A. Environment Details

In this work, we utilize Minecraft as the primary testbed for evaluating our agent's capabilities in multi-task reinforcement learning following large-scale pretraining. Minecraft provides a rich, dynamic environment that necessitates both embodied interactions (e.g., navigation, combat) and complex GUI-based operations (e.g., crafting, inventory management). This duality makes it an ideal benchmark for testing generalist agents capable of mastering diverse, open-ended tasks ranging from low-level motor control to high-level strategic planning.

A.1. Environment Configuration

Minecraft is an open-world sandbox game where players interact with and modify a procedurally generated 3D world. The game offers a vast array of tasks that resemble real-world challenges. Consequently, agents must navigate complex terrains, manage sparse rewards, and engage in long-horizon planning. Additionally, the game's popularity provides a wealth of potential training data, including gameplay videos and textual tutorials, making it invaluable for research into large-scale learning from unstructured data.

Our experimental setup follows the Variable Pretraining Task (VPT) protocol [4], where agents interact with the game in a manner identical to human players. Specifically, the **observation space** consists exclusively of raw RGB screenshots as visual input, with a resolution of 640×360, at a frequency of 20 Hz. No high-level or privileged state information (such as voxel grids or coordinate data) is provided during evaluation. Regarding **interaction modes**, the environment presents a dual challenge: while most observations consist of embodied first-person views, tasks such as crafting and smelting require the agent to operate within distinct GUI interfaces. As illustrated in Figure 5, the agent must seamlessly switch between handling continuous embodied interactions and discrete GUI-based operations.

A.2. Raw Action Space

The raw action space is designed to align with human interfaces, utilizing the native mouse and keyboard controls provided by MineRL v0.4 (Minecraft version 1.16.5) [19]. A summary of the action mapping is provided in Table 3.

Mouse Control. Mouse displacements are discretized into 1800 bins, and their semantic meaning is context-dependent. In **embodied mode**, these displacements control the camera orientation via pitch (Δy) and yaw (Δx) adjustments; in **GUI mode**, they correspond to 2D cursor movements on the screen. Mouse clicks are encoded as dedicated tokens representing left, right, and middle button presses.

Keyboard Control Keyboard actions are treated as unique tokens, covering alphabetic characters (e.g., 'W', 'A', 'S', 'D' for movement), numeric digits (for hotbar selection), and special keys (e.g., `Shift`, `Space`, `Esc`).

A.3. Challenges for Reinforcement Learning

Minecraft presents several unique challenges that test the robustness and generalization capabilities of RL agents:

High Information Density The environment is visually complex, featuring diverse textures, lighting conditions, and objects. Agents must efficiently process high-dimensional sensory inputs to identify relevant cues while filtering out irrelevant background noise. Specific challenges include distinguishing block types for crafting or spotting hostile mobs in low-light conditions. This requires strong visual representation learning and attention mechanisms.

Task Variety and Generalization The open-ended nature of Minecraft offers a vast spectrum of tasks, ranging from primitive resource gathering to complex architectural construction. Agents must learn to generalize skills across these tasks. For instance, the motor skills required to chop a tree can be adapted to combat, while the logic used for crafting a wooden pickaxe serves as a foundational step for crafting diamond tools. This variety forces the agent to acquire adaptable, transferable strategies rather than memorizing fixed sequences.

Reward Sparsity Unlike arcade games with frequent score updates, Minecraft rewards are often extremely sparse and delayed. For example, crafting a high-level item (e.g., a diamond sword) requires a long sequence of prerequisite actions: mining wood, crafting a crafting table, mining stone, smelting iron, and finding diamonds. The agent typically receives no feedback until the final goal is achieved. This necessitates efficient exploration strategies and the ability to reason over long horizons.

Dynamic Environment The game features a dynamic world with day-night cycles, changing weather, and autonomous entities (mobs). An agent must adapt its decision-making to these variables—for example, seeking shelter at night to avoid monsters or navigating slippery terrain during rain. This dynamic nature ensures that agents are tested on their ability to adapt to non-stationary environments.

B. Action Spaces

In the context of Hierarchical Agents (HA), the design of the action space is pivotal, effectively dictating the agent's granularity of interaction with the environment. Existing approaches utilize varying levels of abstraction, spanning

Figure 5. Representative observations in Minecraft, consisting of 640×360 RGB images. The agent must handle diverse visual contexts, including embodied first-person views (left) and GUI-based interfaces (right) for tasks like crafting and inventory management.

Table 3. Summary of the human-aligned raw action space used by our agent. The agent interacts via standard keyboard and mouse inputs, identical to a human player.

Action Type	Human Input	Description
Movement	W	Move forward.
	S	Move backward.
	A	Strafe left.
	D	Strafe right.
	Space	Jump / Swim up.
	Left Shift	Sneak (prevents falling off edges).
	Left Ctrl	Sprint (increases movement speed).
Interaction	Left Mouse	Break blocks (hold) / Attack entities (click).
	Right Mouse	Place blocks / Interact with items / Open GUIs.
Inventory	Keys 1–9	Select corresponding hotbar slot.
Camera	Mouse X	Yaw: Horizontal rotation ($-180°$ to $180°$).
	Mouse Y	Pitch: Vertical rotation ($-180°$ to $180°$).

from atomic environmental controls to high-level semantic planning. The controllers associated with these spaces range in complexity from simple heuristic scripts to sophisticated Large Language Models (LLMs). Typically, as the level of abstraction increases, the reliability of execution becomes harder to guarantee, necessitating more robust decoders to bridge the gap between intent and execution.

Raw Actions represent the lowest level of the hierarchy. They directly link the agent to the environment's native interface by mimicking basic input devices, such as keyboards and mice [22, 56].

Language Skills operate at a high level of abstraction, where the action space comprises goal-oriented commands expressed in natural language. These semantic instructions are interpreted by the agent's policy, which decodes them into sequences of lower-level primitives conditioned on the current visual observation [14, 45].

Motion Actions serve as an intermediate abstraction, focusing on temporally extended, object-agnostic movement primitives. By encapsulating complex navigational or manipulation behaviors without binding them to specific object instances, they offer a more flexible representation compared to step-by-step atomic actions [6, 26].

Grounding Actions further enhance abstraction by explicitly incorporating spatial constraints. This paradigm enables the agent to target specific objects based on their coordinates in visual space. By decoupling the semantic intent of an action from its spatial execution parameters, grounding actions significantly improve generalization across diverse visual scenarios [11, 23, 57].

Latent Actions are derived via self-supervised learning rather than manual definition. Typically encoded into continuous embeddings and subsequently discretized into tokens, these actions allow the agent to operate within a learned abstract space. This approach eliminates the need for human-engineered heuristics, facilitating the acquisition of complex policies directly from large-scale offline data [12, 37, 43, 55].

Collectively, these diverse action spaces contribute to the hierarchical structure of modern agents, providing a scalable and flexible framework for managing tasks of increasing complexity.

C. RL Algorithms

As detailed in section 3, our training framework incorporates two reinforcement learning phases: Single-Turn RL (STRL) and Multi-Turn RL (MTRL). Both phases utilize Group Relative Policy Optimization (GRPO) as the underlying optimization algorithm.

C.1. Single-Turn GRPO

Standard GRPO optimizes the policy by leveraging group-level statistics. The optimization objective follows the formulation defined in Equation 1 in section 3.

Unlike PPO, GRPO calculates the advantage $\hat{A}_i$ directly from the sampled group outputs without a critic network:

$$\hat{A}_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\}) + \delta} \tag{7}$$

where r_i is the reward for the i-th output, and δ (e.g., 10^{-8}) is a small constant added for numerical stability.

The inclusion of the KL-divergence penalty term βD_{KL} acts as a regularizer, ensuring the updated policy does not deviate significantly from the reference model, thereby preventing reward hacking and maintaining linguistic coherence.

C.2. Multi-Turn GRPO

In the multi-turn setting, we adapt the single-turn formulation to handle sequential interactions. We decompose each trajectory τ into a set of state–response pairs and assign the final trajectory-level reward $r(\tau)$ to all intermediate responses.

Let $\mathcal{T}$ denote the trajectory space. The Multi-Turn GRPO objective is defined as:

$$J_{\text{MT}}(\theta) = \mathbb{E}_{\{\tau_i\}_{i=1}^{G}\sim\pi_{\text{old}}}\Bigg\{\frac{1}{G}\sum_{i=1}^{G}\frac{1}{L_i}\sum_{j=1}^{L_i}\frac{1}{|o_{i,j}|}\sum_{t=1}^{|o_{i,j}|}\Bigg[\min\Big(\rho_{i,j,t}\hat{A}_i, \text{clip}\Big(\rho_{i,j,t}, 1-\epsilon, 1+\epsilon\Big)\hat{A}_i\Big) - \beta D_{\text{KL}}[\pi_\theta(\cdot|s_{i,j})||\pi_{\text{ref}}(\cdot|s_{i,j})]\Bigg]\Bigg\} \tag{8}$$

where L_i is the number of turns in trajectory i, and the advantage $\hat{A}_i$ is computed using the group-relative statistics of the trajectory rewards $r(\tau_i)$. This allows the sparse, episodic reward to guide the optimization of every step in the trajectory.

Table 4. Hyperparameter settings across different training stages.

Hyperparameter	**SFT**	**STRL**	**MTRL**
Total Training Tokens	0.07B	0.14B	1.41B
Training Samples (Images)	12K	24K	1.3M
Trainable Components	Full	Full	Full
Global Batch Size	512	1272	128
Optimizer	AdamW	AdamW	AdamW
LR Warmup Steps	15	0	0
Maximum Learning Rate	8.0×10^{-6}	1.0×10^{-6}	5.0×10^{-6}
Minimum Learning Rate	1.0×10^{-6}	1.0×10^{-6}	5.0×10^{-6}
Group Size (G)	-	4	4
KL Coefficient (β)	-	0.01	0.01
Clipping Parameter (ϵ)	-	0.2	0.2

D. Training Details

D.1. Base Model

To enable robust multi-modal reasoning and high-level planning, our system is built upon a pre-trained Vision-Language Model (VLM). Specifically, we initialize our system with the open-source **OpenHA** model [49], which is derived from Qwen2-VL-7B [44]. We selected this model for its demonstrated proficiency in Minecraft-specific visual grounding and its strong instruction-following capabilities within the domain.

D.2. Training Implementation

As detailed in section 3, our training pipeline comprises three progressive stages: Cold-Start Supervised Fine-Tuning (SFT), Single-Turn Reinforcement Learning (STRL), and Multi-Turn Reinforcement Learning (MTRL). To ensure experimental consistency and reproducibility, all models were trained on a cluster of 8 NVIDIA A800-SXM4-80GB GPUs, using Python 3.10 and CUDA 12.6.

The specific hyperparameter configurations for each stage are summarized in Table 4. We maintain consistent hyperparameter settings for the SFT process across both the initial Stage 1 and the cold-start phases to isolate the impact of training data and objectives

E. Case Studies

To qualitatively demonstrate the efficacy of dynamic action space selection, we analyze CrossAgent's behavior across three representative tasks: `Kill Sheep` (Kill Entities), `Chop Tree` (Mine Blocks), and `Craft Enchanting Table` (Craft Items). These cases illustrate how the agent adapts its interface strategy to meet the distinct demands of different task phases. Figure 7 illustrates the probability density curves of different action spaces selected by CrossAgent with respect to task completion progress.

Kill Sheep As illustrated in the `Kill Sheep` task proceeds through three distinct stages: **1) Navigation**: Rotating the camera and traversing the terrain to locate a target. **2) Approaching**: Locking focus on the sheep and closing the distance. **3) Interaction**: Executing attacks to deal damage. During the **Navigation Stage**, CrossAgent predominantly employs the coarse yet efficient Motion Action Space to scan large areas rapidly. Once the sheep enters the field of view, the agent transitions to the **Approaching Stage**, characterized by a balanced mix of action spaces: Motion Actions facilitate fast movement, Grounding Actions provide continuous visual tracking cues, and Raw Actions enable fine-grained adjustments to navigate around obstacles. Finally, in the **Interaction Stage**, sustained Grounding Actions ensure precise targeting of the moving entity, while Raw Actions are leveraged to execute high-frequency attack commands efficiently. This progression demonstrates CrossAgent's ability to switch contextually: prioritizing search efficiency initially, then blending modalities for pursuit, and finally optimizing for targeting precision.

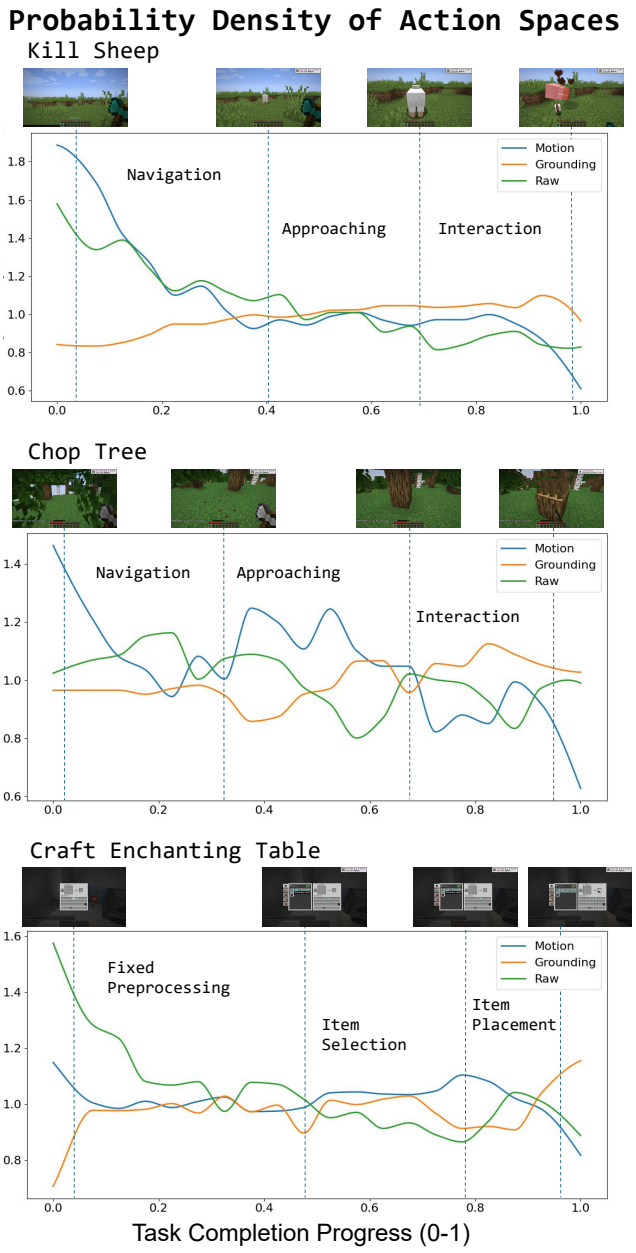

Figure 6. **Case Study: Action distribution during the `Kill Sheep`, `Chop Tree` and `Craft Enchanting` task.** The density curves of each tasks, aggregated over 20 episodes, of different action spaces (Motion, Grounding, Raw) across different task phases. The dynamic shifts in distribution demonstrate the model's in-context adaptive strategy.

Chop Tree The `Chop Tree` task shares the Navigation-Approaching-Interaction structure with `Kill Sheep` but exhibits distinct characteristics. Trees are static targets, and the agent typically spawns in forested areas where targets are abundant. Consequently, the task requires less precise tracking than hunting moving entities. Motion Actions dominate the navigation and approaching phases. Unlike the `Kill Sheep` task, where the agent must constantly adjust its bearing to chase a target, the static nature of trees allows the agent to rely heavily on the simple 'Walk Forward' primitive within the Motion Space. In the interaction phase, Grounding Actions are engaged to accurately target the specific wood block, ensuring successful harvesting.

Craft Enchanting Table Crafting tasks involve complex, multi-step GUI interactions. We categorize the workflow into three phases: **1) Fixed Preprocess**: Opening the recipe book and toggling the 'Only Show Craftable Items' filter. Since these steps are invariant across tasks, Raw Actions dominate this phase, leveraging their precision and stability to execute rote "muscle memory" sequences efficiently. **2) Item Selection**: Locating and selecting the target item within the grid. Here, the identification capability of the Grounding Space becomes crucial for visually recognizing the correct icon. Simultaneously, the Motion Space aids in smooth cursor traversal. **3) Item Placement**: Retrieving the crafted item from the result slot. The Grounding Space is again pivotal, helping the model accurately coordinate the cursor between the result slot and the inventory, mitigating the risk of mis-clicks.

Figure 6 further illustrates some key frames and corresponding outputs from the CrossAgent rollout. These examples demonstrate CrossAgent's ability to adaptively select the suitable action space in context, leading to more precise and efficient task execution.

The qualitative rollouts corroborate our statistical findings, offering a concrete window into the agent's decision-making process. The generated trajectories reveal that CrossAgent's switching mechanism is semantically coherent rather than stochastic. For instance, the seamless transition from high-level *Motion* primitives for efficient exploration to precise *Grounding* or *Raw* commands for interaction closely mirrors human behavioral patterns. This qualitative evidence, combined with the quantitative performance metrics, strongly validates that CrossAgent has successfully internalized a hierarchical understanding of open-world tasks. It demonstrates that the model does not merely memorize trajectories but learns to autonomously decompose complex objectives into optimal sequences of heterogeneous actions, thereby achieving both robustness and efficiency.

Kill Sheep

Motion:move forward

Grounding:Approach sheep(780,661)

Action: move(0, 0) and press() and click(left)"

Motion: no_op

Chop Tree

Action: move(-2, 4) and press() and click(left)

Grounding: Attack oak_log(938,581)

Action: move(0, 0) and press(w) and click(left)"

Action: move(0, 0) and press(w) and click(left)

Craft Enchanting Table

Action: move(-16, -9) and press()

Motion: cursor move left and down

Grounding: move_camera enchanting_table(338,464)

Motion: cursor move down

Figure 7. **Example Rollouts of `Kill Sheep`, `Chop Tree` and `Craft Enchanting` task.** The prefixes 'Motion:', 'Grounding:', and 'Action:' denote actions from the Motion Space, Grounding Space, and Raw Space, respectively.