

OPENHA: A SERIES OF OPEN-SOURCE HIERARCHICAL AGENTIC MODELS IN MINECRAFT

Anonymous authors

Paper under double-blind review

ABSTRACT

A critical challenge in developing capable AI agents is defining their “action space”—the set of possible actions they can take. These spaces can range widely, from generating code and using language skills to operating on latent representations or raw joystick controls. Through a large-scale study in Minecraft, we discovered a major dilemma: no single action space is universally best. The most effective action space is highly task-dependent, which complicates the goal of building one generalist agent that can handle everything. To solve this, we introduce Chain-of-Action (CoA), a novel framework that unifies high-level abstracted actions and low-level control actions within a single model. With CoA, an abstract goal is not just a final command; instead, it serves as an intermediate reasoning step that guides the model to generate the precise, executable actions needed to complete the task. Furthermore, we show that an “All-in-One” agent, trained on a diverse mix of action spaces using CoA, learns a more generalizable policy. This unified agent achieves a new state-of-the-art, outperforming strong, specialized baselines. To support the research community, we are releasing the OpenHA (Open Hierarchical Agents) suite, which includes our benchmark of over 800 tasks, curated datasets, source code, and all model checkpoints at: <https://anonymous.4open.science/anonymize/OpenHA-ACFE>.

1 INTRODUCTION

In their early designs, AI agents were primarily built as modular systems that utilized pre-trained models, including Large Language Models (LLMs) (Touvron et al., 2023; OpenAI, 2023a; Yang et al., 2025; Anthropic, 2025) and Vision-Language Models (VLMs) (Achiam et al., 2023; Wang et al., 2024b; Team et al., 2023; Guo et al., 2025) and combined with external modules—such as memory, planning, and tool use—to generate actions within an environment (Weng, 2023; Yang et al., 2023; Cheng et al., 2024). However, this modular approach leads to a significant limitation: the agent’s performance is restricted by the capabilities of the pre-trained model (Wang et al., 2024a; Yu et al., 2025) and is unable to learn and improve from its interactions with the environment (Li et al., 2025b).

To address this constraint, recent advancements have turned toward the development of end-to-end pre-trained agentic models that learn directly from interaction data (Qin et al., 2025; openai, 2025; Brohan et al., 2023b; Driess et al., 2023). These models embed the capacity for action generation within the agent itself. A key distinction among different agentic models lies in how they represent the action space of agents, which ranges from language skill (Driess et al., 2023), code (Wang et al., 2024a; Liang et al., 2022), MCP (Anthropic, 2024), GUI actions (openai, 2025), and raw actions (Brohan et al., 2023b). Based on the hierarchy of actions, these models can be broadly categorized into two types. The first type, Vision-Language-Action (VLA) models (Kim et al., 2024), directly tokenize low-level actions—such as keyboard inputs or robotic movements—into sequences that are compatible with the vocabulary of language models. This enables the model to predict and generate low-level actions directly from the interaction context. However, a significant challenge arises from the semantic gap between the high-level abstractions of language and the continuous, detailed nature of low-level actions (Wang et al., 2024d).

To overcome this issue, a second class of agent models adopts a hierarchical architecture and defines an abstracted action space for agentic models (Belkhale et al., 2024b; Driess et al., 2023; Bjorck et al., 2025; Zhou et al., 2024). In these models, a high-level language model predicts an abstracted

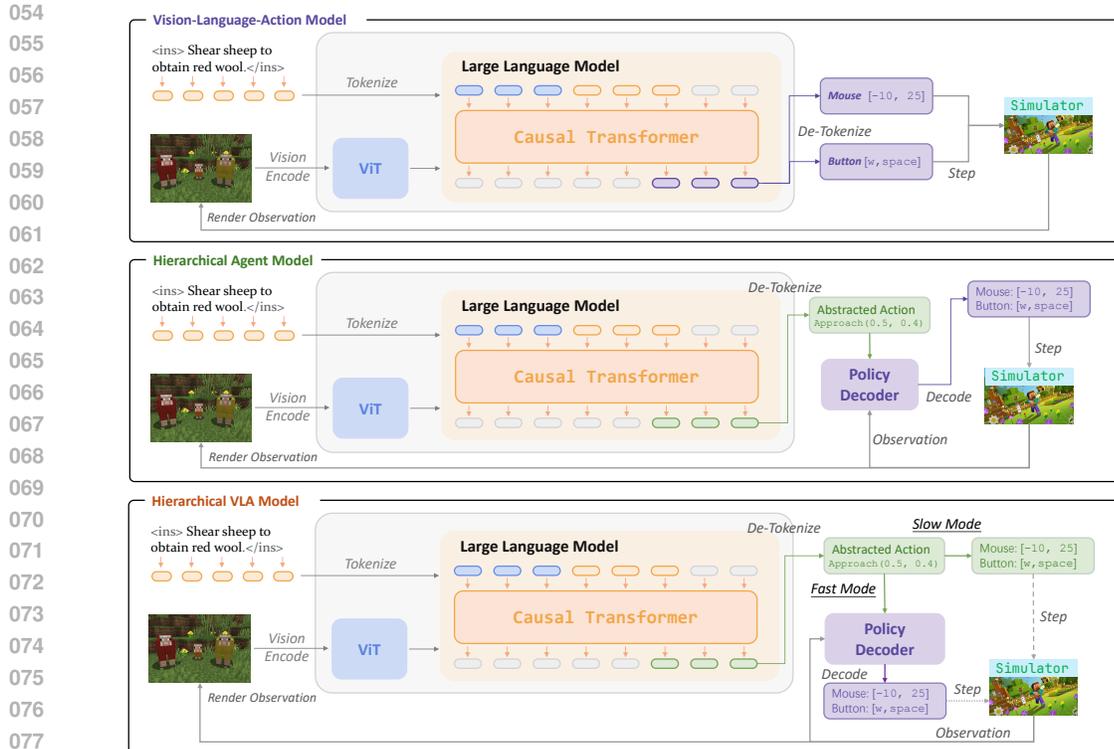


Figure 1: Comparisons between end-to-end vision-language-action (VLA) models and hierarchical agent (HA) models. The core difference between HA and the VLA model is that HA uses a policy-based action tokenizer to predict actions, while the vision-language-action model uses a text tokenizer to decode actions. We further propose integrating the abstracted actions of the hierarchical agent into the training and inference processes of VLA, resulting in the Hierarchical VLA model, which has both fast and slow inference modes, using a policy-based action tokenizer and a text tokenizer to decode environmental actions, respectively.

action, which is then decoded into a sequence of low-level actions by a learned action tokenizer or controller (Wang et al., 2024d; Cai et al., 2024b). While this hierarchical approach separates reasoning and control, a key question arises: *how can abstracted action spaces be effectively defined and formulated from the continuous stream of low-level actions?* Despite the widespread adoption of various action tokenizers (Zhong et al., 2025a), a clear, standardized evaluation remains lacking. Most action tokenizers are tested on domain-specific benchmarks and proprietary datasets, making direct and fair comparisons difficult (Belkhale et al., 2024a; Gu et al., 2023; Kim et al., 2024).

We present a unified and fair evaluation to determine which abstracted action spaces are most effective. This evaluation includes a large-scale comparison of abstracted action spaces across 800 tasks, based on the open-ended environment of Minecraft (Fan et al., 2022). Our experiments reveal that the effectiveness of an action tokenizer is highly task-dependent, with no single action space demonstrating universal superiority.

Traditional abstracted actions are limited and only applied to hierarchical agentic models, which include high-level vision-language models and learning-based decoders or action tokenizers. In these models, joint optimization across different levels is challenging (Driess et al., 2025). We further explored whether these abstracted actions are also beneficial for the end-to-end VLA model. We introduce the Chain-of-action (CoA) framework, which integrates the strengths of both hierarchical agentic models and VLA models. In CoA, action generation is structured as a two-step, auto-regressive process: first, the model generates a high-level abstracted action, which serves as an intermediate “thinking”; then, conditioned on this thought, the model generates the low-level action auto-regressively.

In addition, inspired by the discovery that different abstracted actions excel in distinct task domains, we propose an All-in-One training strategy. The key insight is that, by grounding the model’s final output in low-level actions, CoA enables a unified prediction structure across all action spaces. This

allows us to train a single agent on a composite dataset containing multiple action representations. The resulting agent can master a diverse range of actions, outperforming specialist models trained on a single type of action.

The main contributions of this work are fourfold: 1) We provide a large-scale and systematic evaluation to demonstrate that the optimal abstracted action space is task-dependent. 2) We introduce the Chain-of-Action framework to synergize high-level reasoning and low-level control by using abstracted actions as intermediate thinking. This framework bridges the gaps and offers superior performance. 3) We demonstrate that training a single agentic model on mixed datasets of diverse action spaces enhances its decision-making ability to generalize across tasks. 4) We release a comprehensive suite including trained checkpoints, code, and datasets for various agentic models to support further research in action spaces and generalization.

2 ABSTRACTED ACTIONS IN AGENTIC MODELS

2.1 HIERARCHICAL AGENT STRUCTURE

Hierarchical agents model the agent’s decision-making process through a hierarchical policy decomposition. This process consists of two levels: first, a high-level pretrained LLM-based agentic model generates an abstracted action based on the task instruction and the current observation; second, a low-level policy acts as the **action tokenizer** and uses this abstracted action as guidance to produce a primitive, executable action in the environment. This two-level procedure can be expressed as:

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

Here, *ins* denotes the human-provided textual instruction, and *obs* $\in \mathbb{R}^{H \times W \times 3}$ represents the current visual observation. The variable $A \in \mathcal{A}_{abs}$ signifies an **abstracted action** sampled from a high-level action space \mathcal{A}_{abs} , generated by the high-level policy π_{LM} . Subsequently, $a \in \mathcal{A}_{env}$ represents a **primitive environmental action** from the executable action space \mathcal{A}_{env} , generated by the low-level policy π_{policy} as the action tokenizer (Zhong et al., 2025a). In domains like computer interaction, these primitive actions correspond to raw inputs such as keyboard presses or mouse movements.

The specific formulation of the abstracted action A is a critical design choice and varies significantly across different agent architectures. It can range from symbolic, language-based skills (e.g., “pick up the apple”) (Driess et al., 2023), to continuous, grounded interaction trajectories (Gu et al., 2023), textual descriptions of motion (e.g., “go forward”) (Belkhale et al., 2024b), discrete latent codes learned by encode-decoder models (Wang et al., 2024d; Bjorck et al., 2025), **or even future visual tokens** Zhou et al. (2024); Zhao et al. (2025).

In typical implementations, the high-level auto-regressive models π_{LM} are instantiated as a large-scale Vision-Language Model (VLM) (Wang et al., 2024b; Grattafiori et al., 2024; OpenAI, 2023b), leveraging its powerful reasoning capabilities to process multimodal inputs and generate the abstracted actions A . In contrast, the low-level policy π_{policy} is often a more lightweight, specialized model, such as a Transformer with less than 100M parameters (Dai et al., 2019; Vaswani et al., 2017), tasked with translating the plan A into a sequence of environment-specific primitive actions a (Lifshitz et al., 2024; Cai et al., 2024b).

This hierarchical formulation is general enough to subsume many recent Vision-Language-Action (VLA) models (Chi et al., 2023; Team et al., 2024; Brohan et al., 2022; 2023a; Zheng et al., 2024; Zhang et al., 2024; Zhong et al., 2025b; Li et al., 2024; Zhang et al., 2025; Zhu et al., 2025; Zhou et al., 2025; Chen et al., 2025). These non-hierarchical or “flat” VLA models can be viewed as a special case of our framework where the hierarchy collapses and the policy π_{policy} is the language tokenizer itself (Bai et al., 2023). In this scenario, the high-level policy is trained to directly output the specification of the primitive action, effectively making the abstracted action space equivalent to the primitive one ($A \equiv a$). Consequently, the low-level policy π_{policy} reduces to a trivial identity function, as the planning and execution steps are merged into a single end-to-end network.

2.2 ACTION SPACES IN HIERARCHICAL AGENTS

The efficacy of a hierarchical agent is intrinsically tied to the design of its action space, A . This space defines the vocabulary of behaviors the agent can command (Zhong et al., 2025a). In this work, we

Table 1: Definitions and examples of the different action spaces in agents.

Symbol	Definition	Action Types	Example
a	Env Action	24	{"camera": [-1, -9], "ESC": 0, "back": 0, "drop": 0, "forward": 0, "hotbar.1-9": 0, "inventory": 0, "jump": 0, "left": 0, "right": 0, "sneak": 0, "sprint": 0, "swapHands": 0, "attack": 0, "use": 0, "pickItem": 0}
RA	Raw Action	35	"<reserved_token_11><reserved_token_7><reserved_token_9><reserved_token_2>"
TA	Text Action	3	"Action: move(0, 0) and press(left.control, w, a) and click(left)"
MA	Motion Action	-	"Action: move(4, -1) and press(left.control, w)"
GA	Grounding Action	8	"Motion: Go forward, Turn left."
LA	Latent Action	15000	"Grounding: Mine(object='oak_log', position=[100, 200])"
			"Grounding: Approach(object='sheep', position=[200, 300])"
			"<reserved_token_21>"

investigate a spectrum of action spaces, ranging from low-level environmental controls to various high-level abstractions. A summary and examples of these spaces are provided in Table 1. Examples can be found in Appendix A.

Primitive Environmental Actions (a). At the most fundamental level is the primitive action space dictated by the environment simulator. In our context, this corresponds to low-level keyboard and mouse operations. To be processed by a language-based model, these raw actions must be tokenized using reserved tokens within the language model RA like OpenVLA (Kim et al., 2024; Brohan et al., 2023a) or textual serialization as text action TA =mouseMove(dx=4, dy=-1) (Seed, 2025).

Language Skills SA . A natural choice for high-level models based on VLMs is to use natural language skills as the abstracted action space. These skills typically describe a complete, goal-oriented behavior, often with a verb-object structure like “chop down trees” or “open the chest”. Executing such a skill requires a sophisticated, language-conditioned low-level policy that interprets the command and, conditioned on the current visual observation, decodes it into a sequence of primitive actions to achieve the goal (Wang et al., 2023; Driess et al., 2023).

Motion Primitives MA . Occupying an intermediate level of abstraction, motion primitives represent temporally extended but object-agnostic movements. For instance, a continuous sequence of “press W and Left.Shift” can be abstracted into a single “sprint forward” command. Similarly, a series of mouse movements can be categorized as “turn left” in an egocentric frame. These actions are more abstract than primitive environmental actions but are lower-level than language skills, as they do not specify an object of interaction (Lifshitz et al., 2024; Belkhale et al., 2024b).

Grounding Actions GA . While pure language skills face scalability challenges, as the low-level policy must learn to interpret possible object for a given skill. Grounding actions address this by creating a structured, multimodal representation. They retain the symbolic verb from language skills (e.g., mine) but replace the object’s name with its grounded spatial coordinates in the visual observation (e.g., coordinate=[100, 200]). This decouples the “what” from the “where,” enabling a single, grounding-conditioned policy to execute a skill on any object by simply targeting its location, significantly improving generalization (Zhong et al., 2025b; Cai et al., 2024a; Lee et al., 2025).

Latent Actions LA . The aforementioned action spaces rely on human-defined heuristics and feature engineering. In contrast, latent actions are learned directly from data in a self-supervised manner. This is typically achieved using an encoder-decoder framework, where an encoder compresses sequences of primitive actions into a continuous latent embedding. To make these embeddings compatible with auto-regressive generation, a Vector-Quantized Variational Autoencoder (VQ-VAE) is often employed to discretize the continuous latent space into a finite codebook of tokens (Van Den Oord et al., 2017). The high-level policy then generates these learned latent tokens as its abstracted actions (Deng et al., 2025; Yuan et al., 2024; Shafiq et al., 2022).

3 SYNERGIZING ABSTRACTED ACTIONS AND LOW-LEVEL CONTROL

3.1 TAKING ABSTRACTED ACTIONS AS THOUGHTS

While abstracted actions simplify the decision-making for VLM-based agents, they are usually tied to two-stage hierarchical architectures. Such setups always hinder end-to-end training (Wang et al., 2024c; Driess et al., 2023). The low-level policy is learned in isolation, and often caps overall performance. To circumvent these limitations, we introduce the **Chain of Action (CoA)**, an auto-

regressive formulation that reframes action generation as a single, sequential process. The core idea is to treat the high-level abstracted action A not as a command for a separate module, but as an intermediate reasoning step that guides the subsequent prediction of the low-level environmental action a .

This approach leverages the powerful sequence modeling capabilities of modern auto-regressive transformers (Zawalski et al., 2024; Wei et al., 2022; Vaswani et al., 2017). Instead of predicting the final action a directly from the observation obs and instruction ins , the model first generates the abstracted action A and then, conditioned on this self-generated context, predicts the primitive action a . We formalize this by modeling the joint probability of the action chain (A, a) . Given an instruction ins and an observation obs , the likelihood of generating the sequence is factorized using the chain rule of probability:

$$P(A, a \mid ins, obs) = \underbrace{P(a \mid ins, obs, A)}_{\text{Action Generation}} \cdot \underbrace{P(A \mid ins, obs)}_{\text{"Thought" Generation}} \quad (2)$$

Here, a single auto-regressive language model, π_θ , models the entire generation process. The entire model π_θ is trained end-to-end by maximizing the log-likelihood of this joint probability over a dataset of expert trajectories.

To illustrate, consider the task "CHOP DOWN A TREE." A standard VLA model must implicitly reason about the tree's location, its distance, and the required navigation, all while directly outputting a complex sequence of low-level actions (e.g., `press W`, `turn left`, `left click mouse`). This implicit reasoning is difficult to learn. In contrast, the CoA agent explicitly decomposes this problem:

1. **Thought Generation:** The model first generates a grounded action as a thought, such as `GA: Attack(object=oak_log, coordinate=[100, 200])`.
2. **Action Generation:** Conditioned on this explicit target, the subsequent prediction of low-level actions becomes a simpler, more grounded task. The model can observe that the target at `[100, 200]` is far away and to the left, making the generation of `a: press W` a direct consequence of the explicit reasoning step.

In this way, the CoA formulation grounds complex reasoning in an explicit, intermediate representation, breaking down a difficult prediction problem into a more manageable, sequential one and seamlessly unifying hierarchical reasoning within a single end-to-end model.

3.2 HIERARCHICAL AGENTIC MODELS WITH FAST OR SLOW INFERENCE METHODS

The Chain-of-Action (CoA) formulation is not only a training paradigm but also an architectural principle that affords significant flexibility at inference time. As illustrated in Figure 1, our model can operate in two distinct inference modes, allowing for a dynamic trade-off between reasoning depth and computational efficiency.

Decoupled Inference Mode (Fast). This mode emulates the classic hierarchical agent architecture. The primary large language model, which we refer to as the high-level policy π_{LM} , is queried only to generate the abstracted action A . The subsequent task of decoding it into a sequence of primitive actions $\{a^1, \dots, a^k\}$ is offloaded to a separate, lightweight low-level policy π_{policy} as a de-tokenizer. This approach significantly reduces the inference load on the large VLM, as it is invoked only once per high-level abstracted action, while the more efficient π_{policy} can interact with the environment for multiple steps. However, this mode has two primary drawbacks. First, it typically requires a multi-stage training process: the low-level policy π_{policy} must be trained first to serve as a decoder, followed by the training of the high-level policy π_{LM} . Second, the overall performance is potentially bottlenecked by the capability of the smaller, specialized low-level policy.

Unified Autoregressive Mode (Slow). In contrast, this mode fully leverages the end-to-end nature of the CoA framework. The single, unified model π_θ is responsible for generating the entire action sequence autoregressively, first producing the abstracted action A , and then immediately generating the corresponding low-level action a , as described in Equation 2. This approach ensures that the full reasoning capacity of the large model is applied to both high-level thinking and low-level control, eliminating any potential performance bottlenecks from a separate decoder. The primary trade-off is the increased computational cost at inference time, as the VLM must generate a longer sequence of tokens for each environmental action, resulting in slower decision-making.

A crucial advantage of our framework is that these two inference modes do not require separately trained models. A single agent can be endowed with both capabilities through a unified, end-to-end training process on a mixed dataset. During training, we provide the model with examples formatted for both objectives. The final output format is controlled via a specific system prompt provided as input to the model. This methodology effectively merges the traditional hierarchical agentic models and the end-to-end VLA paradigm into a single, versatile architecture. It empowers a practitioner to dynamically select the most suitable inference cost at runtime—choosing the fast, decoupled mode for simpler tasks or resource-constrained environments, and the slower, more powerful unified mode for tasks demanding complex reasoning.

3.3 OPENHA: TRAINING ONE AGENT TO MASTER DIVERSE ABSTRACTED ACTIONS

The traditional agentic model is usually trained on a fixed, predefined action space. We propose the **All-in-One** training strategy, which aims to produce a single agentic model proficient in multiple action spaces. A naive approach would be to simply train a model on a heterogeneous dataset of trajectories, $\mathcal{D} = \bigcup_k \mathcal{D}_k$, where each sub-dataset \mathcal{D}_k contains trajectories annotated with a different high-level action space $\mathcal{A}^{(k)}$ (e.g., Grounding Actions GA , Motion Actions MA , etc.). However, such a strategy faces a critical challenge: the different action spaces are fundamentally disparate. Without a shared basis, the model may treat them as isolated prediction domains, failing to generalize or transfer knowledge between them.

Our CoA framework provides a principled solution to this challenge. By adopting the CoA data format for all trajectories— $(ins, \{(o_t, A_t, a_t)\}_{t=1}^T)$ —we ensure that every high-level abstracted action a , regardless of its origin space, is ultimately grounded in the same primitive environmental action space \mathcal{A} . The primitive action a_t acts as a **common currency**, creating a shared semantic ground that connects all high-level abstractions.

This common grounding enables the model to learn the underlying relationships and functional equivalences between different action representations. For example, the model can infer that a high-level thinking expressed as a Grounding Action like $GA: \text{Approach}(\text{object}=\text{sheep})$ and one expressed as a Motion Action like $MA: \text{Go forward}$ can be contextually similar if they both result in the same primitive action sequence $a: \text{press}(\text{w})$. The training objective remains identical to that defined in Equation 2, maximized over the full, heterogeneous dataset \mathcal{D} .

We hypothesize that this All-in-One training strategy will yield two significant benefits. First, it will produce a highly versatile agent capable of understanding and generating multiple types of action abstractions. Second, and more importantly, by learning a more holistic and interconnected representation of behavior, the All-in-One agent will develop a more robust and generalizable internal policy, ultimately outperforming any specialist agent trained on only a single form of action abstraction.

4 EXPERIMENTS AND INSIGHTS

In this section, we present a comprehensive suite of experiments designed to empirically validate our proposed frameworks and investigate key questions surrounding action representation for open-ended agents. We begin by detailing our experimental setup, including the training protocols for our models, the large-scale benchmark constructed for evaluation, and the metrics used to measure performance. Our analysis is then structured to systematically answer three central research questions:

Q1: Within a traditional Hierarchical Agentic model (HA) paradigm, what type of abstracted action space is most effective in a complex, open-ended environment like Minecraft?

Q2: Can incorporating these abstractions as intermediate reasoning steps within a unified VLA model improve its overall performance?

Q3: Does training a single agent on a heterogeneous mixture of action spaces enable it to learn a more generalizable and effective decision-making policy?

4.1 EXPERIMENTAL SETUP

Simulator and Evaluation Benchmarks. We employ Minecraft (Version 1.16.5) as our primary testbed (Guss et al., 2019). Previous work on Minecraft agents has often been evaluated on a limited number of canonical tasks (Baker et al., 2022; Lifshitz et al., 2024; Cai et al., 2024b). Although MCU (Lin et al., 2023) and Minedojo (Fan et al., 2022) both claim to have around 3,000 tasks, these are usually automatically generated by

Table 2: Evaluation results of Minecraft agents on over 800+ tasks. For each task category, we report the average steps to finish tasks, the average success rates across mini sets, and the Average Success Rate across all tasks in that category ($ASR \pm$ 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.

Model	Action Tokenizer	Inference	Embodied Tasks			GUI Tasks			Combat Tasks		
			Steps	ASR (Mini)	ASR (All)	Steps	ASR (Mini)	ASR (All)	Steps	ASR (Mini)	ASR (All)
Previous Methods											
VPT	-	End-to-end	377	10.1 \pm 3.6	6.0 \pm 11.4	398	0.7 \pm 0.1	0.8 \pm 3.3	396	3.6 \pm 7.7	3.6 \pm 7.7
STEVE-1	-	End-to-end	384	8.4 \pm 3.0	8.0 \pm 17.0	391	0.0	3.2 \pm 8.4	395	4.9 \pm 1.8	3.9 \pm 12.0
ROCKET-1	-	Workflow	392	19.2 \pm 6.1	18.9 \pm 24.3	N/A	0.0	0.0	320	29.8\pm9.0	27.9\pm29.3
JARVIS-VLA	-	End-to-end	305	31.0 \pm 12.7	30.0 \pm 35.4	339	25.3\pm5.7	25.1\pm23.9	352	18.3 \pm 5.2	18.5 \pm 22.7
VLM-based Hierarchical Agents											
LatentHA	Latent	Hierarchical	363	27.3 \pm 37.4	24.4 \pm 31.1	393	3.5 \pm 8.7	3.0 \pm 7.5	371	8.2 \pm 15.6	8.5 \pm 17.9
MotionHA	Motion	Hierarchical	336	31.6 \pm 10.1	27.4 \pm 35.2	N/A	0.0	0.0	392	9.1 \pm 3.9	4.3 \pm 10.8
GroundingHA	Grounding	Hierarchical	290	39.7\pm13.7	37.1\pm38.5	380	3.7 \pm 2.3	6.7 \pm 10.8	346	28.2 \pm 6.2	26.5 \pm 23.4
SkillHA	Skill	Hierarchical	365	13.8 \pm 7.7	11.3 \pm 14.5	397	3.4 \pm 0.8	6.3 \pm 9.2	393	3.4 \pm 0.8	6.5 \pm 9.3
Vision-Language-Action Models											
TextVLA	Text	End-to-end	321	23.9 \pm 8.9	27.0 \pm 17.0	291	14.0 \pm 4.1	25.8 \pm 14.3	317	27.1 \pm 11.8	10.0 \pm 6.1
OpenHA	Text	End-to-end	287	37.0\pm15.9	30.1\pm13.9	314	33.3\pm13.3	32.5\pm9.2	316	40.0\pm19.6	31.9\pm13.7

LLMs, and cannot guarantee a fair and stable success rate for the agents. To enable a more comprehensive and fair comparison of different action spaces, we construct a large-scale benchmark comprising above 800 distinct tasks. All tasks are designed and verified by hand Appendix F. We categorize these tasks into three groups based on the primary capabilities required: 1) **Embodied** tasks requiring navigation and physical interaction within the 3D world (e.g., finding and chopping down a specific type of tree), 2) **GUI** tasks that involve complex interactions with graphical user interfaces, such as crafting items at a crafting table or smelting ores in a furnace, 3) **Combat** tasks that require the agent to engage with hostile mobs, focusing on survival and combat skills. To rigorously test for generalization, we ensure that all evaluation environments are out-of-distribution with respect to the training data. This is achieved by sampling novel initial world seeds and spawn locations for each task evaluation from Minecraft’s vast procedural generation space. Each agent is evaluated on every task in our benchmark for a minimum of three independent runs, each with a different world seed. To obtain more statistically robust results on a representative sample, we selected 10 tasks of varying difficulty (easy, medium, and hard) from each of the three groups for a more intensive evaluation of over 10 runs each. Our primary metrics are **Success Rate** and **Steps to Completion**. For our final analysis, we report the average Success Rate and average number of steps, aggregated across all tasks within each of the three benchmark groups.

Dataset Curation. The training data for all agents are derived from the same expert dataset (Baker et al., 2022) to ensure experimental fairness. On top of the raw data, we developed a **programmatically labeling pipeline** to facilitate the training of agents with high-level action spaces. This process yields three distinct types of trajectory datasets used in our experiments: (1) **High-Level Action Data (D_A)**: Trajectories containing only observations and their corresponding high-level abstracted actions, formatted as $\{(o_t, A_t)\}_{t=1}^T$. (2) **Low-Level Action Data (D_a)**: The original trajectories containing only observations and primitive environmental actions, formatted as $\{(o_t, a_t)\}_{t=1}^T$. (3) **Chain-of-Action Data (D_{CoA})**: Trajectories containing the complete triplet of observation, generated high-level action, and the corresponding primitive action, formatted as $\{(o_t, A_t, a_t)\}_{t=1}^T$.

Training Recipes. We initialize all agents from a **Qwen2-VL-7B** (Wang et al., 2024b) post-trained on a large corpus of Minecraft-specific VQA and captioning data (Li et al., 2025a). We employ distinct training recipes tailored to the specific architecture of each agent category to ensure a fair evaluation of their respective paradigms. Standard end-to-end VLA models are trained exclusively on the low-level action dataset D_a . The training objective is to maximize the likelihood of the primitive action given the current observation, $P(a_t | o_t, ins)$. In particular, unlike **JARVIS-VLA**, which represents a_t using Raw Actions, we additionally train a **TextVLA** variant where a_t is represented as Text Actions. For the Hierarchical Agent (HA) baselines, we focus on training the high-level VLM policy using the high-level action dataset D_A , to learn the policy $P(A_t | o_t, ins)$. The low-level policies used to decode A_t are trained separately following their original implementations. Depending on the type of abstract action A produced, we denote the corresponding hierarchical agents as **LatentHA** (Latent Actions), **MotionHA** (Motion Actions), **GroundingHA** (Grounding Actions), and **SkillHA** (Skill Actions). The CoA-based agents are trained using a two-stage curriculum. First, the model is pretrained on a mixture of the high-level and low-level datasets, $D_A \cup D_a$. This stage serves as a multi-task pretraining phase, familiarizing the model with the distinct vocabularies and formats of both abstracted plans and primitive actions. The pretrained model is then fine-tuned exclusively on the Chain-of-Action dataset D_{CoA} . This stage is critical for teaching the model to bridge planning and execution by learning the joint probability $P(A_t, a_t | o_t, ins)$, as defined in Equation 2. Throughout this process, we unify the representation of a_t as Text Actions, and refer to the resulting agents as **MotionCoA** (when Motion Actions serve as abstract actions) and **GroundingCoA** (when Grounding Actions serve as abstract actions). Finally, for the all-in-one **OpenHA** agent, the datasets D_A and D_{CoA} used in

Table 3: Ablation study on the trade-off between inference mode and performance for agents using Motion and Grounding actions. We compare traditional Hierarchical Agents (e.g., **MotionHA**) using a **Fast**, decoupled inference mode against our unified models (e.g., **MotionCoA**) trained with Chain of Action (CoA) and using the **Slow**, unified autoregressive mode. FPS denotes inference speed (Frames Per Second). The CoA models consistently achieve higher Success Rates (SR) at the cost of lower FPS.

Model	Inference	FPS	Embodied Tasks			GUI Tasks			Combat Tasks		
			Steps	SR (Mini)	SR (All)	Steps	SR (Mini)	SR (All)	Steps	SR (Mini)	SR (All)
<i>Baseline</i>											
TextVLA	-	1.36	321	23.9 \pm 8.9	27.0 \pm 17.0	291	14.0 \pm 4.1	25.8 \pm 14.3	317	27.1 \pm 11.8	10.0 \pm 6.1
<i>Motion Actions</i>											
MotionHA	Fast	3.75	336	31.6 \pm 10.1	27.4 \pm 35.2	N/A	0.0 \pm 0.0	0.0 \pm 0.0	392	9.1 \pm 3.9	4.3 \pm 10.8
MotionCoA	Slow	0.98	317	28.9 \pm 11.7	31.2 \pm 14.1	316	27.4 \pm 5.1	28.7 \pm 10.9	355	19.2 \pm 3.7	15.8 \pm 4.2
<i>Grounding Actions</i>											
GroundingHA	Fast	5.61	290	39.7 \pm 13.7	37.1 \pm 38.5	380	3.7 \pm 2.3	6.7 \pm 10.8	346	28.2 \pm 6.2	26.5 \pm 23.4
GroundingCoA	Slow	1.22	313	31.3 \pm 16.6	31.8 \pm 12.5	304	33.9 \pm 8.1	30.1 \pm 4.2	327	29.1 \pm 10.8	28.3 \pm 9.9

this curriculum are composed of a heterogeneous mixture of all considered action abstraction types. **For fairness of comparison, the VLA models, the various HA models, and the OpenHA models are all trained using the same total amount of computation.** For additional training details, please refer to Appendix B.

Compared Models and Baselines. Our primary analysis focuses on a controlled, systematic comparison of agents equipped with the different action spaces detailed in Section 3, including our proposed CoA-based hierarchical VLA models and OpenHA models, which are evaluated under the settings specified in Appendix C. To contextualize our results within the broader literature, we also compare against specialized policies that were trained on the VPT dataset. These baselines include the original **VPT** (Baker et al., 2022), **ROCKET-1** (Cai et al., 2024a), and **STEVE-1** (Lifshitz et al., 2024).

4.2 FAIR EVALUATION OF ACTION SPACES IN HIERARCHICAL AGENTS

To identify the most effective action space for Hierarchical Agents (HAs), we conducted a controlled evaluation across our diverse, large-scale benchmark. Each VLM-based HA was trained on a dataset with an equivalent number of action tokens to ensure a fair comparison.

Our first finding, detailed in Table 2, is that prior methods like STEVE-1 exhibit a significant performance drop on our benchmark (e.g., 8.0% ASR on Embodied tasks). This highlights their limited generalizability, as they were often trained on narrower task sets and struggle with diverse challenges like fishing or defeating creepers. In contrast, our VLM-based HAs demonstrate substantially stronger performance, underscoring the benefits of large pre-trained models.

However, our central finding is that **no single action space is universally optimal; performance is highly contingent on the task domain.** This is illustrated in Figure 1 and supported by task-specific results: **MotionHA** achieves a nearly 100% success rate on “kill the sheep,” while **SkillHA** is the top performer on “Chop down trees” with 95% success. The performance disparities stem from the inherent trade-offs in each action space’s expressive capabilities.

The agent using grounding actions, **GroundingHA**, is the most capable HA overall, achieving the highest ASR in both Embodied (37.1%) and Combat (26.5%) tasks. Its strength lies in explicitly binding an action (e.g., mine) to an entity’s pixel coordinates, effectively leveraging the VLM’s visual grounding ability. Its primary weakness is in partially observable scenarios where the target is off-screen. Conversely, the agent using object-agnostic motion primitives, **MotionHA**, excels at navigation-heavy Embodied tasks (27.4%) but completely fails in GUI tasks (0.0%). Its object-agnostic nature is a critical flaw, as it cannot specify *what* to interact with, rendering it incapable of precise, object-centric actions required for crafting. The other agents, **LatentHA** and **SkillHA**, show moderate but not leading performance in any single category.

4.3 EXPERIMENTS ON CHAIN-OF-ACTION TRAINING

Having established that different action abstractions are specialized for different tasks, we now investigate our central hypothesis: Can incorporating these abstractions as intermediate reasoning steps within a unified VLA model improve its overall performance?

To do this, we compare our unified Chain-of-Action (CoA) models against their traditional hierarchical counterparts. As detailed in Section 3.2, our architecture supports both a **Fast (Decoupled)** mode, which mimics a traditional HA by using a separate lightweight decoder, and a **Slow (CoA)** mode, which performs full autoregressive generation of the plan and action. This allows for a direct comparison of the two paradigms.

Table 4: Performance comparison between specialist agents (trained on a single action space) and our universal **OpenHA** agent (trained on all action spaces). For each comparison, OpenHA is prompted to use the same inference format as the corresponding specialist (e.g., MotionCoA). The results show that the universally-trained OpenHA model outperforms the specialists, demonstrating positive knowledge transfer across action spaces.

Model	Inference	Embodied Tasks			GUI Tasks			Combat Tasks		
		Steps	SR (Mini)	SR (All)	Steps	SR (Mini)	SR (All)	Steps	SR (Mini)	SR (All)
<i>Text Action</i>										
TextVLA	Text	321	23.9 \pm 8.9	27.0 \pm 17.0	291	14.0 \pm 4.1	25.8 \pm 14.3	317	27.1 \pm 11.8	10.0 \pm 6.1
OpenHA	Text	326	37.0 \pm 15.9	27.5 \pm 13.8	314	33.3 \pm 13.3	32.5 \pm 9.2	304	40.0 \pm 19.6	31.9 \pm 13.7
<i>Motion Actions</i>										
MotionCoA	MotionCoA	317	28.9 \pm 11.7	31.2 \pm 14.1	316	27.4 \pm 5.1	28.7 \pm 10.9	355	19.2 \pm 3.7	15.8 \pm 4.2
OpenHA	MotionCoA	330	29.7 \pm 12.2	24.8 \pm 10.9	308	28.7 \pm 10.0	29.5 \pm 8.7	344	26.3 \pm 13.1	25.6 \pm 9.8
<i>Grounding Actions</i>										
GroundingCoA	GroundingCoA	313	31.3 \pm 16.6	31.8 \pm 12.5	304	33.9 \pm 8.1	30.1 \pm 4.2	327	29.1 \pm 10.8	28.3 \pm 9.9
OpenHA	GroundingCoA	316	35.5 \pm 11.2	30.1 \pm 13.9	288	34.6 \pm 6.6	35.1 \pm 10.4	342	30.6 \pm 15.6	29.8 \pm 12.6

The results, presented in Table 3, reveal a trade-off between inference efficiency and task performance. For agents using Motion Actions, the traditional hierarchical model (**MotionHA**) is highly efficient, operating at 3.75 FPS. Our unified model using the CoA paradigm (**MotionCoA**) is computationally more intensive, running at 0.98 FPS. However, this additional computational cost at inference time effectively compensates for MotionHA’s weakness in Combat tasks: the ASR increases three fold, from 4.3% to 15.8%. A similar trend can be observed for Grounding Actions. The success rate in Combat tasks improves from 26.5% to 28.3%. Besides, our strongest CoA model, **GroundingCoA**, also substantially outperforms the robust **TextVLA** baseline in both Embodied (31.8% vs. 27.0% ASR) and Combat (28.3% vs. 10.0% ASR) domains. This provides a clear answer to our initial question: explicitly generating an abstracted action as an intermediate “thought” allows the VLA model to better structure its decision-making process. For more details on how COA models operate, please refer to Appendix D.

We posit that this demonstrates a novel and powerful form of **inference-time scaling** for autonomous agents. While performance is commonly scaled by increasing model size or training data (at training time), CoA offers a complementary axis for improvement: allocating more computational resources at inference time to a structured, explicit reasoning process. By “thinking” more before acting, the agent achieves a higher level of performance, confirming that abstracted actions, when integrated thoughtfully, improve the capabilities of VLA models.

4.4 EFFECTIVENESS OF TRAINING ON DIVERSE ACTION SPACES

In this section, we try to answer: Does training a single agent on a mixture of action spaces enable it to learn a more generalizable and effective decision-making policy?

we conduct a series of experiments comparing our universally-trained agent, **OpenHA**, against the specialist models to answer these questions. To ensure a fair comparison, we evaluate OpenHA by constraining its output to match the format of each specialist agent. See Appendix E for details. This allows us to isolate the effect of the training data diversity. The results of this comparison are presented in Table 4.

The data reveals that: the universally-trained OpenHA agent outperforms the specialist models. For example, when using the **MotionCoA** inference format on Combat tasks, OpenHA achieves an ASR of 25.6% compared to the specialist **MotionCoA**’s 15.8%. This consistent improvement across different inference formats and task categories strongly suggests that the model benefits from the diverse training data, learning a more robust and comprehensive internal policy. It shows that OpenHA has learned a generalized concept of object interaction from its exposure to Text and Grounding actions, and it can leverage this knowledge even when forced to express its intent through an object-agnostic action format. This confirms that the All-in-One training fosters a deeper understanding of tasks that transcends any single action representation. Suggest that co-training on diverse action spaces is a powerful mechanism for improving an agent’s core reasoning and generalization capabilities.

5 CONCLUSIONS

In this work, we first conduct a large-scale analysis demonstrating that the optimal abstracted action for autonomous agents is highly task-dependent. To address this, we introduce the **Chain of Action** framework, which unifies high-level planning and low-level control within a single end-to-end VLA model. Building on this, we show that an **All-in-One** agent, trained on a diverse mixture of action spaces, learns a more generalizable policy and achieves superior performance compared to any specialist agent. Our findings suggest that synergizing multiple action representations, rather than selecting a single one, is a crucial step towards developing more capable and general-purpose autonomous agents.

REFERENCES

- 486
487
488 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo
489 Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. [arXiv preprint](#)
490 [arXiv:2303.08774](#), 2023.
- 491 Anthropic. Introducing the model context protocol, 2024. URL [https://www.anthropic.com/news/](https://www.anthropic.com/news/model-context-protocol)
492 [model-context-protocol](https://www.anthropic.com/news/model-context-protocol).
- 493 Anthropic. Introducing claude 4, 2025. URL <https://www.anthropic.com/news/claude-4>.
494
- 495 Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei
496 Huang, et al. Qwen technical report. [arXiv preprint arXiv:2309.16609](#), 2023.
- 497 Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul
498 Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos.
499 [Advances in Neural Information Processing Systems](#), 35:24639–24654, 2022.
- 500 Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Cheb-
501 otar, Debidatta Dwibedi, and Dorsa Sadigh. Rt-h: Action hierarchies using language. [arXiv preprint](#)
502 [arXiv:2403.01823](#), 2024a.
- 503 Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Cheb-
504 otar, Debidatta Dwibedi, and Dorsa Sadigh. Rt-h: Action hierarchies using language. [arXiv preprint](#)
505 [arXiv:2403.01823](#), 2024b.
- 506 Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox,
507 Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots.
508 [arXiv preprint arXiv:2503.14734](#), 2025.
- 509 Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana
510 Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world
511 control at scale. [arXiv preprint arXiv:2212.06817](#), 2022.
- 512 Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli
513 Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web
514 knowledge to robotic control. [arXiv preprint arXiv:2307.15818](#), 2023a.
- 515 Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli
516 Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web
517 knowledge to robotic control. [arXiv preprint arXiv:2307.15818](#), 2023b.
- 518 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind
519 Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners.
520 [Advances in neural information processing systems](#), 33:1877–1901, 2020.
- 521 Shaofei Cai, Zihao Wang, Kewei Lian, Zhancun Mu, Xiaojian Ma, Anji Liu, and Yitao Liang. Rocket-1:
522 Mastering open-world interaction with visual-temporal context prompting. [arXiv preprint arXiv:2410.17856](#),
523 2024a.
- 524 Shaofei Cai, Bowei Zhang, Zihao Wang, Xiaojian Ma, Anji Liu, and Yitao Liang. Groot: Learning to
525 follow instructions by watching gameplay videos. In [The Twelfth International Conference on Learning](#)
526 [Representations](#), 2024b.
- 527 Peng Chen, Pi Bu, Yingyao Wang, Xinyi Wang, Ziming Wang, Jie Guo, Yingxiu Zhao, Qi Zhu, Jun Song, Siran
528 Yang, et al. Combatvla: An efficient vision-language-action model for combat tasks in 3d action role-playing
529 games. [arXiv preprint arXiv:2503.09527](#), 2025.
- 530 Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai
531 Wang, Feng Yin, Junhua Zhao, et al. Exploring large language model based intelligent agents: Definitions,
532 methods, and prospects. [arXiv preprint arXiv:2401.03428](#), 2024.
- 533 Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and
534 Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. [The International Journal of](#)
535 [Robotics Research](#), pp. 02783649241273668, 2023.
- 536 Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl:
537 Attentive language models beyond a fixed-length context. [arXiv preprint arXiv:1901.02860](#), 2019.

- 540 Jingwen Deng, Zihao Wang, Shaofei Cai, Anji Liu, and Yitao Liang. Open-world skill discovery from
541 unsegmented demonstrations. [arXiv preprint arXiv:2503.10684](#), 2025.
- 542
- 543 Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid,
544 Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. [arXiv
preprint arXiv:2303.03378](#), 2023.
- 545
- 546 Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z Ren, Homer
547 Walke, Quan Vuong, Lucy Xiaoyang Shi, and Sergey Levine. Knowledge insulating vision-language-action
548 models: Train fast, run fast, generalize better. [arXiv preprint arXiv:2505.23705](#), 2025.
- 549 Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-
550 An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with
551 internet-scale knowledge. [Advances in Neural Information Processing Systems Datasets and Benchmarks](#),
552 2022.
- 553 Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle,
554 Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. [arXiv
preprint arXiv:2407.21783](#), 2024.
- 555
- 556 Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu,
557 Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, et al. Rt-trajectory: Robotic task generalization via
558 hindsight trajectory sketches. [arXiv preprint arXiv:2311.01977](#), 2023.
- 559
- 560 Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang,
561 Jiawei Wang, et al. Seed1. 5-vl technical report. [arXiv preprint arXiv:2505.07062](#), 2025.
- 562 William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan
563 Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations. [arXiv preprint arXiv:1907.13440](#),
564 2019.
- 565 Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov,
566 Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model.
567 [arXiv preprint arXiv:2406.09246](#), 2024.
- 568
- 569 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gon-
570 zalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with
571 pagedattention. In [Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles](#),
572 2023.
- 573 Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru
574 Wang, Sangho Lee, et al. Molmoact: Action reasoning models that can reason in space. [arXiv preprint
arXiv:2508.07917](#), 2025.
- 575
- 576 Muyao Li, Zihao Wang, Kaichen He, Xiaojian Ma, and Yitao Liang. Jarvis-vla: Post-training large-scale vision
577 language models to play visual games with keyboards and mouse. [arXiv preprint arXiv:2503.16365](#), 2025a.
- 578
- 579 Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben
580 Chen, Weichen Sun, Qiexiang Wang, et al. Chain-of-agents: End-to-end agent foundation models via
581 multi-agent distillation and agentic rl. [arXiv preprint arXiv:2508.13167](#), 2025b.
- 582
- 583 Xinghang Li, Peiyan Li, Minghuan Liu, Dong Wang, Jirong Liu, Bingyi Kang, Xiao Ma, Tao Kong, Hanbo
584 Zhang, and Huaping Liu. Towards generalist robot policies: What matters in building vision-language-action
585 models. [arXiv preprint arXiv:2412.14058](#), 2024.
- 586
- 587 Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng.
588 Code as policies: Language model programs for embodied control. [arXiv preprint arXiv:2209.07753](#), 2022.
- 589
- 590 Shalev Lifshitz, Keiran Paster, Harris Chan, Jimmy Ba, and Sheila McIlraith. Steve-1: A generative model for
591 text-to-behavior in minecraft. [Advances in Neural Information Processing Systems](#), 36, 2024.
- 592
- 593 Haowei Lin, Zihao Wang, Jianzhu Ma, and Yitao Liang. Mcu: A task-centric framework for open-ended agent
594 evaluation in minecraft. [arXiv preprint arXiv:2310.08367](#), 2023.
- 595
- 596 Qianli Ma, Yaowei Zheng, Zhelun Shi, Zhongkai Zhao, Bin Jia, Ziyue Huang, Zhiqi Lin, Youjie Li, Jiacheng
597 Yang, Yanghua Peng, et al. Veomni: Scaling any modality model training with model-centric distributed
598 recipe zoo. [arXiv preprint arXiv:2508.02317](#), 2025.

- 594 OpenAI. Chatgpt: Optimizing language models for dialogue, 2023a. URL [https://openai.com/blog/
595 chatgpt/](https://openai.com/blog/chatgpt/).
- 596
- 597 OpenAI. Gpt-4 technical report, 2023b.
- 598
- 599 openai. Operator, 2025. URL <https://openai.com/index/introducing-operator/>.
- 600 Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin
601 Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. [arXiv preprint
602 arXiv:2501.12326](https://arxiv.org/abs/2501.12326), 2025.
- 603 ByteDance Seed. Ui-tars-1.5. <https://seed-tars.com/1.5>, 2025.
- 604 Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers:
605 Cloning k modes with one stone. [Advances in neural information processing systems](https://arxiv.org/abs/2205.22955), 35:22955–22968,
606 2022.
- 607 Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalk-
608 wyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal
609 models. [arXiv preprint arXiv:2312.11805](https://arxiv.org/abs/2312.11805), 2023.
- 610
- 611 Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey
612 Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. [arXiv preprint
613 arXiv:2405.12213](https://arxiv.org/abs/2405.12213), 2024.
- 614 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix,
615 Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation
616 language models. [arXiv preprint arXiv:2302.13971](https://arxiv.org/abs/2302.13971), 2023.
- 617 Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. [Advances in neural
618 information processing systems](https://arxiv.org/abs/1704.00027), 30, 2017.
- 619 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and
620 Illia Polosukhin. Attention is all you need. [Advances in neural information processing systems](https://arxiv.org/abs/1706.03762), 30, 2017.
- 621
- 622 Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert,
623 Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. [https:
624 //github.com/huggingface/trl](https://github.com/huggingface/trl), 2020.
- 625 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima
626 Anandkumar. Voyager: An open-ended embodied agent with large language models. [Transactions on
627 Machine Learning Research](https://arxiv.org/abs/2401.14409), 2024a.
- 628 Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang,
629 Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model’s perception of the world at any resolution.
630 [arXiv preprint arXiv:2409.12191](https://arxiv.org/abs/2409.12191), 2024b.
- 631 Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. Describe,
632 explain, plan and select: interactive planning with large language models enables open-world multi-task
633 agents. In [Proceedings of the 37th International Conference on Neural Information Processing Systems](https://arxiv.org/abs/2310.12977), pp.
634 34153–34189, 2023.
- 635 Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bawei Zhang, Haowei Lin, Zhaofeng He, Zilong
636 Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal
637 language models. [IEEE Transactions on Pattern Analysis and Machine Intelligence](https://arxiv.org/abs/2401.14409), 2024c.
- 638 Zihao Wang, Shaofei Cai, Zhancun Mu, Haowei Lin, Ceyao Zhang, Xuejie Liu, Qing Li, Anji Liu, Xiaojian Ma,
639 and Yitao Liang. Omnijarvis: Unified vision-language-action tokenization enables open-world instruction
640 following agents. [Advances in Neural Information Processing Systems](https://arxiv.org/abs/2401.14409), 2024d.
- 641
- 642 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain
643 of thought prompting elicits reasoning in large language models. [36th Conference on Neural Information
644 Processing Systems \(NeurIPS 2022\)](https://arxiv.org/abs/2201.07900), 2022.
- 645 Lilian Weng. Llm-powered autonomous agents. [lilianweng.github.io,
646 github.io/posts/2023-06-23-agent/](https://lilianweng.github.io/posts/2023-06-23-agent/), Jun 2023. URL [https://lilianweng.
647 github.io/posts/2023-06-23-agent/](https://lilianweng.github.io/posts/2023-06-23-agent/).
- 648 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen
649 Huang, Chenxu Lv, et al. Qwen3 technical report. [arXiv preprint arXiv:2505.09388](https://arxiv.org/abs/2505.09388), 2025.

- 648 Hui Yang, Sifu Yue, and Yunzhong He. Auto-gpt for online decision making: Benchmarks and additional
649 opinions. [arXiv preprint arXiv: 2306.02224](#), 2023.
- 650
651 Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiyang Yu, Ya-Qin Zhang, Wei-Ying
652 Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based
653 memory agent. [arXiv preprint arXiv:2507.02259](#), 2025.
- 654 Haoqi Yuan, Zhancun Mu, Feiyang Xie, and Zongqing Lu. Pre-training goal-based models for sample-efficient
655 reinforcement learning. In [The Twelfth International Conference on Learning Representations](#), 2024.
- 656 Michal Zawalski, William Chen, Karl Pertsch, Oier Mees, Chelsea Finn, and Sergey Levine. Robotic control via
657 embodied chain-of-thought reasoning. [arXiv preprint arXiv:2407.08693](#), 2024.
- 658
659 Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt:
660 Enhancing robotic control with hierarchical robot transformers. [arXiv preprint arXiv:2410.05273](#), 2024.
- 661
662 Jianke Zhang, Yanjiang Guo, Yucheng Hu, Xiaoyu Chen, Xiang Zhu, and Jianyu Chen. Up-vla: A unified
663 understanding and prediction model for embodied agent. [arXiv preprint arXiv:2501.18867](#), 2025.
- 664
665 Zhonghan Zhao, Wenwei Zhang, Haiyan Huang, Kuikun Liu, Jianfei Gao, Gaoang Wang, and Kai Chen. Rig:
666 Synergizing reasoning and imagination in end-to-end generalist policy. [arXiv preprint arXiv:2503.24388](#),
667 (arXiv:2503.24388), March 2025. URL <http://arxiv.org/abs/2503.24388>. arXiv:2503.24388 [cs].
- 668
669 Ruijie Zheng, Yongyuan Liang, Shuaiyi Huang, Jianfeng Gao, Hal Daumé III, Andrey Kolobov, Furong Huang,
670 and Jianwei Yang. Tracevla: Visual trace prompting enhances spatial-temporal awareness for generalist
671 robotic policies. [arXiv preprint arXiv:2412.10345](#), 2024.
- 672
673 Yifan Zhong, Fengshuo Bai, Shaofei Cai, Xuchuan Huang, Zhang Chen, Xiaowei Zhang, Yuanfei Wang,
674 Shaoyang Guo, Tianrui Guan, Ka Nam Lui, et al. A survey on vision-language-action models: An action
675 tokenization perspective. [arXiv preprint arXiv:2507.01925](#), 2025a.
- 676
677 Yifan Zhong, Xuchuan Huang, Ruochong Li, Ceyao Zhang, Yitao Liang, Yaodong Yang, and Yuanpei Chen.
678 Dexgraspvla: A vision-language-action framework towards general dexterous grasping, 2025b.
- 679
680 Enshen Zhou, Yiran Qin, Zhenfei Yin, Yuzhou Huang, Ruimao Zhang, Lu Sheng, Yu Qiao, and Jing Shao.
681 Minedreamer: Learning to follow instructions via chain-of-imagination for simulated-world control. [arXiv
682 preprint arXiv:2403.12037](#), 2024.
- 683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701

THE USE OF LARGE LANGUAGE MODELS (LLMs)

We employed Large Language Models (LLMs) (OpenAI, 2023b; Team et al., 2023) to assist with proofreading and language refinement in this article. The LLMs were used solely for improving grammar, spelling, and clarity of expression. No part of the scientific content, analysis, or conclusions was generated by the LLMs; all research ideas, experiments, and results are original to the authors. The use of LLMs did not affect the integrity or validity of the scientific contributions.

REPRODUCIBILITY STATEMENT

We provide an anonymous code release, including inference scripts and evaluation pipelines. Appendix B presents full details of the training setup (hardware, software versions and complete training recipe). Appendix B.2 explains the dataset construction and preprocessing, including our programmatic labeling pipeline. Table 7 summarizes the specific training recipes for each agent variant.

A suite of specialist models trained under individual action spaces—namely MotionHA, GroundingHA, and TextVLA—alongside a unified **OpenHA** model jointly trained across multiple abstraction types, is planned for release. Complementing these models, we prepare large-scale datasets aligned with different action spaces: single-space datasets (Motion, Grounding, Text) and Chain-of-Action (CoA) datasets that integrate high-level reasoning with low-level execution. These resources are intended to enable fine-tuning and benchmarking, facilitate reproducibility, and accelerate progress in open-ended hierarchical agent research.

A DETAILS OF ABSTRACTED ACTION SPACE USED IN DHA

We adopt multiple abstracted action spaces, including four core types: **Skill Actions**, **Grounding Actions**, **Motion Actions**, and **Latent Actions**. These abstract actions are represented as action tokens that guide the low-level policy during execution. Below, we provide details of each abstract action space and its semantics.

A.1 SKILL ACTION

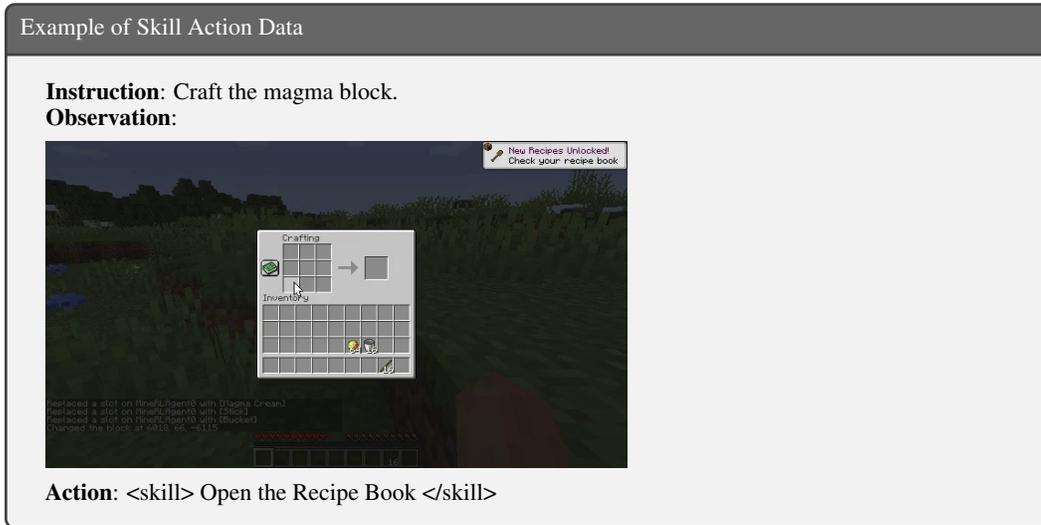
Skill Actions (Driess et al., 2023) refer to semantically meaningful, atomic task segments that can serve as self-contained skill units within a larger instruction-following trajectory. In the context of long-horizon tasks, these actions act as high-level decompositions—each skill action represents a discrete subgoal that contributes to accomplishing the overarching instruction. For example, the instruction `Obtain a diamond` might be decomposed into a sequence of skill actions such as `Gather wood`, `Craft a pickaxe`, and `Mine stone`.

To construct a diverse and grounded set of skill actions, we begin by collecting discrete in-game events from the simulator environment (Guss et al., 2019). The events include common player interactions such as: *crafting an item*, *mining a block*, *killing an entity*, *dropping an item*, *using a tool*, and *interacting with blocks*.

These raw events are then provided to an LLM (Achiam et al., 2023), which autonomously clusters and merges semantically related events into higher-level skill task descriptions. This merging process is designed to reflect human-like reasoning about goal structures (e.g., “chop trees and collect logs” or “smelt iron ore into ingots”).

Skill Actions are used during training and inference as shown in Table A.1, which illustrates a representative example.

756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809



A.2 GROUNDING ACTIONS

Grounding actions are designed to facilitate fine-grained interactions with visual targets in the environment by referencing specific spatial coordinates. When an object is visible and interactable in the current observation frame, we associate it with a point-based reference and assign a corresponding action type. If the object is not observable, the agent instead performs a general behavior (e.g., exploration).

To collect grounding action data, we follow a modified version of the procedure proposed in Jarvis-VLA (Li et al., 2025a). Specifically, we manually annotate the (x, y) coordinates of interactable objects across 2,000 human gameplay trajectories. Each trajectory yields an average of 5 valid grounding samples. We further implement Backward Trajectory Relabeling (Cai et al., 2024a) to propagate the object references to earlier timesteps ($t - k$), allowing for trajectory alignment. All relabeled coordinates are then manually verified, and erroneous or out-of-frame points are filtered out.

The full list of supported grounding actions and their formats is presented in Table 5.

Table 5: The grounding action space used in DHA.

Index	Action	Format	Description
1	Kill	Kill < point_start >(x,y)< point_end >	Attack the mob located at coordinate (x, y) . The target must appear in the current frame.
2	Mine	Mine < point_start >(x,y)< point_end >	Destroy the block located at coordinate (x, y) . The target must appear in the current frame.
3	Approach	Approach < point_start >(x,y)< point_end >	Move toward the object at coordinate (x, y) . The object must be visible in the current observation.
4	Explore	Explore	Explore the environment when no specific target is visible in the current frame.
5	Right Click	Right Click < point_start >(x,y)< point_end >	Use the right-click interaction on the object at coordinate (x, y) .
6	Move To	Move To < point_start >(x,y)< point_end >	Move the cursor in the GUI to coordinate (x, y) and interact.
7	No-op	No-op	Remain idle and perform no action.

Below is an example of grounding action representation used during training and inference.

810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863

Example of Grounding Action Data

Instruction: Mine the wall torch to remove it.



Observation:

Action: Mine <|point_start|>(432, 604)<|point_end|>

A.3 MOTION ACTION

Motion actions are mid-level abstractions derived by analyzing the human action space defined in the Minecraft Wiki. To construct these, we follow a methodology similar to RT-H (Belkhale et al., 2024a), in which temporally and semantically adjacent raw actions are grouped into unified motion primitives. This abstraction enables more structured and interpretable behavior modeling while maintaining sufficient fidelity to the original control granularity.

By clustering similar low-level key or mouse inputs (e.g., repeated `W` presses and mouse movements for forward walking and camera turning), we generate compact representations such as `Turn left and down` or `Jump forward`. These motion-level actions serve as a bridge between high-level symbolic planning (e.g., grounding or skill actions) and low-level execution (raw keyboard/mouse events), and are directly executable by their corresponding motion policy within DHA’s Mixture-of-Experts module.

The set of Motion Actions is shown in Table 6.

Table 6: The basic Motion Action space we use in Minecraft.

Index	Motion Action	Description
1	<motion>turn left and down</motion>	Move the camera to the lower left.
2	<motion>turn right and down</motion>	Move the camera to the upper left.
3	<motion>turn left and up</motion>	Move the camera to the upper left.
4	<motion>turn right and up</motion>	Move the camera to the upper right.
5	<motion>attack / mine</motion>	Press and hold the left mouse button to attack or mine.
6	<motion>move forward</motion>	Press and hold the forward key to move forward.
7	<motion>jump</motion>	Jump. When swimming, it keeps the player afloat.
8	<motion>sprint</motion>	Move quickly in the direction of current motion.
9	<motion>use</motion>	Interact with the block.
10	<motion>hotbar.[1-9]</motion>	Selects the appropriate hotbar item from 1 to 9.

The model can output combined motions by merging different motions mentioned above into the same <motion> tag and separating them with commas in sequence, for example: <motion>move forward, turn right and up</motion> means the combination of <motion>move forward</motion> and <motion>turn right and up</motion>.

Below is an example of motion action

864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917

Example of Motion Action Data

Instruction: Mine the white concrete block.

Observation:



Action: <motion> turn left and down </motion>

B DETAILS OF TRAINING

Table 7: Training recipes for different agent models.

Model	HA	VLA	Hierarchical VLA		OpenHA Model	
Stage	1	1	1	2	1	2
Datasets	$D^A[GA MA SA]$	D^a	D^A+D^a	D^{CoA}	$D^A[MA GA]+D^a$	$D^{CoA}[MA+GA]$
Consumed Tokens	1.58B	3.59B	3.33B	0.27B	3.40B	0.22B
Learning Rate	5e-6	1e-5	1e-5	5e-6	1e-5	5e-6
Training Steps	4000	10400	10800	400	8200	300

B.1 TRAINING CONFIGURATION

To ensure fairness, as mentioned throughout the paper, we keep the training settings as consistent as possible. All experiments were conducted on 32 NVIDIA A800-SXM4-80GB GPUs with Python 3.10 and CUDA 12.6. We adopt imitation learning using the AdamW optimizer with hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $weight_decay = 0.1$ to train Qwen2-VL-based models (Wang et al., 2024b). The batch size is fixed at 128. We employ a cosine learning rate schedule with a minimum learning rate of 1×10^{-6} , a maximum gradient norm of 1.0, and a fixed random seed of 42. This training setup first builds foundational knowledge and then enables the model to learn the crucial link between high-level actions and low-level execution. We use the TRL (von Werra et al., 2020) and VeOmni (Ma et al., 2025) libraries to train the agent models.

B.2 CONSTRUCTION OF DATASETS

Our raw training data is derived from the OpenAI Video Pre-Training (VPT) dataset (Baker et al., 2022), which contains a large collection of expert trajectories in Minecraft. However, the original VPT dataset only provides pairs of visual observations and corresponding low-level environmental actions (o_t, a_t) . To enable the training of agents with high-level action spaces, we developed a programmatic labeling pipeline. This pipeline applies a set of rule-based heuristics to transform the raw low-level action sequences into their corresponding high-level abstracted representations (A_t) , including Latent, Motion, Grounding, and Language Skill actions.

The detailed training parameters and datasets are summarized in Table 7.

C DETAILS OF INFERENCE

We standardize the output format of all agentic models. For each interaction, both the CoA model and the hierarchical agentic model are restricted to producing a single executable instruction. The complete output format is shown below.

918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971

```

<|im_start|>system\nYou are a helpful assistant.<|im_end|>
<|im_start|>user\n[system_prompt] [instruction] [observation_1]<|im_end|>
<|im_start|>assistant\n [Action_1]<|im_end|>
...
<|im_start|>user\n[observation_t]<|im_end|>
<|im_start|>assistant\n

```

For CoA models, this constraint means that only one environment action can be produced at each step. For hierarchical models, whenever the high-level VLM outputs an abstract action, the corresponding action policy is allowed to execute up to four instructions.

During inference, we allow agentic models to maintain their own working memory. Specifically, the VLM can retain a trajectory of up to 15 steps, keeping track of its previous outputs together with the observed environmental information.

We adapt vLLM (Kwon et al., 2023) to accelerate inference speed.

D CASE STUDY ON COA MODELS

In this section, we present a detailed case study to evaluate the performance of the OpenHA model using the Chain-of-Action (CoA) framework. We selected four successful rollouts performed by OpenHA, two from the **GroundingCoA Inference Format** and two from the **MotionCoA Inference Format**.

As discussed in Section 3, each task begins with the model generating an abstract action, which serves as an intermediate reasoning step, followed by the model outputting the corresponding **primitive action**. To improve interpretability and reduce the complexity of displaying hundreds of interactions, we use a time-adjacent clustering strategy. This method groups consecutive, similar abstract actions that occur within a specific time window. For example, in the **GroundingCoA mode**, if the model outputs the same abstract action (e.g., “move camera”) repeatedly over several time steps, we consider these actions part of the same grounding action as long as the Euclidean distance between action points remains below a threshold (e.g., 50 pixels). In the **MotionCoA mode**, abstract actions are grouped together if the action type remains consistent over time.

Rather than displaying each individual step, we group the primitive actions generated for a particular abstract action during a time slice. For instance, if the abstract action is “turn right,” and the model repeatedly generates similar low-level actions (such as moving the camera), we aggregate these primitive actions into a single **action sequence**. These grouped sequences are then analyzed to assess task success.

Figures 2, 3, 4, and 5 illustrate the model’s execution for each task, with both the abstract and primitive actions performed at each step.

From these case studies, we observe that an abstract action often corresponds to a specific pattern of primitive actions. For example, when the grounding action is ‘move camera to (x,y)’, the primitive action sequence typically begins with several mouse movements followed by a series of mouse clicks. This observation suggests that the CoA framework operates as a two-phase reasoning process: first, the model decides on the **abstract action**, and then it generates the **primitive actions** based on the abstract plan. In other words, CoA models decompose tasks into abstract actions, which are then broken down into executable actions, facilitating a clear separation between reasoning and action execution.

This two-stage process bears resemblance to the Chain-of-Thought framework, where intermediate reasoning steps guide the model’s eventual execution of low-level tasks. In this analogy, the abstract action can be thought of as a “rationale,” similar to the “thought” process in CoT. However, instead of purely reasoning through abstract steps in isolation, CoA uses this intermediate action to directly inform the subsequent low-level actions, effectively bridging high-level planning with executable action in an integrated manner.

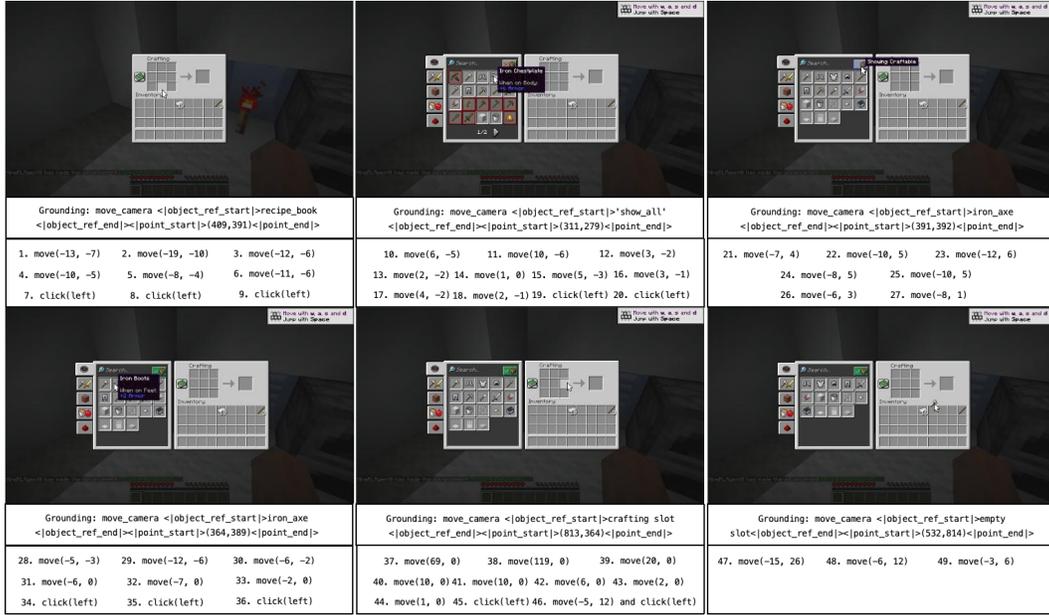


Figure 2: Execution of the "Craft an Iron Sword" task using the GroundingCoA inference format.

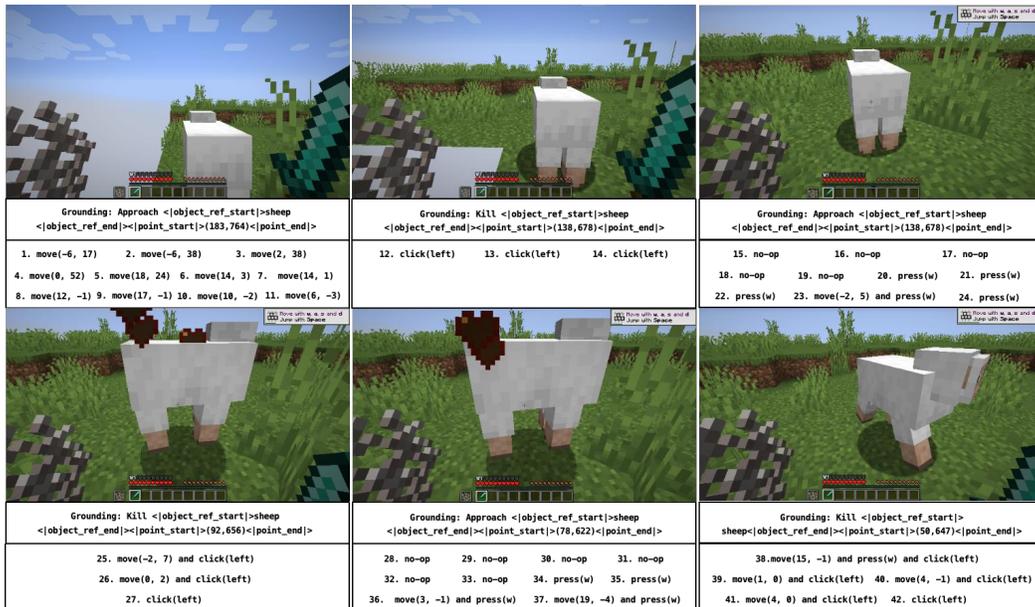
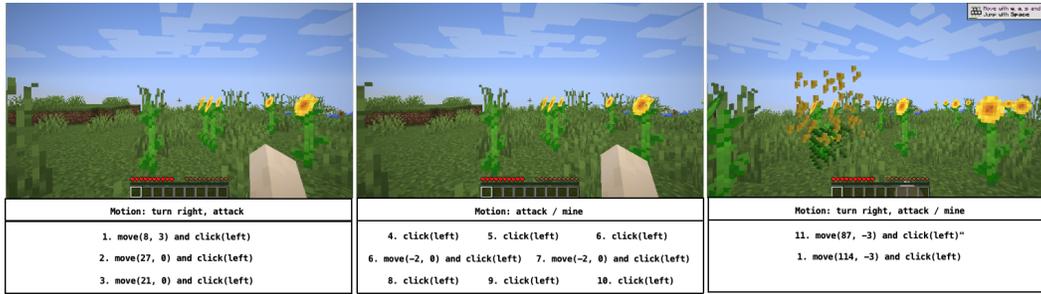


Figure 3: Execution of the "Kill the Sheep" task using the GroundingCoA inference format.

1026
1027
1028
1029
1030
1031
1032
1033
1034
1035



1036 Figure 4: Execution of the "Mine a Sunflower" task using the MotionCoA inference format.

1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057



1058 Figure 5: Execution of the "Mine a Sugar Cane" task using the MotionCoA inference format.

1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069

1070 E OPENHA OUTPUT FORMAT CONSTRAINTS

1071
1072
1073
1074
1075
1076
1077
1078
1079

Foundation models possess strong instruction-following abilities (Brown et al., 2020). We leverage different contexts to control the output format of OpenHA. For example, if we want OpenHA to produce outputs in the same format as MotionCoA, we must specify in the system prompt that OpenHA should generate actions in a “first motion, then text action” structure. For decision-making models, a well-designed system prompt should include the agent’s role-play identity, the assigned task, the action space it must adhere to, and the organizational form of the chain of action. In this section, we introduce the system prompts used by OpenHA across different modalities.

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

System Prompt for TextVLA-like Output

```

You are an AI agent performing tasks in Minecraft based on given
instructions, action history, and visual observations (screenshots).
Your goal is to take the optimal action to complete the task.

## Output Format
Action: ...

## Action Space
* move('dx', 'dy') # Move the mouse position; dx and dy represent horizontal
  and vertical movement, respectively.
* click('left' or 'right') # left click or right click the mouse
  - left_click # Attack; In GUI, pick up the stack of items or place the
    stack of items in a GUI cell; when used as a double click (attack -
    no attack - attack sequence), collect all items of the same kind
    present in inventory as a single stack.
  - right_click # Place the item currently held or use the block the
    player is looking at. In GUI, pick up the stack of items or place a
    single item from a stack held by mouse.
* press(keys) # press the keyboard buttons
  - 'w' # Move forward.
  - 's' # Move backward.
  - 'a' # Strafe left.
  - 'd' # Strafe right.
  - 'e' # Open or close inventory and the 2x2 crafting grid.
  - 'space' # Jump.
  - 'q' # Drop a single item from the stack of items the player is
    currently holding. If the player presses ctrl+q then it drops the
    entire stack. In the GUI, the same thing happens except to the item
    the mouse is hovering over.
  - '1'-'9' # Switch active item to the one in a given hotbar cell.
  - 'left.control' # Move fast in the current direction of motion.
  - 'left.shift' # Move carefully in current direction of motion. In the
    GUI it acts as a modifier key: when used with attack it moves item
    from/to the inventory to/from the hotbar, and when used with craft
    it crafts the maximum number of items possible instead of just 1.
* no_op # wait and do not interact with the world

If multiple actions are activated, use ', ' connect.

## User Instruction
{instruction}

```

1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187

System Prompt for MotionHA-like Output

You are an AI agent performing tasks in Minecraft based on given instructions and visual observations (screenshots). Your goal is to take the next optimal action to complete the task.

Action Space

- * move <direction> # Move in one or more of the following directions: forward, backward, right, left. You may combine directions (e.g., forward and left).
- * sprint # Sprint in the current movement direction.
- * sneak # Sneak; modifies movement in GUI and world.
- * turn <direction> # Turn the camera view. Supported directions: up, down, right, left, or combinations like left and down.
- * cursor move <direction> # Move the GUI cursor. Supported directions: up, down, right, left, or combinations like right and up.
- * jump
- * drop # Drop one item; Use Ctrl + drop to release the full stack.
- * swap # Swap items between hands.
- * switch hotbar # Switch to a different hotbar slot.
- * open inventory
- * close inventory
- * close gui
- * attack / mine # Use the main hand to attack or break a block.
- * place / use # Use or place the held item.
- * operate gui # Click or operate an element in the GUI.
- * choose

If multiple actions are activated, use ', ' connect.

Continuously take action until the task is completed.

Motion: motion action

User Instruction

{instruction}

1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241

System Prompt for GroundingHA-like Output

```

You are an AI agent performing tasks in Minecraft based on given
instructions, action history, and visual observations (screenshots).
Your goal is to take the next optimal action to complete the task.

# Action
## You have the following actions.
* Kill <|point_start|>(x,y)<|point_end|> # Kill the Minecraft mobs marked by
the coordinate (x, y). Note: The target of Action Kill needs to appear
in the current observation image.
* Mine <|point_start|>(x,y)<|point_end|> # Destroy the Minecraft blocks
marked by the coordinate (x, y). Note: The target of Action Mine needs
to appear in the current observation image.
* Approach <|point_start|>(x,y)<|point_end|> # Approach to the target object
marked by the coordinate (x, y). Note: The target of Action Approach
needs to appear in the current observation image.
* Explore # Explore the environment. Note: The target of Action Explore
should not appear in current observation image.
* right click <|point_start|>(x,y)<|point_end|> # Use the right click button
to interact with the object marked by the coordinate (x, y).
* move_camera <|point_start|>(x,y)<|point_end|> # Move the cursor in the GUI
to the location of the object identified by the coordinate (x, y).
* no-op # Wait and do not interact with the world.

If multiple actions are activated, use ', ' connect.

## Continuously take action until the task is completed.
Grounding: grounding actions
## User Instruction
{instruction}

```

1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

System Prompt for MotionCoA-like Output

```

You are an AI agent performing tasks in Minecraft based on given
instructions, history, and visual observations (screenshots). Your goal
is to take the next optimal action to complete the task.
## Action Space
Your action space is hierarchical, including motion-level and raw-action.
You need to output a hierarchical action chain until the final
raw-action is output. The action spaces at each level are as follows:

### motion action

* move <direction>
* sprint / sneak
* turn <direction>
* cursor move <direction>
* jump
* drop
* swap
* switch hotbar
* open / close inventory
* inventory
* close gui
* attack / mine
* place / use
### raw actions

* move('dx', 'dy') # Move the mouse position; dx and dy represent horizontal
and vertical movement, respectively.
* click('left' or 'right') # left click or right click the mouse
  - left_click # Attack;
  - right_click # Place the item currently held or use the block the
player is looking at.
* press(keys) # press the keyboard buttons
  - 'w' # forward W key Move forward.
  - 's' # Move backward.
  - 'a' # Strafe left.
  - 'd' # Strafe right.
  - 'e' # Open or close inventory and the 2x2 crafting grid.
  - 'space' # Jump.
  - 'q' # Drop a single item from the stack of items the player is
currently holding.
  - '1'-'9' # Switch active item to the one in a given hotbar cell.
  - 'left.control' # Move fast in the current direction of motion.
  - 'left.shift' # Move carefully in current direction of motion.
* no_op # wait and do not interact with the world
If multiple actions are activated, use and connect.
## Continuously take action until the task is completed.
Wrap your hierarchical action:
Motion: motion actions
Action: raw actions
## User Instruction
{instruction}

```

1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349

System Prompt for GroundingCoA-like Output

```

You are an AI agent performing tasks in Minecraft based on given
instructions, history, and visual observations (screenshots). Your goal
is to take the next optimal action to complete the task.
## Action Space
Your action space is hierarchical, including grounding-level and raw-action.
You need to output a hierarchical action chain until the final
raw-action is output. The action spaces at each level are as follows:

### grounding actions
* Kill <|object_ref_start|>mob<|object_ref_end|>
  <|point_start|>(x,y)<|point_end|> # Kill the Minecraft mobs.
* Mine <|object_ref_start|>block<|object_ref_end|>
  <|point_start|>(x,y)<|point_end|> # Destroy the Minecraft blocks.
* Approach <|object_ref_start|>object<|object_ref_end|>
  <|point_start|>(x,y)<|point_end|> # approach to the target object.
* right click <|point_start|>(x,y)<|point_end|> # use the right lick button
  to interact with the object.
* move to <|point_start|>(x,y)<|point_end|> # Move the cursor in the GUI to
  the location of the object.
* no-op

### raw actions
* move('dx', 'dy') # Move the mouse position; dx and dy represent horizontal
  and vertical movement, respectively.
* click('left' or 'right') # left click or right click the mouse
  - left_click # Attack;
  - right_click # Place the item currently held or use the block the
    player is looking at.
* press(keys) # press the keyboard buttons
  - 'w' # forward W key Move forward.
  - 's' # Move backward.
  - 'a' # Strafe left.
  - 'd' # Strafe right.
  - 'e' # Open or close inventory and the 2x2 crafting grid.
  - 'space' # Jump.
  - 'q' # Drop a single item from the stack of items the player is
    currently holding.
  - '1'-'9' # Switch active item to the one in a given hotbar cell.
  - 'left.control' # Move fast in the current direction of motion.
  - 'left.shift' # Move carefully in current direction of motion.
* no_op # wait and do not interact with the world
If multiple actions are activated, use and connect.

## Continuously take action until the task is completed.
Wrap your hierarchical action:
Grounding: grounding actions
Action: raw actions
## User Instruction
{instruction}

```

Table 8: Task configurations in the combat, embodied, and GUI task categories.

Task name	Seed	Coordinates	Resources provided	Language instruction
kill_entity:sheep	3001859308298079279	[31, 64, 61]	Diamond sword	Kill the sheep.
kill_entity:slime	3001859308298079279	[31, 64, 61]	Diamond axe	Combat the slime.
mine_block:bamboo	1	[170, 65, 69]	None	Mine bamboo with your hand.
mine_block:glass	1997	[2554, 74, 1837]	Iron axe	Mine the glass.
craft_item:diamond_hoe	1	[-1838, 72, 30601]	2 sticks, 2 diamonds	Craft a diamond hoe.
craft_item:glass_bottle	1	[-1838, 72, 30601]	3 glass	Craft a glass bottle with glass.

F OPENHA BENCHMARK

The open-ended game *Minecraft* provides an excellent foundation for studying different types of action spaces. Building upon the environment configuration of Lin et al. (2023), we design a benchmark of 800 diverse task settings, organized into three major categories: *embodied tasks*, *GUI tasks*, and *combat tasks*. Each agentic model is evaluated on every task with three independent trials, and we record both the number of steps required to complete the task and the success rate. Table 8 presents an overview of representative task configurations.