. The detailed algorithms for the construction and utilization of this action pyramid are elaborated in the Appendix B.3.

3 Deep Hierarchical Agent

In this section, we first present the model structure of Deep Hierarchical Agents (DeepHA) in Section 3.1, which supports multiple action spaces. Next, we explain how we utilize the action pyramid to enhance inference-time computation by scaling the action chains (Section 3.2). We also introduce three inference modes based on the action pyramid. Finally, we emphasize the hierarchical memory mechanism, which efficiently manages memory content according to the action hierarchy (Section 3.3).

3.1 Model Structure

Our model architecture is illustrated in Figure 2. While previous LLM-based agents typically operate within a monolithic abstract action space A (as described in Equation 1), our agent is designed to generate actions from $\mathcal{A}$ **mixture of heterogeneous action spaces**, denoted as $\mathcal{A}_k$. This allows for greater flexibility and specialization in action generation.

The core of our hierarchical agent consists of a high-level Vision Language Model (VLM) π_{LM}, and a low-level Mixture-of-Policies (MoP) $\{\pi^k_{policy}\}_k$. The VLM first generates a **typed abstract action** A_k, where k is an index identifying the chosen action space. A router then directs A_k to the corresponding specialized low-level policy π^k_{policy} to be decoded into an executable, low-level action a. This two-stage generation process can be formulated as:

$$(k, A_k) \sim \pi_{\text{LM}}(\cdot \mid \text{obs}, \text{ins}), \quad a \sim \pi^k_{\text{policy}}(\cdot \mid \text{obs}, A_k) \tag{2}$$

where $k \in 1, \ldots, K$ serves as the routing token for the low-level policies. In our implementation, the high-level VLM component adopts the Qwen2-VL architecture (Wang et al., 2024b). A key feature of our design is that π_{LM} can generate diverse types of abstract actions, each corresponding to a distinct action space $\mathcal{A}_k$. These include high-level language skills $A^s \in \mathcal{A}^s$ (e.g., `<skill>chop down a tree</skill>`), coordinate-based grounding actions $A^g \in \mathcal{A}^g$ (e.g., `<point>mine log (500, 500)</point>`), and direct motion commands $A^m \in \mathcal{A}^m$ (e.g., `<motion>turn left</motion>`). A comprehensive list of these action spaces is provided in Appendix B.3. To enable this functionality, we do not modify the VLM's internal structure but instead repurpose several reserved tokens in the

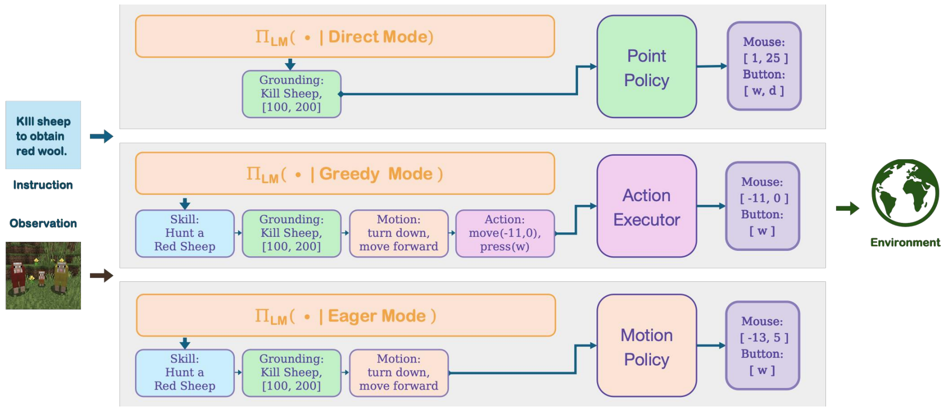

Kill sheep to obtain red wool.
Instruction
Observation
Π_LM(• | Direct Mode)
Grounding: Kill Sheep, [100, 200]
Point Policy
Mouse: [1, 25] Button: [w, d]
Π_LM(• | Greedy Mode)
Skill: Hunt a Red Sheep
Grounding: Kill Sheep, [100, 200]
Motion: turn down, move forward
Action: move(-11,0), press(w)
Action Executor
Mouse: [-11, 0] Button: [w]
Environment
Π_LM(• | Eager Mode)
Skill: Hunt a Red Sheep
Grounding: Kill Sheep, [100, 200]
Motion: turn down, move forward
Motion Policy
Mouse: [-13, 5] Button: [w]

Figure 3: DeepHA with **Direct** inference mode allows users to specify the generated abstracted action space. **Greedy** mode generates abstracted actions sequentially as thoughts, ultimately producing raw actions. **Eager** mode truncates based on greedy and uses the policy to decode after reaching the specified abstracted action.

VLM vocabulary to represent these distinct action types, followed by post-training on the pretrained VLM, a technique inspired by prior work (Brohan et al., 2023; Kim et al., 2024).

The **router** for the low-level policies is an implicit module that uses the index k generated by the VLM to dynamically dispatch the abstract action A_k to its corresponding expert. For instance, a motion command A^m is routed to a dedicated motion policy π^m_{policy} specialized for that task. Each expert policy network utilizes a lightweight CNN and Transformer-XL architecture (Dai et al., 2019), following established practices Lifshitz et al. (2024); Baker et al. (2022); Cai et al. (2024a). Further details on the policy network structures are available in Appendix A.

3.2 Chain-of-Action Scales Inference-time Computation

We further enable DeepHA with end-to-end interaction with environments by the unification of the high-level policy π_{LM} and the low-level policies π_{policy} within a single VLM, enabling end-to-end training. Instead of employing separate models, our VLM is trained to auto-regressively generate a sequence containing both the abstract and low-level actions. This process can be formulated as:

$$A_k, a \sim \pi_{\text{VLM}}(\cdot \mid \text{obs}, \text{ins}) \tag{3}$$

Here, the joint probability distribution is factorized within the VLM as $\pi_{LM}(A_k \mid obs, ins) \cdot \pi_{LM}(a \mid A_k, obs, ins)$. In this framework, the abstract action A_k functions as an explicit Chain-of-Action (CoA), serving as an intermediate "thought" that guides the generation of the final, executable action a. This end-to-end architecture simplifies the modeling process, allowing the agent to learn the mapping from high-level reasoning to low-level environmental interactions directly. And the model can choose to use the light-weight mixture-of-policy to decode the A_k into a or auto-regressively generate the final low-level action a to interact with the environment. We categorize the inference of DeepHA into three modes based on whether VLM uses action-chain for thinking and whether it uses mixture-of-policy to decode actions. Several cases are presented in Appendix D.

Direct Mode. In this mode, the VLM generates an action A_k from a single, pre-specified abstract action space $\mathcal{A}_k$. The choice of $\mathcal{A}_k$ can be manually set by a system prompt. This approach minimizes the VLM's per-step computational load by focusing generation on a single level of abstraction, offering flexibility to switch between action types (e.g., a for crafting vs. A^m for navigation).

Greedy Mode. This mode executes the complete top-down CoA reasoning process. The VLM sequentially generates the full action hierarchy, such as $A^s \rightarrow A^g \rightarrow A^m \rightarrow a$. Each higher-level action serves as an explicit thinking step guiding the generation of the subsequent, more concrete action. While this ensures thorough reasoning, it is the most computationally intensive mode, as the full chain is generated at each decision step.

Eager Mode. This mode balances reasoning depth and computational cost by enabling early termination of the generation chain. It initiates the CoA process as in Greedy Mode (e.g., generating A^s and A^g as intermediate thoughts), but halts generation once an executable intermediate action

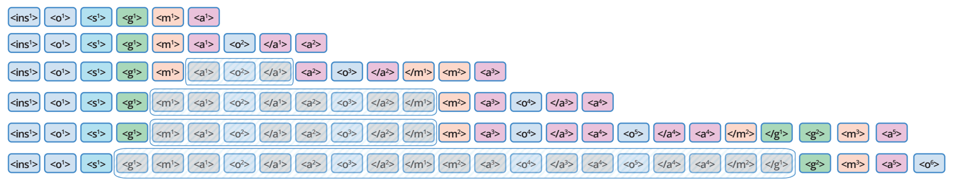


Figure 4: Illustration of the inference process incorporating the efficient memory mechanism with the Chain-of-Action. Each row depicts a snapshot of the token sequence of agent inference. <ins>: Initial instruction. <o>: Visual observation. <s>, <g>, <m>, <a>: Start tokens for skill, grounding, motion, and raw actions. Superscripts denote the action's sequence ID. </g>, </m>, </a>: Stop tokens that signal the completion of an action. Shaded Tokens: History pruned by the efficient memory mechanism.

(e.g., A^m) is produced. This allows the agent to benefit from high-level thinking while avoiding the full computational cost of the Greedy approach. We employ two distinct termination mechanisms for this mode: (1) **Manually-Controlled Termination:** Used for controlled experiments, this mechanism halts generation upon detecting a pre-specified action tag (e.g., `</motion>`). This allows for rigorous ablation studies by isolating the performance of specific action spaces. (2) **Learned Dynamic Termination:** To grant the agent full autonomy, we fine-tune the VLM to generate a special token, `<|Eager-Stop|>`, when it determines an intermediate action is sufficient for execution. This enables the agent to dynamically adapt its reasoning depth and action modality based on the current context.

3.3 Memory-Efficient Chain-of-Action

While the deep reasoning of the Greedy Mode enhances performance, its sequential generation of the full CoA at each step leads to a rapid expansion of the VLM's context length in long-horizon tasks. This can exceed hundreds of thousands of tokens, significantly impairing the model's reasoning capabilities due to issues like "lost in the middle," where critical instructions are overlooked (Li et al., 2024a; Liu et al., 2025). Furthermore, the context becomes saturated with redundant observations and low-level actions from consecutive steps where the environment changes minimally (Seed, 2025).

To address this, we introduce a memory-efficient CoA mechanism that dynamically manages the VLM context. As illustrated in Figure 4, this approach leverages the hierarchical nature of tasks, where high-level abstract actions persist over multiple low-level steps. The process unfolds as follows: 1) **Initial Generation and Re-use:** The agent initially generates a complete CoA, establishing a high-level action (e.g., A_1^g) as a contextual anchor. This high-level action is reused across subsequent steps while the agent generates the necessary low-level actions (e.g., $a_1, a_2, \ldots$). 2) **Memory Compression:** Upon the completion of an intermediate action, signaled by a stop token (e.g., `</g>`), our mechanism triggers a memory compression step. It prunes the detailed execution history—including the intermediate actions and their corresponding visual observations (depicted as shaded tokens in Figure 4)—from the input sequence, while preserving the high-level semantic goal. Consequently, the agent maintains a concise yet informative context, enabling efficient long-horizon reasoning. And this dynamic context management yields significant efficiency gains by synergizing with the VLM.

Efficient Context Inference with KV Caching. By pruning tokens from the input sequence, we drastically reduce the computational load for the subsequent generation step. Critically, the computational influence of the pruned history is preserved in the VLM's Key-Value (KV) cache. This strategy avoids the costly re-computation (prefilling) of past states, allowing the agent to benefit from a long-term history without the full cost of reprocessing it at every step.

Training with Causal Attention Masks. This inference-time process informs our training methodology. We employ causal attention masks to train the model to understand that for an ongoing high-level goal, it should attend to the initial goal definition rather than the full, detailed history of already-completed sub-tasks.

Table 1: Evaluation results of Minecraft agents on over 800 tasks. For each task category, we report the success rate on a representative task (indicated by the icon), the percentage of Finished Tasks within the category (*FT*), and the Average Success Rate across all tasks in that category (*ASR* ± standard deviation). Results highlighted in blue represent the second-best performances, while those in red indicate the state-of-the-art performance for each metric across all agents. '-' signifies tasks where the agent failed to achieve any success.

Method	Mine Blocks			Kill Entities			Craft Items			Smelt Items		
		FT ↑	*ASR* ↑		*FT* ↑	*ASR* ↑		*FT* ↑	*ASR* ↑		*FT* ↑	*ASR* ↑
Instruction-Conditioned Policies												
VPT (Baker et al., 2022)	20.0	30.7	$6.0^{\pm 11.4}$	10.0	24.6	$3.6^{\pm 7.7}$	N/A	6.7	$0.8^{\pm 3.3}$	0.0	0.0	$0.0^{\pm 0.0}$
STEVE-I (Lifshitz et al., 2024)	50.0	29.4	$8.0^{\pm 17.0}$	0.0	14.7	$3.9^{\pm 12.0}$	0.0	16.4	$3.2^{\pm 8.4}$	0.0	0.0	$0.0^{\pm 0.0}$
ROCKET-1 (Cai et al., 2024a)	60.0	57.5	$18.9^{\pm 24.3}$	60.0	63.9	$27.9^{\pm 29.3}$	-	-	-	-	-	-
JARVIS-VLA (Li et al., 2025)	55.0	55.3	$30.0^{\pm 35.4}$	60.0	61.9	$18.5^{\pm 22.7}$	40.0	**74.3**	$25.1^{\pm 23.9}$	60.0	**92.3**	$65.4^{\pm 32.3}$
Hierarchical Agents												
LanguageHA (Driess et al., 2023)	60.0	31.3	$11.3^{\pm 14.5}$	0.0	12.8	$6.5^{\pm 9.3}$	5.0	19.3	$6.3^{\pm 9.2}$	0.0	0.0	$0.0^{\pm 0.0}$
PointHA (Cai et al., 2024a)	**90.0**	61.0	$37.1^{\pm 38.5}$	50.0	**90.1**	$26.5^{\pm 23.4}$	15.0	27.5	$6.7^{\pm 10.8}$	10.0	37.0	$5.0^{\pm 8.7}$
MotionHA (Belkhale et al., 2024b)	70.0	51.0	$27.4^{\pm 35.2}$	20.0	29.5	$4.3^{\pm 10.8}$	0.0	0.0	$0.0^{\pm 0.0}$	0.0	0.0	$0.0^{\pm 0.0}$
LatentHA (Wang et al., 2024d)	70.0	54.2	$24.4^{\pm 31.1}$	50.0	24.6	$8.5^{\pm 17.9}$	0.0	19.1	$3.0^{\pm 7.5}$	0.0	2.2	$0.2^{\pm 1.5}$
OpenHA (Wang et al., 2025)	80.0	**67.3**	$30.1^{\pm 13.9}$	**70.0**	62.6	$32.5^{\pm 9.2}$	**80.0**	58.8	$31.9^{\pm 13.7}$	**75.0**	**92.3**	$66.6^{\pm 28.1}$
Ours												
DeepHA	**100.0**	**82.4**	$48.9^{\pm 43.6}$	**90.0**	**92.3**	$38.9^{\pm 45.3}$	**75.0**	**85.6**	$43.2^{\pm 26.5}$	**90.0**	**93.5**	$82.4^{\pm 22.6}$

4 Experiments and Discussions

4.1 Experimental Setups

Simulator and Benchmarks. We employ Minecraft (Version 1.16.5) as our primary test environment (Guss et al., 2019). The agent's observation space consists solely of first-person RGB visual images, with a resolution of $360 \times 640 \times 3$. The action space comprises discretized, human-like mouse and keyboard controls. Specifically, it includes: mouse displacement, mouse clicks and keyboard inputs. Further detailed specifications of the observation and action spaces are provided in Appendix C.1. We utilized OpenHA's benchmarks (Wang et al., 2025) to evaluate our agent model, which includes over 800 tasks across embodied mineblock, GUI crafting, and combat tasks, among others. Our primary metrics are **Success Rate** and **Finished Tasks**. In our final analysis, we report the average success rate and average finished tasks, aggregated across all tasks within each of the three benchmark groups.

Baselines. We evaluate our proposed method against two main categories of baseline agents. Instruction-Conditioned Policies: This group includes representative models such as OpenAI VPT (Baker et al., 2022), the language-conditioned STEVE-I (Lifshitz et al., 2024), the visual-prompt-conditioned ROCKET-1 (Cai et al., 2024a), and the Vision-Language-Action (VLA) model JARVIS-VLA (Li et al., 2025). These methods typically encode the input instruction into an embedding and directly predict actions. Hierarchical Agents: This category consists of agents with a two-level structure, where a high-level Vision-Language Model (VLM) processes instructions and observations to generate an abstract intermediate action (e.g., a language skill, visual coordinates, motion primitive, or latent code), which is then executed by a low-level policy. We consider four representative agents according to the modality of the abstract action: LanguageHA (Driess et al., 2023) and MotionHA (Belkhale et al., 2024b), which use fine-tuned Qwen2-VL-7B (Wang et al., 2024b) as the high-level VLM paired with specialized low-level policies (Baker et al., 2022); PointHA (Cai et al., 2024a), which employs Qwen2-VL-7B as the high-level VLM with ROCKET-1 (Cai et al., 2024a) as its low-level policy; and LatentHA, operating within the OmniJARVIS framework (Wang et al., 2024d), which similarly uses Qwen2-VL-7B as its high-level component with FSQ-GROOT (Van Den Oord et al., 2017) as its low-level policy (Cai et al., 2024c). Our comparisons benchmark performance against both flat policy structures and contemporary hierarchical approaches.

For fairness, all models are retrained on the same expert datasets via imitation learning and are constrained to use comparable computational resources.

Table 2: Ablation experiments on action chain depth revealed that as the chain deepens, the agent's performance consistently improves when utilizing the same outcome action.

Outcome	Depth	Action Chain	Mine Block ASR ↑
Raw	1	Raw	$30.0^{\pm 35.4}\%$
Raw	2	Motion → Raw	$39.2^{\pm 31.7}$
Raw	2	Grounding → Raw	$42.6^{\pm 39.3}\%$
Raw	3	Grounding → Motion → Raw	$48.9^{\pm 43.6}\%$

Table 3: DeepHA with efficient memory achieves a 3x reduction in memory tokens while maintaining performance comparable to full memory.

Method	Context ↓	*ASR* ↑	*FT* ↑
No Memory	385	37.1	68.3
Full Memory	7948	38.5	78.5
Efficient Memory	1976	39.6	77.3

4.2 Main Results

To rigorously assess generalization, we expanded the evaluation benchmark tenfold from 80 to over 800 distinct tasks, categorized into **Mine Blocks**, **Kill Entities**, **Craft Items**, and **Smelt Items** (details in Appendix C.2). This challenging suite reveals the brittleness of prior methods; for instance, ROCKET-1 (Cai et al., 2024a), previously state-of-the-art with a 93% success rate on its original mining benchmark, achieves only 18.9% ASR on our expanded set. To better measure capability breadth, we also introduce a new metric, *Finished Tasks (FT)*, which quantifies the percentage of unique tasks an agent can complete at least once.

As presented in Table 1, our DeepHA establishes a new state-of-the-art, demonstrating superior performance and generalization. The key insight from these results is that DeepHA consistently surpasses not only generalist models but also the strongest specialized baselines in their respective domains. This validates our core hypothesis that an agent capable of dynamically operating across a mixture of action spaces is fundamentally more robust and capable.

Specifically, in **Mine Blocks** and **Kill Entities**, tasks that favor precise spatial reasoning, DeepHA outperforms the specialized PointHA by significant margins in both Average Success Rate (ASR) (+11.8 and +12.4 points, respectively) and FT. This demonstrates that DeepHA's flexible architecture can adopt a point-centric strategy when optimal, while enhancing it with deeper reasoning. The advantage is even more pronounced in complex, multi-step categories. In **Craft Items** and **Smelt Items**, which require long-horizon planning, DeepHA achieves an ASR of 43.2% and 82.4%, substantially exceeding the next-best hierarchical agent, OpenHA, by over 11 and 15 points.

These results confirm that by transcending the limitations of a single, fixed action modality, DeepHA achieves a new level of generalization. Its ability to dynamically select the appropriate level of action abstraction—from precise coordinates for mining to high-level skills for crafting—allows it to excel across a far wider variety of tasks than any single-paradigm agent.

4.3 Analysis and Discussions

4.3.1 Ablation experiments on the action chain depth

We conducted a controlled ablation to quantitatively measure how performance varies with the depth of the Chain-of-Action (CoA). To ensure a fair comparison and isolate the effect of action depth itself, the final outcome action is always a Raw Action. The only variable is the length and composition of the preceding action chain generated before producing the final output. Results are shown in Table 2.

Directly generating a Raw Action with depth 1 serves as the baseline, achieving an ASR of 30.0%. Increasing the action depth to 2 significantly boosts performance: incorporating a Motion action raises ASR to 39.2%, while using a more informative Grounding action increases it further to 42.6%. Finally, the full three-step action chain (Grounding → Motion → Raw) achieves the best performance, reaching 48.9%.

This step-by-step improvement provides strong quantitative evidence for our central claim: greater action depth directly correlates with higher task success rates. Each additional level in the Chain-of-Action supplies valuable context that refines the final decision, validating the core motivation for our architecture and the principle of scaling inference-time computation for more effective deliberation.

Table 4: Comparison of inference modes across different outcome actions. The maximum context token length is set as 8k.

Inference Mode	Outcome Action	MineBlock ASR ↑	Craft Item ASR ↑	Inference FPS ↑	Avg. Memory Context ↓
Direct	Motion	$32.9^{\pm 38.7}$	$2.7^{\pm 8.6}$	6.7	8000
Eager	Motion	$37.6^{\pm 33.3}$	$\mathbf{11.5}^{\pm \mathbf{13.6}}$	4.9	**1934**
Direct	Grounding	$37.7^{\pm 35.8}$	$8.0^{\pm 11.4}$	6.9	8000
Eager	Grounding	$\mathbf{40.7}^{\pm \mathbf{28.3}}$	$\mathbf{23.4}^{\pm \mathbf{19.6}}$	6.4	**1926**
Direct	Raw	$34.5^{\pm 31.9}$	$26.6^{\pm 25.1}$	2.3	8000
Greedy	Raw	$\mathbf{48.9}^{\pm \mathbf{43.6}}$	$\mathbf{43.2}^{\pm \mathbf{26.5}}$	1.2	**1976**

4.3.2 Experiment on the Efficient Memory Mechanism

In this section, we examine how our efficient memory mechanism affects both the performance and efficiency of DeepHA. Specifically, we evaluate the proposed memory-efficient CoA mechanism by analyzing its impact on average context length, Average Success Rate (ASR), and Average Task Finish Rate (FT). The comparison is conducted across three configurations: a minimal-history baseline DeepHA (No Memory), a version retaining full uncompressed trajectory history (Full Memory), and our proposed trajectory-compressed variant (Efficient Memory).

The results in Table 3 compellingly demonstrate the benefits of our approach. DeepHA (Efficient Memory) reduces the average context length to 494 tokens from 1987 in the Full Memory setting, bringing it close to the No Memory baseline's 385 tokens. Importantly, this significant context reduction is achieved not only without performance degradation but with a slight improvement in ASR: DeepHA (Efficient Memory) achieves 39.6%, surpassing both Full Memory (38.5%) and No Memory (37.1%). Furthermore, its Average Task Finish Rate (FT) of 77.3% is nearly on par with Full Memory (78.5%) and markedly better than No Memory (68.3%).

These findings indicate that our mechanism not only yields substantial computational efficiency—implying faster inference and lower resource usage—but also enhances agent scalability for long-horizon tasks while maintaining, or even improving, task success. This improvement stems from more effective memory management: redundant low-level details are pruned, while strategically relevant high-level information is preserved, enabling more focused and robust reasoning.

4.3.3 Comparisons on the Inference Mode of DeepHA

We further conduct experiments to evaluate the inference speed and performance across different inference modes within the DeepHA. The experimental results are shown in the Table 4.

According to the experimental results, we find that Eager Mode Outperforms Direct Mode based on the same outcome action space. The partial reasoning in Eager Mode provides a clear performance benefit. For example, Eager (Motion) achieves 37.6% ASR versus 32.9% from Direct (Motion), proving the value of the initial CoA "thoughts." Eager Mode Offers an Excellent Performance/Efficiency Trade-off vs. Greedy Mode: Eager Mode is substantially more efficient, operating up to 5x faster than Greedy Mode on inference speed(e.g., 6.4 FPS vs. 1.2 FPS). It achieves this speed while retaining a large portion of Greedy Mode's performance (e.g., 40.7% ASR vs. 48.9% ASR for Grounding/Greedy comparison), making it a highly practical choice. This table quantitatively demonstrates that Eager Mode successfully balances computational cost with high performance.

5 Conclusions

In this paper, we introduced the Deep Hierarchical Agent (DeepHA), a novel vision-language-action model that significantly advances instruction-following capabilities in complex, open-world environments like Minecraft. DeepHA integrates a flexible, multi-level action hierarchy with a Chain-of-Action (CoA) reasoning mechanism, enabling dynamic selection of control modalities and adaptive reasoning depth. Furthermore, DeepHA has a memory-efficient CoA mechanism that significantly reduces context memory for long-horizon tasks while maintaining performance. Extensive evaluations on a diverse benchmark of over 800 Minecraft tasks confirm that DeepHA achieves state-of-the-art performance and generalization, offering a powerful and computationally adaptable framework for developing open-world agents.

REFERENCES

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

B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

S. Belkhale, T. Ding, T. Xiao, P. Sermanet, Q. Vuong, J. Tompson, Y. Chebotar, D. Dwibedi, and D. Sadigh. Rt-h: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024a.

S. Belkhale, T. Ding, T. Xiao, P. Sermanet, Q. Vuong, J. Tompson, Y. Chebotar, D. Dwibedi, and D. Sadigh. Rt-h: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024b.

J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022a.

A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *6th Annual Conference on Robot Learning*, 2022b.

A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

S. Cai, Z. Wang, X. Ma, A. Liu, and Y. Liang. Open-world multi-task control through goal-aware representation learning and adaptive horizon prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13734–13744, 2023.

S. Cai, Z. Wang, K. Lian, Z. Mu, X. Ma, A. Liu, and Y. Liang. Rocket-1: Mastering open-world interaction with visual-temporal context prompting. *arXiv preprint arXiv:2410.17856*, 2024a.

S. Cai, B. Zhang, Z. Wang, H. Lin, X. Ma, A. Liu, and Y. Liang. Groot-2: Weakly supervised multi-modal instruction following agents. *arXiv preprint arXiv:2412.10410*, 2024b.

S. Cai, B. Zhang, Z. Wang, X. Ma, A. Liu, and Y. Liang. Groot: Learning to follow instructions by watching gameplay videos. In *The Twelfth International Conference on Learning Representations*, 2024c.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

P. Chen, P. Bu, Y. Wang, X. Wang, Z. Wang, J. Guo, Y. Zhao, Q. Zhu, J. Song, S. Yang, et al. Combatvla: An efficient vision-language-action model for combat tasks in 3d action role-playing games. *arXiv preprint arXiv:2503.09527*, 2025.

C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

J. Deng, Z. Wang, S. Cai, A. Liu, and Y. Liang. Open-world skill discovery from unsegmented demonstrations. *arXiv preprint arXiv:2503.10684*, 2025.

D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems Datasets and Benchmarks*, 2022.

J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

M. Li, Z. Wang, K. He, X. Ma, and Y. Liang. Jarvis-vla: Post-training large-scale vision language models to play visual games with keyboards and mouse. *arXiv preprint arXiv:2503.16365*, 2025.

T. Li, G. Zhang, Q. D. Do, X. Yue, and W. Chen. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*, 2024a.

X. Li, P. Li, M. Liu, D. Wang, J. Liu, B. Kang, X. Ma, T. Kong, H. Zhang, and H. Liu. Towards generalist robot policies: What matters in building vision-language-action models. *arXiv preprint arXiv:2412.14058*, 2024b.

Z. Li, Y. Xie, R. Shao, G. Chen, D. Jiang, and L. Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *arXiv preprint arXiv:2408.03615*, 2024c.

J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

S. Lifshitz, K. Paster, H. Chan, J. Ba, and S. McIlraith. Steve-1: A generative model for text-to-behavior in minecraft. *Advances in Neural Information Processing Systems*, 36, 2024.

J. Liu, D. Zhu, Z. Bai, Y. He, H. Liao, H. Que, Z. Wang, C. Zhang, G. Zhang, J. Zhang, Y. Zhang, Z. Chen, H. Guo, S. Li, Z. Liu, Y. Shan, Y. Song, J. Tian, W. Wu, Z. Zhou, R. Zhu, J. Feng, Y. Gao, S. He, Z. Li, T. Liu, F. Meng, W. Su, Y. Tan, Z. Wang, J. Yang, W. Ye, B. Zheng, W. Zhou, W. Huang, S. Li, and Z. Zhang. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*, 2025.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

Y. Qin, Y. Ye, J. Fang, H. Wang, S. Liang, S. Tian, J. Zhang, J. Li, Y. Li, S. Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

B. Seed. Ui-tars-1.5. `https://seed-tars.com/1.5`, 2025.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *ArXiv*, abs/1508.07909, 2015. URL `https://api.semanticscholar.org/CorpusID:1114678`.

G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024a.

O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024b.

O. T. Team. Open Thoughts, Jan. 2025a.

S. T. Team. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv: 2509.02544*, 2025b.

A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a.

P. Wang, S. Bai, S. Tan, S. Wang, Z. Fan, J. Bai, K. Chen, X. Liu, J. Wang, W. Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024b.

Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, Y. Liang, and T. CraftJarvis. 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, 2023.

Z. Wang, S. Cai, A. Liu, Y. Jin, J. Hou, B. Zhang, H. Lin, Z. He, Z. Zheng, Y. Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024c.

Z. Wang, S. Cai, Z. Mu, H. Lin, C. Zhang, X. Liu, Q. Li, A. Liu, X. Ma, and Y. Liang. Omnijarvis: Unified vision-language-action tokenization enables open-world instruction following agents. *Advances in Neural Information Processing Systems*, 2024d.

Z. Wang, M. Li, K. He, X. Wang, Z. Mu, A. Liu, and Y. Liang. Openha: A series of open-source hierarchical agentic models in minecraft. *arXiv preprint arXiv:2509.13347*, 2025.

R. Xu and J. Peng. A comprehensive survey of deep research: Systems, methodologies, and applications. *arXiv preprint arXiv:2506.12594*, 2025.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

J. Zhang, Y. Guo, X. Chen, Y.-J. Wang, Y. Hu, C. Shi, and J. Chen. Hirt: Enhancing robotic control with hierarchical robot transformers. *arXiv preprint arXiv:2410.05273*, 2024.

J. Zhang, Y. Guo, Y. Hu, X. Chen, X. Zhu, and J. Chen. Up-vla: A unified understanding and prediction model for embodied agent. *arXiv preprint arXiv:2501.18867*, 2025.

G. Zhao, K. Lian, H. Lin, H. Fu, Q. Fu, S. Cai, Z. Wang, and Y. Liang. Optimizing latent goal by learning from trajectory preference. *arXiv preprint arXiv:2412.02125*, 2024.

H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, and C. Gan. 3d-vla: A 3d vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024.

R. Zheng, Y. Liang, S. Huang, J. Gao, H. Daumé III, A. Kolobov, F. Huang, and J. Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist robotic policies. *arXiv preprint arXiv:2412.10345*, 2024.

S. Zheng, J. Liu, Y. Feng, and Z. Lu. Steve-eye: Equipping llm-based embodied agents with visual perception in open worlds. *arXiv preprint arXiv:2310.13255*, 2023.

Y. Zhong, F. Bai, S. Cai, X. Huang, Z. Chen, X. Zhang, Y. Wang, S. Guo, T. Guan, K. N. Lui, et al. A survey on vision-language-action models: An action tokenization perspective. *arXiv preprint arXiv:2507.01925*, 2025a.

Y. Zhong, X. Huang, R. Li, C. Zhang, Y. Liang, Y. Yang, and Y. Chen. Dexgraspvla: A vision-language-action framework towards general dexterous grasping, 2025b.

E. Zhou, Y. Qin, Z. Yin, Y. Huang, R. Zhang, L. Sheng, Y. Qiao, and J. Shao. Minedreamer: Learning to follow instructions via chain-of-imagination for simulated-world control. *arXiv preprint arXiv:2403.12037*, 2024.

Z. Zhou, Y. Zhu, M. Zhu, J. Wen, N. Liu, Z. Xu, W. Meng, R. Cheng, Y. Peng, C. Shen, et al. Chatvla: Unified multimodal understanding and robot control with vision-language-action model. *arXiv preprint arXiv:2502.14420*, 2025.

M. Zhu, Y. Zhu, J. Li, Z. Zhou, J. Wen, X. Liu, C. Shen, Y. Peng, and F. Feng. Objectvla: End-to-end open-world object manipulation without demonstration, 2025.

THE USE OF LARGE LANGUAGE MODELS (LLMS)

In preparing this manuscript, we utilized Large Language Models (LLMs) exclusively for linguistic support, including proofreading, grammar correction, and refinement of phrasing. The LLMs were not involved in generating scientific content, designing experiments, or interpreting results. All conceptual development, data analysis, and conclusions are original to the authors. Thus, the application of LLMs was limited to improving readability and did not compromise the validity or integrity of the research findings.

A DHA MODEL DETAILS

A.1 VISION LANGUAGE MODELS

The core of our DHA's high-level reasoning and multi-modal understanding is a pre-trained Vision-Language Model (VLM). For this, we utilize **Qwen2-VL-7B** (Wang et al., 2024b) as our base model. We conduct post-training on this VLM to adapt it to the Minecraft domain and the task of generating hierarchical actions, but we make no fundamental modifications to its underlying architecture (e.g., its Transformer layers or vision encoder structure). This VLM is responsible for processing both textual instructions (e.g., user commands) and visual observations (i.e., first-person perspective images from the Minecraft environment) to produce a selected abstracted action.

Inspired by the methodology presented in RT-2 (Brohan et al., 2023), we integrate action generation capabilities into the VLM by re-purposing a subset of existing, typically infrequently used, tokens within its language tokenizer's vocabulary. These tokens are endowed with specific semantics corresponding to environment-related actions or structured action components. This approach allows the VLM to output diverse types of abstracted actions (as detailed in Section **??**) directly within its generative output space, without requiring changes to its core vocabulary size or output head architecture. The VLM thus learns to map complex multi-modal inputs to these specialized action tokens as part of the hierarchical decision-making process.

A.2 MIXTURE OF POLICIES

Given that the high-level VLM in DHA can generate various distinct types of abstracted actions (i.e., language skills A^s, grounding actions A^g, motion actions A^m, and raw action specifications A^r), a flexible mechanism is required to translate these diverse abstractions into concrete, environment-executable actions. To achieve this, we employ a **Mixture of Policies (MoP)** architecture for the low-level control module, which conceptually resembles a Mixture-of-Experts (MoE) system.

This MoP module consists of multiple specialized low-level policy networks, each an "expert" in handling a specific type of abstracted action. These expert policies are structured in parallel. A crucial component of this design is a **Router module**. When the high-level VLM generates an abstracted action, the Router inspects this output (typically by identifying the type or category of the generated abstracted action, e.g., via special tokens indicating if it's an A^s, A^g, A^m, or A^r) and dispatches it to the corresponding expert policy for further processing and execution.

Specifically, each low-level expert policy within the MoP module shares a common architectural foundation but can be further specialized:

- **Core Architecture:** The policies are generally based on the **Transformer-XL** (Dai et al., 2019) architecture, enabling them to handle sequences and maintain memory where necessary.
- **Visual Processing:** Each policy that requires visual input (e.g., for grounding or fine-grained control based on the current view) incorporates an **Impala CNN encoder** (Baker et al., 2022) to process the raw visual image observations (obs_t) and convert them into fixed-dimensional visual embeddings. These embeddings are then provided as input to the Transformer-XL policy body.
- **Parameter Size:** Each individual expert policy model has approximately **100 million parameters**.
- **Specializations and Foundations:**

- The **Grounding-based Policy** (for A^g) is designed based on principles from **Rocket-1** (Cai et al., 2024a), likely adept at interpreting coordinate-based instructions or visual targets and translating them into navigational actions.
- The **Motion-based Policy** (for A^m) and the **Language Skill-based Policy** (for A^s) draw inspiration and architectural elements from **VPT (Video PreTraining)** (Baker et al., 2022), suggesting they are well-suited for translating sequential or language-defined skills and motion commands into more concrete action sequences.
- The **Raw Action Policy** (often referred to as an Action Tokenizer in our diagrams, for A^r) is responsible for translating structured raw action specifications from the VLM (e.g., descriptions of mouse movements and key presses) into the discrete keyboard and mouse operations permissible by the Minecraft environment.

This MoP architecture allows DHA to leverage the strengths of different action abstractions and specialized low-level controllers, contributing to its versatility and effectiveness across a wide range of tasks.

B Details of Training Datasets

B.1 Detailed Description of Base Datasets for Minecraft World Knowledge

To address the disparity in Minecraft-specific world knowledge between open-source Vision-Language Models (VLMs) and leading proprietary models (e.g., the GPT series, Claude series), we have developed a comprehensive suite of base datasets. This initiative is driven by compelling experimental evidence that indicates deep, domain-specific world knowledge is crucial for achieving high Vision-Language-Action (VLA) performance in Minecraft (Li et al., 2025; Wang et al., 2024d). Our primary objective at this stage is to significantly improve the foundational Minecraft understanding and capabilities of open-source VLMs. The dataset, which primarily originates from and is further expanded based on resources such as JARVIS-VLA (Li et al., 2025), encompasses four main categories detailed below. The generation and curation of these components extensively utilize self-instruct methodologies and leverage the capabilities of powerful foundation models:

- **World Knowledge Question Answering (QA):** This component aims to instill a strong textual understanding of the Minecraft world. We initiated this by crawling a comprehensive textual corpus from the Minecraft Wiki. Utilizing this corpus, a set of approximately 202K questions was generated via a self-instruct methodology, employing large language model (LLM) such as GPT-3.5-turboOuyang et al. (2022). Subsequently, diverse and high-quality answers to these questions were generated by advanced LLMs, including GPT-4o (Hurst et al., 2024), Claude-3.5 Sonnet (Anthropic, 2025), and Gemini 1.5 Pro (Team et al., 2024a), operating in a zero-shot setting. The resulting 277K QA pairs were then subjected to an automated quality control process. In this step, a VLM, augmented with access to the relevant Minecraft Wiki texts, evaluated and filtered the answers to ensure accuracy and relevance.
- **Visual Question Answering (VQA) and Scene Captioning:** To enrich the model's visually grounded knowledge, we curated a diverse collection of approximately 35K images. These images were carefully selected from video recordings of human gameplay in Minecraft, ensuring a wide variety of scenes and contexts. GPT-4o (Hurst et al., 2024) was then employed to generate detailed captions for these images. Furthermore, GPT-4o engaged in a self-interrogation process (i.e., asking and answering questions) based on the visual content of each image to create a rich set of VQA pairs, yielding approximately 15K image-caption pairs and 20K VQA instances.
- **Visual Grounding:** For fine-grained object and region understanding within Minecraft scenes, we constructed a dedicated visual grounding dataset. An automated annotation algorithm was first employed to propose candidate grounding points and bounding boxes for salient objects or regions in approximately 600K Minecraft images. The accuracy of these initial (often noisy) proposals was then critically assessed and refined by GPT-4o (Hurst et al., 2024), which served as a visual critic. This process resulted in a high-fidelity dataset of 404K visual grounding annotations.

- **Reasoning Capabilities Enhancement:** To specifically bolster the model's capacity for in-game reasoning and planning, we adopted a two-pronged strategy. Firstly, we incorporated the Open-Thoughts dataset (Team, 2025a) (comprising 75K instances) to warm-up our pre-trained VLM, thereby fostering its latent reasoning abilities. Secondly, we designed a novel set of 5K unique Minecraft-centric tasks that necessitate multi-step logical deduction or planning to complete. The warmed-up VLM was then tasked with generating explicit reasoning pathways (chains of thought) for these tasks. A rule-based critic was subsequently employed, along with rejection sampling, to curate and select only high-quality and correct reasoning chains.

In total, this aggregated dataset, comprising approximately `[Total Number, e.g., 300k data points/pairs]`, was utilized for Supervised Fine-Tuning (SFT) of our Qwen2-VL based model (as detailed in Section 3.1). The objective of this SFT stage is to equip the model with robust foundational knowledge of Minecraft and a preliminary yet effective capacity for exploration and task completion within its environment.

B.2 Mixture of Abstracted Action Prediction Datasets

After processing the base instruction-following trajectories to derive datasets for each type of abstracted action (i.e., language skills A^s, grounding actions A^g, motion actions A^m, and raw actions A^r, we combine these into a comprehensive "mixture of abstracted action" dataset. This composite dataset is pivotal for training our Deep Hierarchical Agent (DHA) and is structured into two primary components, serving different stages of our training pipeline (see Section E).

1. Markovian Abstracted Action Data. This component of the dataset is designed primarily for the initial pre-training of the Vision-Language Model (VLM) to accurately generate various individual abstracted actions. Each instance in this dataset follows a Markovian structure, typically consisting of a triplet: $(\text{ins}_t, \text{obs}_t, A_t^*)$ where ins_t is the high-level language instruction, obs_t is the current visual observation, and A_t^* is the target abstracted action to be predicted at timestep t. A_t^* can be an instance of a language skill (A^s), a grounding action (A^g), a motion action (A^m), or a raw action sequence (A^r).

Crucially, to guide the VLM in generating the correct type and format of action, each training instance is accompanied by a specific system prompt. This prompt explicitly instructs the model on the desired format for its response and specifies the permissible action space for that particular instance. For example, a prompt might ask the VLM to generate a 'motion action' or a 'raw action sequence' based on the input.

When constructing this part of the dataset, we mix data instances corresponding to different abstracted action types. Specifically, for the non-language abstracted actions, we maintain a sampling ratio where instances requiring grounding actions (A^g), motion actions (A^m), and raw actions (A^r) are mixed in a ratio of approximately **1:4:16**, respectively. This ratio is designed to provide sufficient exposure to the more frequently occurring and granular raw actions while ensuring the model also learns higher-level motion and grounding primitives. Instances involving language skill (A^s) generation are also included in this mixture, though their sampling might follow a different distribution or be determined by their natural occurrence in the processed trajectories. This dataset component primarily facilitates Stage 1 of our training pipeline (Action Prediction Pre-training), enabling the VLM to master the semantics and generation of diverse, individual action abstractions.

2. Chain-of-Action (CoA) Structured Data. The second component of our dataset is formatted specifically for Chain-of-Action (CoA) learning. This data is crucial for the post-training phase (Stage 2) where the DHA learns to perform hierarchical reasoning by sequentially generating linked actions. Each instance in this dataset represents a complete abstracted action chain for a given observation obs_t and instruction ins_t, formulated as: $(\text{ins}_t, \text{obs}_t, (A_t^s, A_t^g, A_t^m, A_t^r))$ This structure explicitly provides the full hierarchy of actions, from the high-level language skill A_t^s down to the executable raw action A_t^r, with each intermediate action (A_t^g, A_t^m) conditioned on its predecessors in the chain, as detailed by our CoA formulation. Training on this CoA-structured data enables the DHA to learn the dependencies between different levels of action abstraction and to use higher-level actions as guiding "thoughts" for generating subsequent, more detailed actions. This phase is critical for imbuing the agent with its advanced reasoning and planning capabilities.

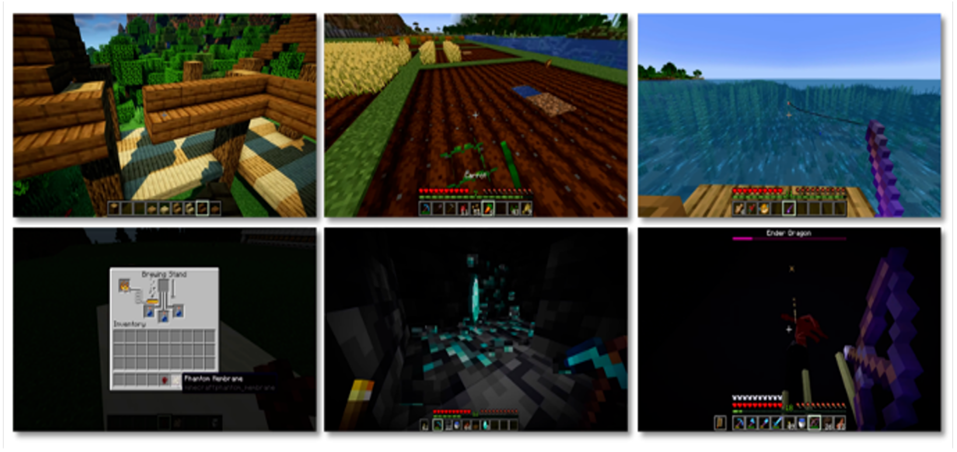


Figure C.1: Observation space of Minecraft agents.

By training on this two-part mixture dataset, DHA first learns to proficiently generate individual types of abstracted actions in a prompted manner (via the Markovian data) and then learns to strategically chain these actions together for complex, multi-step task execution (via the CoA-structured data).

B.3 Action Spaces and Action Pyramid

Motion. These consist of sequences of continuous actions that involve physical movement or changes in position. Examples include commands such as `move forward`, `turn right`, `jump` and compositions of short motion actions. Each motion action typically involves fine-grained control over the agent's movement dynamics, requiring precise, low-level policies that adjust parameters such as speed, direction, and acceleration. These actions are often dependent on the agent's current environment and may require real-time adjustments to account for obstacles or changes in the surroundings. The execution of motion actions necessitates a continuous flow of feedback and adjustments to ensure the agent's movement aligns with the intended goal.

Grounding. Grounding actions involve using visual coordinates to specify the location of objects instead of directly naming them. By combining a symbolic verb (e.g., `mine`) with object coordinates (e.g., `coordinate=[180, 320]`), grounding actions decouple the "what" from the "where." This enables a policy to execute a skill on any object by targeting its location.

Raw action. Low-level, direct control actions, such as keyboard and mouse events, that correspond to basic human-like interactions with the environment. These actions involve discrete or continuous inputs, such as moving the mouse or pressing keys, providing immediate control over the agent's behavior in real time.

C Simulator and Evaluation Tasks

C.1 Observation Space and Action Space

We adopt native Minecraft-rendered images as the observation space for all agents, as illustrated in Figure C.1. The action space is defined by human-like mouse and keyboard controls. Mouse displacement is discretized into 1800 bins. When the agent is in a GUI interface, these displacements correspond to cursor movements; otherwise, they adjust the camera view by changing pitch (Δy) and yaw (Δx). Mouse clicks are represented by dedicated tokens for left, right, and middle button presses. Keyboard inputs are encoded as unique tokens corresponding to alphabetic characters, numeric digits, and a predefined set of special keys (e.g., `Shift`, `Space`, `Escape`, `Enter`, etc.). A detailed mapping of mouse clicks and keyboard inputs is provided in Table 5.

Table 5: The action space we use in Minecraft.

Index	Action	Human Action	Description
1	Forward	key W	Move forward.
2	Back	key S	Move backward.
3	Left	key A	Strafe left.
4	Right	key D	Strafe right.
5	Jump	key Space	Jump. When swimming, keeps the player afloat.
6	Sneak	key left Shift	Slowly move in the current direction of movement.
7	Sprint	key left Ctrl	Move quickly in the direction of current motion.
8	Attack	left Button	Destroy blocks (hold down); Attack entity (click once).
9	Use	right Button	Interact with the block.
10	hotbar.[1-9]	keys 1 - 9	Selects the appropriate hotbar item.
11	Yaw	move Mouse X	Turning; aiming; camera movement. Ranging from -180 to +180.
12	Pitch	move Mouse Y	Turning; aiming; camera movement. Ranging from -180 to +180.
13	Equip	-	Equip the item in main hand from inventory.
14	Craft	-	Execute a crafting recipe to obtain new item.
15	Smelt	-	Execute a smelting recipe to obtain new item.

C.2 Evaluation Tasks

We evaluate the performance of our method on a diverse suite of tasks within the Minecraft environment, categorized into five main groups: mining blocks, crafting items, killing entities, smelting items, and interacting with blocks. To thoroughly assess generalization capabilities, we have designed an extensive benchmark comprising over 800 distinct tasks distributed across these groups, with each group containing at least 100 different task variations.

Task success is determined automatically by parsing information returned by the environment upon specific events. For example, if the agent is given a task with the natural language description such as "use an iron pickaxe to mine a diamond ore," successful completion is logged if the environment registers events corresponding to the criteria for that task (e.g., an event like `mine_block:diamond_ore` occurring, potentially with conditions on tool usage like `use_item:iron_pickaxe` if specified in the success criteria for that particular task variant). It is important to note that this backend success-checking information is utilized exclusively for evaluation and is not available to the agent as part of its observation space, which consists solely of visual images.

Below, we describe the general setup for one of the major task categories, "mine blocks," as an illustration of our task design methodology.

Task Setup for *Mine Blocks*. Tasks within the "Mine Blocks" category require the agent to locate and mine a specific type of block within the Minecraft world. Each individual task, identified by a unique name (e.g., `mine_block:oak_log`, `mine_block:iron_ore`), is procedurally generated with controlled initial conditions to ensure reproducibility and varied challenges. The setup for each task variant is defined by the following parameters, as exemplified by the JSON structure you've provided:

- **Target Block**: The specific Minecraft item ID of the block to be mined (e.g., `oak_log`, `diamond_ore`).
- **Initial Conditions** (`seeds`): Each task variant is associated with one or more predefined world `seed` values and specific agent starting `position` coordinates (x, y, z). This allows for multiple scenarios for mining the same block type, varying the environmental context and agent starting location. For example, the task `mine_block:diamond_ore` might have multiple seed/position combinations for increased diversity.
- **Permitted/Required Tools** (`tool`): A list of item IDs for tools that are either required or permitted for successfully mining the target block. This can range from a single specific tool (e.g., `["diamond_pickaxe"]` for `obsidian`) to a list of acceptable tools of varying materials (e.g., `["stone_pickaxe", "iron_pickaxe", "diamond_pickaxe"]` for `iron_ore`). Some tasks may even specify tools with particular enchantments,

such as `wooden_shovel:{Enchantments:[{id:silk_touch,lvl:1}]}` for mining `grass_block` to obtain the block itself rather than dirt.

For instance, a task to mine iron ore might be defined as `mine_block:iron_ore`, with an initial setup placing the agent at coordinates like `[194, 61, 897]` within a world generated by seed `2025`. Successful completion would require the agent to mine an iron ore block, permissibly using a stone, iron, or diamond pickaxe. The automatic evaluation would check for the corresponding `mine_block:iron_ore` event from the environment, under valid tool conditions if specified.

Task Setup for *Craft Items*. The "Craft Items" task category challenges the agent to create specific items using the in-game crafting system. Each task, such as `craft_item:dispenser` or `craft_item:black_dye`, sets a specific `goal` item for the agent to produce. The setup for these tasks is defined by several key parameters derived from your JSON data:

- **Target Item** (`goal`): The unique item ID of the object the agent must craft (e.g., `dispenser`, `golden_leggings`).
- **Initial Inventory** (`init_inventory`): A predefined set of items and materials provided to the agent at the start of the task. Each entry specifies the item `type` (e.g., `redstone`, `cobblestone`, `wither_rose`), the required `quantity` (which can be exact or a minimum, e.g., `">=7"`), and often a `slot` for initial placement (e.g., `"random"`). This ensures the agent has the necessary, and sometimes exact, raw materials for the recipe.
- **Crafting Table Requirement** (`need_crafting_table`): A boolean value (`true` or `false`) indicating whether the target item's recipe necessitates the use of a crafting table, or if it can be crafted directly in the player's 2x2 inventory grid.
- **Initial Conditions** (`seeds`): Similar to other task categories, each crafting task variant can be associated with specific world `seed`(s) and agent starting `position` coordinates, although the initial inventory is typically the more critical factor for task success in this category.

For example, the task `craft_item:dispenser` requires the agent to craft a dispenser. The agent would start with an initial inventory containing at least seven cobblestone blocks, one redstone dust, and one bow, and would need to use a crafting table. In contrast, a task like `craft_item:black_dye` can be completed without a crafting table, starting with at least one wither rose. Task success is determined automatically by checking if the agent's inventory contains the specified `goal` item after its actions, based on information logged by the environment.

D CASE STUDY

We list a few cases of different inference modes in Figure D.2. We present several observations from rollout examples for the tasks `kill a sheep` in eager mode, `craft paper` in greedy mode, and `chop trees` in direct mode.

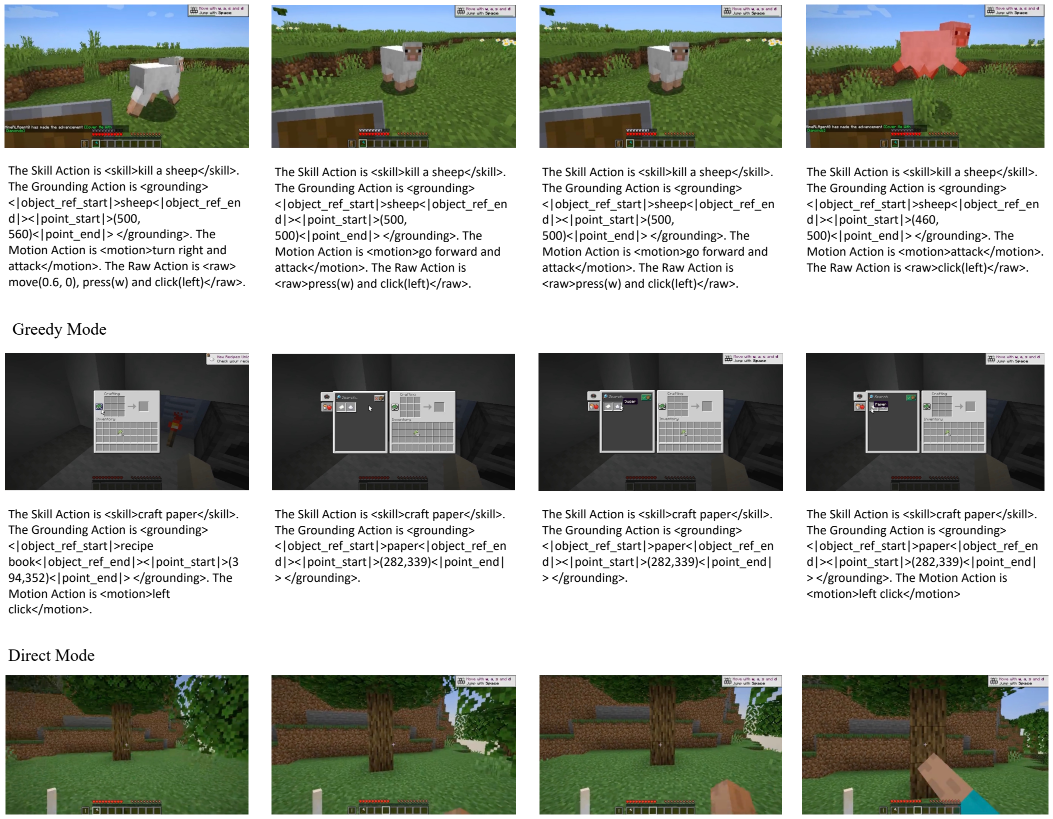
E TRAINING RECIPE

Our Deep Hierarchical Agent (DHA) is trained via a multi-stage pipeline designed to progressively build its capabilities, from foundational knowledge to sophisticated chained-action generation. Key hyperparameters for these three stages are summarized in Table 6.

The training proceeds as follows:

Stage 0: Basic Knowledge Finetuning. First, we imbue the base Vision-Language Model (VLM) with fundamental Minecraft domain knowledge. This involves supervised finetuning of the LLM backbone using 3 billion tokens from our curated basic knowledge datasets.

Stage 1: Mixture of Abstracted Action Prediction Pre-training. Next, all components of the VLM are pretrained on a significantly larger dataset (11 million images) comprising a mixture of abstracted actions. This stage teaches the model the semantics of diverse action types (e.g., language actions, grounding actions, motion actions, and raw actions) and how to associate them with instructions and observations, without yet enforcing explicit sequential dependencies as in CoA.



Eager Mode

The Skill Action is <skill>kill a sheep</skill>. The Grounding Action is <grounding><|object_ref_start|>sheep<|object_ref_end|><|point_start|>(500, 560)<|point_end|> </grounding>. The Motion Action is <motion>turn right and attack</motion>. The Raw Action is <raw> move(0.6, 0), press(w) and click(left)</raw>.

The Skill Action is <skill>kill a sheep</skill>. The Grounding Action is <grounding><|object_ref_start|>sheep<|object_ref_end|><|point_start|>(500, 500)<|point_end|> </grounding>. The Motion Action is <motion>go forward and attack</motion>. The Raw Action is <raw>press(w) and click(left)</raw>.

The Skill Action is <skill>kill a sheep</skill>. The Grounding Action is <grounding><|object_ref_start|>sheep<|object_ref_end|><|point_start|>(500, 500)<|point_end|> </grounding>. The Motion Action is <motion>go forward and attack</motion>. The Raw Action is <raw>press(w) and click(left)</raw>.

The Skill Action is <skill>kill a sheep</skill>. The Grounding Action is <grounding><|object_ref_start|>sheep<|object_ref_end|><|point_start|>(460, 500)<|point_end|> </grounding>. The Motion Action is <motion>attack</motion>. The Raw Action is <raw>click(left)</raw>.

Greedy Mode

The Skill Action is <skill>craft paper</skill>. The Grounding Action is <grounding><|object_ref_start|>recipe book<|object_ref_end|><|point_start|>(394,352)<|point_end|> </grounding>. The Motion Action is <motion>left click</motion>.

The Skill Action is <skill>craft paper</skill>. The Grounding Action is <grounding><|object_ref_start|>paper<|object_ref_end|><|point_start|>(282,339)<|point_end|> </grounding>.

The Skill Action is <skill>craft paper</skill>. The Grounding Action is <grounding><|object_ref_start|>paper<|object_ref_end|><|point_start|>(282,339)<|point_end|> </grounding>.

The Skill Action is <skill>craft paper</skill>. The Grounding Action is <grounding><|object_ref_start|>paper<|object_ref_end|><|point_start|>(282,339)<|point_end|> </grounding>. The Motion Action is <motion>left click</motion>

Direct Mode

The Raw Action is <raw>press(w) and click(left)</raw>.

The Raw Action is <raw>move(-1, 0) and click(left)</raw>.

The Raw Action is <raw>press(w) and click(left)</raw>.

The Raw Action is <raw>click(left)</raw>.

Figure D.2: Case study of inference modes.

Table 6: Training setup and hyperparameters for the three training stages of our DHA.

Stages	Stage 0	Stage 1	Stage 2
Training budget (tokens)	3B	100B	1B
Training Images	550K	11M	319K
Trainable Components	LLM backbone	all	all
Batch Sizes	128	512	128
LR warmup steps	100	200	100
Maximum LR	5.0×10^{-5}	5.0×10^{-6}	3.0×10^{-6}
Minimum LR	5.0×10^{-6}	0	0

Stage 2: Chain-of-Action Post-Training. Finally, the entire DHA is post-trained using 1 billion tokens of data formatted specifically for CoA learning. This teaches the model to generate actions in a sequentially dependent, hierarchical manner, where higher-level actions guide lower-level ones, refining its reasoning and planning capabilities. This three-stage pipeline ensures our DHA first acquires essential domain knowledge, then learns a broad repertoire of actions, and ultimately masters the sophisticated Chain-of-Action reasoning process for complex task execution.