HiSTEVE: Harnessing Human-Interface VLA Agents for Open-World Games

Anonymous authors
Paper under double-blind review

Abstract

Building virtual embodied agents that operate in open-ended games like human players is a long-standing goal requiring both long-horizon autonomy and reliable native-interface control. As a procedurally generated open world, Minecraft provides a natural testbed for studying the integration of these capabilities. Existing agents typically provide either native visuomotor control without long-horizon autonomy or high-level planning executed through environment APIs or predefined skills. To bridge this gap, we propose HiSTEVE, a hierarchical VLA agent with a capability-aware agent harness. At the orchestration level, we equip the harness with structured memory for tracking progress and a VLM planner for selecting subtasks within the VLA's capabilities. To reliably determine subtask completion, we further introduce a policy-coupled task completion state detector that conditions on VLA policy context to signal candidate completion. At the control level, a single VLA executes all subtasks with vision-only perception and native keyboard-and-mouse actions. To mitigate interference between heterogeneous GUI and non-GUI interactions, we introduce regime-specific RMSNorm branches and separate mouse-movement token embeddings. We also train the VLA through world knowledge injection, imitation learning, and reinforcement learning with environment feedback to improve reliability and recovery. HiSTEVE achieves state-of-the-art atomic-task performance, scoring 85.0% on MineStudio Bench and a 37.4% average success rate on OpenHA. To our knowledge, HiSTEVE is the first agent to complete long-horizon tasks such as crafting an iron pickaxe from scratch under the same interaction constraints as human players.

1 Introduction

A long-standing goal in embodied intelligence is to build virtual embodied agents (Fung et al., 2025) that can operate in open-ended games under the same interaction constraints as human players. Achieving this goal requires two complementary capabilities: long-horizon autonomy to maintain goals and adapt plans; and reliable native-interface control to translate first-person visual observations into keyboard-and-mouse actions. As a procedurally generated open world, Minecraft provides a natural testbed for studying these capabilities by combining partially observable, compositional tasks with diverse visuomotor interactions (Fan et al., 2022; Zheng et al., 2023). A generally capable Minecraft agent must therefore integrate long-horizon planning with native-interface execution in a closed loop, continually updating its decisions from visual feedback.

Existing Minecraft agents have primarily emphasized one of these two capabilities. On the one hand, visuomotor policies and Vision-Language-Action (VLA) models address native-interface control by directly mapping raw first-person observations to keyboard-and-mouse actions (Baker et al., 2022; Lifshitz et al., 2023; Li et al., 2025). However, these models typically lack mechanisms for persistently tracking task state and explicitly reasoning over task dependencies, limiting their long-horizon autonomy. On the other hand, agents driven by large language models (LLMs) or vision-language models (VLMs) support long-horizon reasoning through task decomposition planning (Wang et al., 2023a; Zhu et al., 2023; Wang et al., 2024b; Li et al., 2024). However, many of these methods still rely on environment APIs, predefined skills, or functional actions for low-level execution, which may be unavailable in most games. Although recent work has begun to combine high-level planning with learned policies (Wang et al., 2025b; Park et al., 2025), integrating task-state tracking, completion verification, and failure recovery remains an open challenge.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Satisfying both requirements introduces coupled challenges at the planning and action timescales. At the planning timescale, long-horizon autonomy requires tracking the objective, progress, and execution history across subtasks, while ensuring that plans are consistent with Minecraft mechanics and remain within the lower-level policy's capabilities (Wang et al., 2023b). The agent must also determine when to terminate the current subtask: terminating too early leaves the subtask unfinished, whereas terminating too late wastes actions and may even undo existing progress. At the action timescale, native-interface control requires a single VLA policy to operate reliably across heterogeneous task distributions. In particular, non-GUI control and GUI interaction differ in both visual distributions and mouse-action semantics: mouse movement rotates the camera in the former but displaces the cursor in the latter, making naive joint training susceptible to negative transfer (He et al., 2026; Yang et al., 2026). Each VLA invocation must also be reliable, since even small per-subtask failure risks accumulate over repeated invocations in long-horizon tasks.

To address these challenges, we propose HiSTEVE, a hierarchical VLA agent that combines a capability-aware harness for long-horizon autonomy and reliable native-interface control. At the slower planning timescale, we design the harness with a structured memory of task progress and a VLM planner that selects the next atomic subtask. To close the planning–execution loop, we introduce a policy-coupled task completion state detector that conditions on VLA policy context to signal candidate completion. The harness then verifies the task state and decides whether to advance, recover, or replan. At the faster action timescale, the VLA predicts native keyboard-and-mouse actions from the subtask instruction and first-person visual observation. To mitigate negative transfer across heterogeneous interaction regimes, we introduce regime-specific RMSNorm branches and separate mouse-movement token embeddings for non-GUI and GUI tasks. Following world knowledge injection and imitation learning, we further finetune the VLA through reinforcement learning (RL) to improve recovery when execution deviates from demonstration trajectories.

We comprehensively evaluate HiSTEVE on atomic and long-horizon benchmarks. Compared with the strongest baseline in our main comparison, HiSTEVE improves average success by 20.0 percentage points on MineStudio Bench and 0.6 points on OpenHA, with family-level gains of up to 28.6 points on LongBench. The largest long-horizon improvements occur in tasks requiring multi-stage resource production or the reuse of information gathered during exploration. Ablations reveal that stronger atomic execution alone does not yield comparable gains in long-horizon performance. With the VLA held fixed, the full harness more than doubles goal success relative to fixed-horizon handoff, highlighting the importance of organizing local execution around a persistent task state. The learned completion detector also outperforms prompted general-purpose VLMs, providing more accurate completion judgments to support this execution loop.

The main contributions of this work are summarized as follows:

- We present HiSTEVE, a hierarchical VLA agent with a capability-aware agent harness, enabling HiSTEVE to maintain goals and adapt plans throughout long-horizon execution.
- We enable reliable native-interface control through regime-specific RMSNorm branches, separate mouse-movement token embeddings, and a three-stage training strategy.
- HiSTEVE achieves state-of-the-art atomic-task performance and is the first agent to complete long-horizon tasks such as crafting an iron pickaxe from scratch like a human player.

2 Related Work

Agent Harnesses for Execution Foundation-model agents often use an agent harness, an external orchestration layer that supports interaction across model calls (Sumers et al., 2023; Wang et al., 2025a; Lin et al., 2026). Early language agents connect language models to external knowledge and tools and alternate between reasoning and action (Nakano et al., 2021; Karpas et al., 2022; Yao et al., 2022). Later systems added memory, feedback-based reflection, and search-based evaluation, allowing the harness to retain experience and refine subsequent decisions (Zhou et al., 2023; Packer et al., 2023; Shinn et al., 2023; Wang et al., 2024c; Xu et al., 2026). In long-horizon embodied tasks, agent harnesses coordinate high-level planning and low-level execution across multiple steps (Huang et al., 2022a; Wang et al., 2024a; Shah et al., 2025). Such harnesses use environmental observations, feasibility assessments, and execution outcomes to update subsequent decisions, closing the planning–execution loop (Huang et al., 2022b; Guo et al., 2024; Zhi et al., 2025).

VLA Models and Systems VLA models map visual observations and language instructions to embodied actions (Ma et al., 2026; Brohan et al., 2023). Early language-conditioned policies learned this mapping directly from multi-task demonstrations (Shridhar et al., 2022; Jang et al., 2022; Brohan et al., 2022). As model capacity and robot datasets grew, these policies generalized to a broader range of tasks, environments, and robot embodiments (Brohan et al., 2023; O'Neill et al., 2024; Team et al., 2024; Kim et al., 2024; Black et al., 2026; Pertsch et al., 2025). To support long-horizon planning, hierarchical systems use a semantic planner to generate policy-compatible subtasks for execution by a low-level policy (Ma et al., 2026; Ahn et al., 2022; Driess et al., 2023; Shi et al., 2025). More recent approaches allow intermediate reasoning or predictive representations to directly guide low-level control (Belkhale et al., 2024; Yang et al., 2025; Lin et al., 2025; Zhang et al., 2026b).

Game-Playing Agents in Virtual Worlds Games have long served as testbeds for sequential decision making, with research progressing from search-based systems to deep RL agents (Campbell et al., 2002; Mnih et al., 2015; Silver et al., 2016). More recently, pretrained models have been used to build agents that follow instructions and reason over observations (Ma et al., 2024; Zhou et al., 2026; Tan et al., 2025). Researchers have also trained unified policies on multi-game data, but such policies often sacrifice control precision and long-horizon reliability (Raad et al., 2024; Magne et al., 2026; Zhang et al., 2026a). Alongside efforts toward cross-game generalization, Minecraft has emerged as a focused testbed for studying embodied interaction within an open world (Guss et al., 2019; Fan et al., 2022). Research in Minecraft includes visuomotor policies learned from interaction trajectories (Baker et al., 2022; Lifshitz et al., 2023; Li et al., 2025), as well as LLM- or VLM-based agents for exploration and skill acquisition (Zhu et al., 2023; Wang et al., 2023a; 2024b).

3 METHODOLOGY

3.1 CAPABILITY-AWARE HARNESS FOR LONG-HORIZON AUTONOMY

We propose a capability-aware agent harness that coordinates a VLM planner and a VLA policy for long-horizon autonomy. Specifically, we integrate capability-constrained planning, evidence-driven handoff and recovery, and structured memory (Appendix A).

3.1.1 CAPABILITY-CONSTRAINED PLANNING

Long-horizon Minecraft tasks cannot be reliably solved through a single static plan because the environment is partially observable, and local execution failures may change the feasibility of subsequent subtasks. Planning within the harness is reference-free: no predefined subtask sequence or expert task route is provided, and the planner selects the next atomic subtask online from the final objective and current execution context. We constrain the planner to select and instantiate a parameterized atomic subtask schema from a set restricted to the VLA's execution capabilities. Each schema specifies its applicability, semantic parameters, visually observable progress and completion conditions, recognizable failure conditions, and execution budget.

Let G denote the long-horizon objective, O_j the current and recent visual observations at the j-th decision step, and h_j the structured planning state, which summarizes verified task progress, unresolved requirements, and previous execution outcomes. At each decision step, the harness evaluates schema applicability against h_j to construct the currently applicable schema set $\mathcal{S}_j$, while R_j contains contextual memory retrieved according to the current goal and planning state. Given the planning context and admissible schema set constructed by the harness, the VLM planner selects and instantiates the atomic subtask as

$$u_j = \mathcal{P}_{\mathrm{VLM}}\left(G, O_j, h_j, \mathcal{S}_j, R_j\right), \tag{1}$$

The structured output u_j specifies a semantic objective rather than a callable skill or scripted action sequence. The harness converts it into a fixed natural-language instruction I_j, which the same VLA executes through native keyboard-and-mouse actions within a bounded invocation.

3.1.2 EVIDENCE-DRIVEN HANDOFF AND RECOVERY

To address the handoff-timing problem between high-level planning and VLA control discussed above, we introduce an evidence-driven handoff-and-recovery protocol. During each bounded VLA

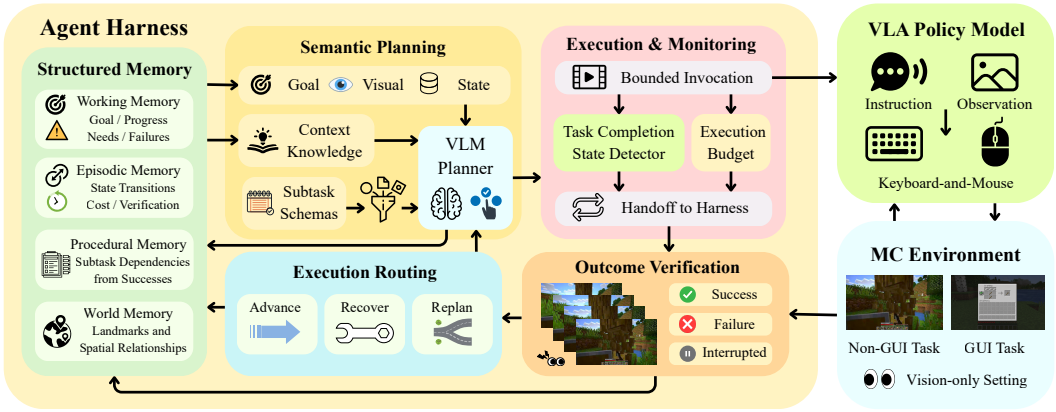

Figure 1: System overview of HiSTEVE, illustrating the agent harness mechanism.

invocation, the task-completion state detector estimates a completion probability for the current instruction I_j. When this probability exceeds a confidence threshold, the current invocation ends and returns control to the harness, and exhaustion of the execution budget triggers the same handoff. Our mechanism differs from failure detection: the detector signals candidate completion rather than execution failure, while the harness verifies progress and completion conditions using current and recent visual observations. Based on this reassessment, the execution outcome v_j is recorded as $\mathrm{success}$, $\mathrm{failure}$, or $\mathrm{interrupted}$, where $\mathrm{interrupted}$ indicates that control has been returned before either success or failure can be verified. Following verification, the harness advances if the completion conditions are satisfied and attempts local recovery if the subtask remains incomplete but recoverable. The harness replans from the updated state if recovery is infeasible or repeatedly fails, the current subtask becomes inapplicable, or the visual state changes substantially.

3.1.3 Structured Memory

Long-horizon execution requires persistent task state, but retaining the full history increases context length and accumulates unverified intermediate information. The harness therefore maintains a compact structured memory centered on verified state transitions. The memory system comprises four stores with distinct roles. Working memory corresponds to the current planning state h_j and records the active objective and atomic subtask, verified task progress, unresolved requirements, and recent failures or lack of progress. Episodic memory records the planning states before and after each VLA invocation, together with the selected subtask, the harness-verified outcome, execution cost, and diagnostic information. Procedural memory stores reusable dependencies among parameterized subtasks derived from verified successful executions. World memory stores landmarks and spatial relations inferred from visual observations to support subsequent planning within the current world. At each high-level decision step, the working memory h_j is used directly, whereas task-relevant entries are retrieved from the remaining memory categories to form the R_j.

3.2 VLA Policy for Native Visuomotor Control

The lower-level VLA policy maps a subtask instruction with first-person visual observations to native keyboard-and-mouse actions. We describe the policy architecture and action representation, followed by a three-stage training strategy for improving execution reliability.

3.2.1 VLA Policy Architecture

Architecture. We initialize the policy from Qwen2.5-VL-7B (Bai et al., 2025). At control step t, the policy takes I_j and the current first-person observation o_t as input and autoregressively generates the tokenized action $\mathbf{a}_t$. After executing $\mathbf{a}_t$, the agent receives a new observation and repeats this process at 20 Hz. Non-GUI and GUI interactions require different visuomotor skills, which can lead to conflicting optimization signals when all parameters are shared. Gradient analysis of a fully shared baseline shows that cross-regime gradients are less aligned than within-regime gradients,

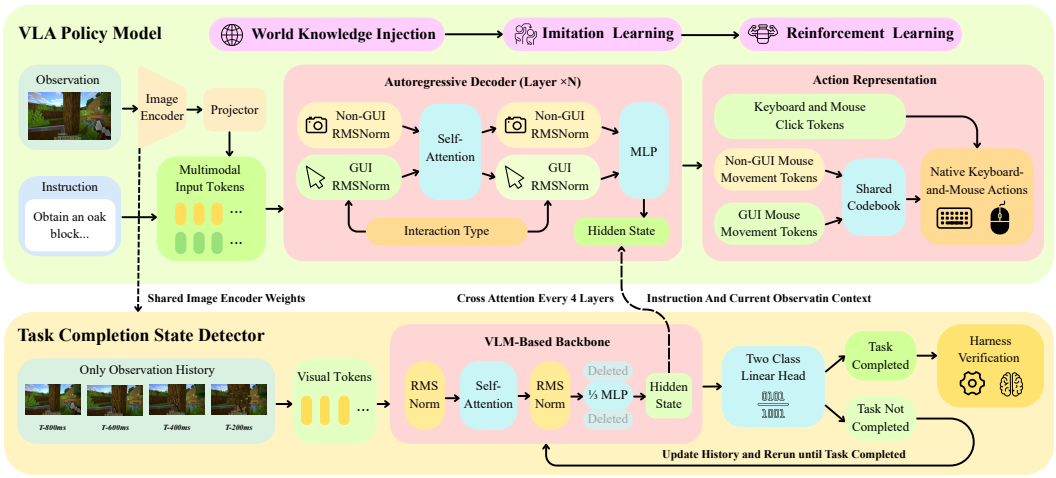

Figure 2: Architecture of the VLA policy model and task completion state detector.

with the two RMSNorm modules exhibiting the strongest misalignment among shared components. The complete measurement protocol and results analysis are provided in Appendix B.1.

Based on this analysis, we introduce Regime-Specific RMSNorm (RS-Norm), a novel lightweight normalization mechanism for heterogeneous interaction regimes. Specifically, for both the pre-attention and pre-MLP RMSNorms in each decoder layer, we maintain two learnable scale vectors, one for non-GUI interactions and one for GUI interactions. The appropriate scale vector is selected according to the interaction regime of the current subtask. For interaction type $c \in \{\mathrm{NG}, \mathrm{GUI}\}$, the RMSNorm at decoder layer ℓ and module $q \in \{\mathrm{attn}, \mathrm{mlp}\}$ is defined as

$$\mathrm{RMSNorm}^{(c)}_{\ell,q}(\mathbf{h}) = \boldsymbol{\gamma}^{(c)}_{\ell,q} \odot \frac{\mathbf{h}}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} h_i^2 + \epsilon}}, \tag{2}$$

where $\boldsymbol{\gamma}^{(c)}_{\ell,q} \in \mathbb{R}^d$ is the interaction-specific scale vector, while all other parameters remain shared. For a model with L decoder layers and hidden size d, this design adds only $2Ld$ parameters relative to the original backbone, resulting in minimal parameter overhead. RS-Norm can also be extended to other heterogeneous task combinations, such as vision-language understanding and visuomotor control, as well as multimodal understanding and generation.

Action Representation. We represent Minecraft's native keyboard-and-mouse actions as discrete action tokens, allowing supervised action prediction to use the same next-token prediction objective as language modeling. Following the VPT-style action space (Baker et al., 2022), we organize each single-step action into multiple categorical action groups. Keyboard and click inputs are represented as discrete choices within predefined action groups, while horizontal and vertical mouse displacements are μ-law transformed and quantized into the same number of bins.

However, mouse displacement controls camera rotation in non-GUI interactions but cursor movement in GUI interactions, so sharing a single action token set creates an ambiguity. We therefore assign disjoint mouse-displacement token sets: the original mouse tokens are retained for non-GUI interactions, while a separate subset is allocated to GUI interactions. We further insert an interaction-type token at the beginning of the sequence to indicate whether the current interaction is non-GUI or GUI. Because both sets of mouse-displacement tokens decode back to the same physical action codebook, the branching occurs solely at the representation level. We refer to this regime-separated mouse-movement token representation as **Dual-Mouse**. (Appendix B.3)

3.2.2 Three-Stage Policy Training

Because per-subtask failures compound over repeated VLA invocations, reliable long-horizon execution requires high reliability. We therefore train the VLA in three stages: world knowledge injection, imitation learning, and reinforcement learning with environment feedback.

Supervised Initialization. Before reinforcement learning, we use two supervised stages to establish the semantic knowledge and control capabilities required for reliable native-interface execution. World knowledge injection equips the policy with Minecraft concepts and their visual grounding, while imitation learning teaches it to map subtask instructions and first-person observations to native keyboard-and-mouse actions. Further details on data sources, sample construction and curation, training objectives, and optimization settings are provided in Appendix C.

Reinforcement Learning After imitation learning, the policy remains less reliable in GUI interactions, because precise cursor positioning and multi-step operations make GUI execution more sensitive to small action errors. We address this imbalance through environment-verified branch comparisons at both trajectory and state-matched segment levels. For each initial condition s_0, we independently sample K trajectories using a behavior policy π_b. Branch outcomes can be scored using either VLMs judge or direct environment feedback. Considering cost and efficiency, we adopt the latter for trainer-side reward construction and state matching. Each trajectory score R_i combines certified completion, maximum verified progress, execution cost, and failure penalties. The resulting trajectories form a same-start branch group, whose scores yield the group-relative advantage:

$$\mathcal{B}(s_0) = \{\tau_i\}_{i=1}^{K}, \quad \tau_i \sim \pi_b(\cdot \mid s_0), \qquad A_i = \frac{R_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}} + \varepsilon_A}, \tag{3}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ denote the within-group score mean and standard deviation.

We optimize informative branch groups with clipped GRPO (Shao et al., 2024) updates, separating detached behavior-policy importance correction from the current-policy update ratio. However, trajectory-level advantages can penalize correct prefixes in failed rollouts. We therefore construct local preference pairs from successful and failed branches, prioritizing the same rollout origin and matching structured signatures of inventory, GUI, cursor, execution stage, and environment events. The resulting DPO-style pairwise loss targets divergent action segments, using the same completion certificates as group-relative learning. To preserve non-GUI capabilities, we interleave branch updates with world-action replay and regularize against a fixed reference policy π_{ref}:

$$\mathcal{L}_{\mathrm{post}} = \mathcal{L}_{\mathrm{branch}} + \lambda_{\mathrm{world}}\mathcal{L}_{\mathrm{world}} + \lambda_{\mathrm{ref}} D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\mathrm{ref}}\right), \tag{4}$$

where $\mathcal{L}_{\mathrm{branch}}$ is the group-relative loss for informative branch groups or the pairwise loss for matched segments, depending on the available comparisons, and $\mathcal{L}_{\mathrm{world}}$ is the world-action replay loss. Reward construction and optimization details are provided in Appendix C.3.

3.3 Task Completion State Detector

We introduce a novel policy-coupled task completion state detector to support the harness in determining when to pause the VLA invocation. To reduce system overhead, the detector shares the VLA image encoder and additionally conditions on VLA policy context for completion prediction, with Qwen2.5-VL-3B serving as its backbone. At each query, the detector uses the shared image encoder and its self-attention stream to process historical frames sampled at 800, 600, 400, and $200\,\mathrm{ms}$ before the query, while the VLA processes the current frame and subtask instruction. The final-layer hidden states of the input prefix preceding the first action token form the policy context H_t^{π}, which is injected through cross-attention after every fourth detector decoder layer. Let $Z_t^{(\ell)}$ denote the detector hidden states at layer ℓ. We apply query-normalized residual cross-attention as

$$\widetilde{Z}_t^{(\ell)} = Z_t^{(\ell)} + \mathrm{MHA}_\ell\left(\mathrm{LN}_\ell(Z_t^{(\ell)}), H_t^{\pi}, H_t^{\pi}\right), \tag{5}$$

where each cross-attention module has independent projection parameters. To further reduce detector overhead, we use Wanda-inspired activation–weight channel scores to prune the intermediate width of each feed-forward network to approximately one third (Sun et al., 2024). For classification, we map the detector's final-token hidden state to two logits through a linear classification head and apply a softmax to obtain the completion probability p_t. We train the detector with the VLA frozen on completion labels derived from VPT human gameplay trajectories (Baker et al., 2022). At deployment, the detector runs before action decoding and raises a candidate completion signal to the harness only when p_t exceeds τ for two consecutive queries. Details of dataset construction, channel pruning and training are provided in Appendix D.

Table 1: Comparison on MineStudio Bench, OpenHA, and LongBench. MineStudio reports strict pass@1, OpenHA reports family-wise episode success and macro average, and LongBench reports final-goal success. R&C, E&M, Rec., and Open denote the four LongBench task families. The best and second-best results are highlighted in **bold** and underline, respectively.

	MineStudio Bench				OpenHA				LongBench			
Method	**Mine**	**Kill**	**Craft**	**Smelt**	**Emb.**	**GUI**	**Combat**	**Avg.**	**R&C**	**E&M**	**Rec.**	**Open**
VPT	20.0	13.3	0.0	6.7	10.1	0.7	3.6	4.8	0.0	1.4	0.0	0.0
STEVE-1	20.0	13.3	0.0	6.7	8.4	0.0	4.9	4.4	0.0	0.7	0.0	0.0
ROCKET-1	33.3	40.0	0.0	6.7	19.2	0.0	29.8	16.3	1.4	2.9	0.7	0.0
JARVIS-VLA	66.7	60.0	26.7	60.0	31.0	25.3	18.3	24.9	7.1	5.7	1.4	0.7
LatentHA	46.7	33.3	6.7	20.0	27.3	3.5	8.2	13.0	2.1	2.9	0.7	0.0
MotionHA	53.3	33.3	6.7	20.0	31.6	0.0	9.1	13.6	2.9	3.6	0.7	0.0
GroundingHA	60.0	53.3	13.3	33.3	39.7	3.7	28.2	23.9	5.7	4.3	1.4	0.0
TextVLA	46.7	46.7	13.3	40.0	23.9	14.0	27.1	21.7	7.1	5.0	1.4	0.7
OpenHA Text/CoA	73.3	73.3	40.0	73.3	37.0	**33.3**	**40.0**	36.8	14.3	7.1	2.9	1.4
HiSTEVE (Ours)	**93.3**	**86.7**	**66.7**	**93.3**	**46.0**	27.2	39.0	**37.4**	**42.9**	**28.6**	**14.3**	**14.3**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We instantiate HiSTEVE with GPT-4o (Hurst et al., 2024) as the upper-level planner and Qwen2.5-VL-7B (Bai et al., 2025) as the low-level VLA. We evaluate atomic execution on MineStudio Bench (Zheng et al., 2023) and OpenHA (Wang et al., 2025b), and long-horizon agency on LongBench. Ablation scores are averaged over repeated independent evaluations for each fixed checkpoint. MineStudio contains 20 tasks with three trials each, OpenHA reports family-wise episode success over Embodied, GUI, and Combat, and LongBench contains 14 goal-only tasks with 49 hidden diagnostic nodes. We compare against prior native-control agents, including VPT (Baker et al., 2022), STEVE-1 (Lifshitz et al., 2023), ROCKET-1 (Cai et al., 2025), and JARVIS-VLA (Li et al., 2025), as well as OpenHA variants—LatentHA, MotionHA, GroundingHA, TextVLA, and Text/CoA (Wang et al., 2025b). Complete evaluation protocols are provided in Appendix E.

4.2 MAIN RESULTS

As shown in Table 1, we evaluate HiSTEVE across atomic and long-horizon tasks.

Atomic task execution. On MineStudio Bench, HiSTEVE achieves strict pass@1 scores of 93.3%, 86.7%, 66.7%, and 93.3% on Mine, Kill, Craft, and Smelt, respectively, corresponding to an overall average of 85.0%. Compared with OpenHA Text/CoA, HiSTEVE improves the four-family average from 65.0% to 85.0%, with gains of 20.0, 13.4, 26.7, and 20.0 percentage points on Mine, Kill, Craft, and Smelt, respectively. The largest absolute gain occurs on Craft, although Craft remains the weakest MineStudio category, indicating that multi-step GUI manipulation remains more challenging than first-person mining and combat.

On OpenHA, HiSTEVE achieves a 37.4% average success rate, compared with 36.8% for OpenHA Text/CoA. The gain is concentrated in Embodied tasks, where success increases from 37.0% to 46.0%, while GUI and Combat remain slightly below the corresponding baseline scores. Our trajectory analysis shows that failed GUI episodes can reach and interact with the correct interface but break down during multi-step item placement, intermediate-state correction, or result retrieval, indicating that robust multi-step GUI execution remains a key bottleneck.

Long-horizon execution. LongBench evaluates whether atomic native-control capabilities remain effective when they must be repeatedly composed over extended interaction. Unlike atomic benchmarks, success depends not only on executing individual subtasks, but also on maintaining task state across invocations and adapting subsequent decisions to observed outcomes. We evaluate Resource

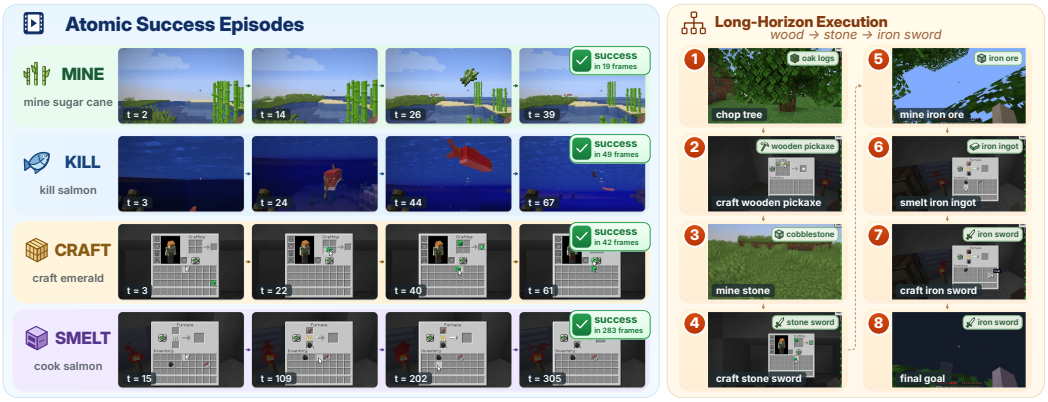

Figure 3: Qualitative examples of HiSTEVE. The left panel shows successful atomic-task executions, while the right panel illustrates multi-stage long-horizon execution.

& Crafting, Exploration & Memory, Recovery, and Open-ended tasks using final-goal success as the primary metric, with hidden diagnostic nodes used only for progress analysis.

HiSTEVE achieves final-goal success rates of 42.9%, 28.6%, 14.3%, and 14.3% on Resource & Crafting, Exploration & Memory, Recovery, and Open-ended tasks, respectively, with the overall Goal SR in Table 3 computed as the macro-average over the four task families. Relative to OpenHA Text/CoA, these correspond to improvements of 28.6, 21.5, 11.4, and 12.9 percentage points. The largest gains appear on Resource & Crafting and Exploration & Memory, where successful execution requires repeated composition of atomic skills and persistent task-state tracking. Recovery and Open-ended tasks remain substantially harder, indicating that robust recovery and weakly structured objectives remain important challenges at longer horizons. Figure 3 complements these results with qualitative examples of atomic and long-horizon execution.

4.3 Atomic Task Completion Detection Evaluation

We evaluate completion detection on a class-balanced validation split of 4,674 samples drawn from VPT trajectories (Baker et al., 2022) across four task families: mining, killing, crafting, and smelting (details see Appendix D.1). As baselines, we adopt a random classifier, the open-source VLMs Qwen3.5 at 9B and 35B parameters (Team, 2026), and the closed-source GPT-4o and GPT-5.5 (Hurst et al., 2024; Singh et al., 2025). These general-purpose models are not trained for task-completion detection, and we prompt them with the same history frames, current frame, and subtask instruction and ask for a binary completion decision. We report accuracy, precision, recall, and F1 for the completion class, with higher values indicating better performance.

Table 2: Atomic-subtask completion detection. All models use the same temporal sampling.

Method	#Params	Acc ↑	Prec ↑	Rec ↑	F1 ↑
Random	–	0.500	0.500	0.500	0.500
Qwen3.5	9B	0.500	0.504	0.030	0.057
Qwen3.5	35B	0.502	0.530	0.042	0.078
GPT-4o	–	0.698	0.708	0.673	0.690
GPT-5.5	–	0.780	0.791	0.761	0.776
Ours	1.4B	**0.863**	**0.861**	**0.865**	**0.863**

As shown in Table 2, our specialized detector consistently outperforms the prompted general-purpose VLM baselines. Qwen3.5 strongly favors the incomplete class, with recalls of only 0.030 and 0.042 at 9B and 35B parameters, respectively. GPT-4o and GPT-5.5 improve F1 to 0.690 and 0.776. In contrast, our detector reaches 0.863 in both accuracy and F1, with precision and recall of 0.861 and 0.865, providing a more accurate and balanced candidate-completion signal for the planning–control interface. The downstream effect of state-dependent handoff within the complete agent is evaluated in Section 4.4, while detector architectural ablations are reported in Appendix F.

Table 3: Ablation of staged VLA training, regime-specific representation, and agent harness design. For MineStudio, World averages Mine and Kill, GUI averages Craft and Smelt. For OpenHA, World averages Embodied and Combat, GUI denotes the GUI family. LongBench reports final-goal success (Goal SR) and hidden diagnostic progress (Progress); Full H. denotes the complete agent harness.

	Method components					MineStudio			OpenHA			LongBench	
Method	**WKI**	**RL**	**RS**	**DM**	**Full H.**	**World**	**GUI**	**All**	**World**	**GUI**	**Avg.**	**Goal SR**	**Progress**
Staged policy training													
Imitation learning only	–	–	✓	✓	–	54.7	46.2	50.4	36.4	15.7	29.5	7.1	11.9
+ world knowledge injection	✓	–	✓	✓	–	71.4	58.9	65.1	41.8	22.4	35.3	14.3	20.7
+ environment-feedback RL	✓	✓	✓	✓	–	89.7	71.6	80.7	43.1	26.8	37.7	14.3	21.9
Regime-specific representation													
Shared representation	✓	✓	–	–	–	47.8	34.5	41.1	34.2	11.5	26.6	3.6	8.6
+ RS-Norm	✓	✓	✓	–	–	62.9	51.8	57.3	36.9	19.2	31.0	7.1	13.9
+ Dual-Mouse	✓	✓	–	✓	–	69.5	54.3	61.9	39.7	18.4	32.6	10.7	16.7
+ RS-Norm + Dual-Mouse	✓	✓	✓	✓	–	89.7	71.6	80.7	43.1	26.8	37.7	14.3	21.9
Harness orchestration													
Full VLA + fixed-horizon handoff	✓	✓	✓	✓	–	**90.5**	71.9	81.2	**43.4**	26.5	**37.8**	10.7	17.9
Full HiSTEVE	✓	✓	✓	✓	✓	90.0	**80.0**	**85.0**	42.5	**27.2**	37.4	**25.0**	**28.6**

4.4 Ablation Study

Table 3 isolates staged policy training, regime-specific representation, and harness orchestration.

Effect of regime-specific representation. RS-Norm and Dual-Mouse provide complementary gains: from a shared-representation baseline of 41.1% on MineStudio All, they individually improve performance to 57.3% and 61.9%, and jointly reach 80.7%. Together, they also improve OpenHA Avg. from 26.6% to 37.7% and LongBench Goal SR from 3.6% to 14.3%. Concurrent gains on both MineStudio World and GUI suggest that regime-specific feature scaling and mouse-action separation mitigate complementary interference between the two interaction regimes.

Effect of staged policy training. World knowledge injection substantially improves the lower-level executor, increasing MineStudio All from 50.4% to 65.1% and OpenHA Avg. from 29.5% to 35.3%, while LongBench Goal SR increases from 7.1% to 14.3%. Environment-feedback RL further raises MineStudio All to 80.7% and OpenHA Avg. to 37.7%, but leaves LongBench Goal SR unchanged at 14.3%. This separation indicates that post-training strengthens local execution reliability, while improving the executor alone is insufficient to solve long-horizon coordination.

Effect of the agent harness. With the lower-level VLA fixed, this ablation evaluates system-level execution, where handoff and verification can still affect atomic success. The complete harness changes atomic performance modestly, from 81.2% to 85.0% on MineStudio All and from 37.8% to 37.4% on OpenHA Avg. In contrast, LongBench Goal SR increases from 10.7% to 25.0%, with diagnostic progress rising from 17.9% to 28.6%. These gains highlight the role of capability-constrained planning and execution handoff and verification in organizing VLA invocations.

5 Conclusion

We present HiSTEVE, a hierarchical VLA agent combining a capability-aware agent harness for long-horizon autonomy with native-interface control. The harness enables long-horizon autonomy through capability-constrained planning, evidence-driven handoff and recovery, and structured memory. Regime-specific representations and three-stage training enable a single VLA to reliably handle both non-GUI and GUI interactions. HiSTEVE achieves state-of-the-art performance on MineStudio Bench (85.0%) and OpenHA (37.4%) and is the first agent to craft an iron pickaxe from scratch using only human-interface. Our work shows that regime-specific representations support reliable control across heterogeneous tasks, while reliable long-horizon execution further requires a harness that combines local control with state-aware orchestration. These results represent a step toward virtual embodied agents that operate in open-ended games human players.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation, refinement of methodological and experimental descriptions, and the preparation and revision of some scientific figures. These tools were not used to generate the training or evaluation data used in this work, produce experimental results, or independently determine the research conclusions. The authors manually reviewed and revised all AI-assisted text, technical descriptions and figures, and verified them against the system implementation, experimental records, and original sources. The authors take full responsibility for the final content of the paper and all material produced with AI assistance.

ETHICS STATEMENT

This study is conducted solely in the Minecraft virtual environment and involves no real-world deployment, recruitment of human participants, or collection of new personal data. Training uses publicly released third-party datasets, including human gameplay trajectories. We follow their applicable licenses and terms of use and do not claim ownership of third-party content. Potential risks include biases or unreliable behavior arising from pretrained models and human gameplay data. We therefore evaluate the system only in controlled research environments.

REPRODUCIBILITY STATEMENT

We provide the model architecture, training objectives, data composition, and evaluation protocols in the main paper and appendix. All reported MineStudio and OpenHA results are associated with fixed checkpoints and complete evaluation denominators. The long-horizon evaluations use fixed task definitions, execution budgets, and validation rules. Additional implementation and optimization details are documented in the appendix and supplementary material.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL https://arxiv.org/abs/2502.13923.

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

Suneel Belkhale, Tianli Ding, Ted Xiao, Pierre Sermanet, Quon Vuong, Jonathan Tompson, Yevgen Chebotar, Debidatta Dwibedi, and Dorsa Sadigh. Rt-h: Action hierarchies using language. *arXiv preprint arXiv:2403.01823*, 2024.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Shaofei Cai, Zihao Wang, Kewei Lian, Zhancun Mu, Xiaojian Ma, Anji Liu, and Yitao Liang. Rocket-1: Mastering open-world interaction with visual-temporal context prompting. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12122–12131. IEEE, 2025.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134 (1-2):57–83, 2002.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

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

Pascale Fung, Yoram Bachrach, Asli Celikyilmaz, Kamalika Chaudhuri, Delong Chen, Willy Chung, Emmanuel Dupoux, Hongyu Gong, Hervé Jégou, Alessandro Lazaric, et al. Embodied ai agents: Modeling the world. *arXiv preprint arXiv:2506.22355*, 2025.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. Doremi: Grounding language model by detecting and recovering from plan-execution misalignment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12124–12131. IEEE, 2024.

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

Kaichen He, Zihao Wang, Muyao Li, Anji Liu, and Yitao Liang. Training one model to master cross-level agentic actions via reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 724–734, 2026.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022a.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022b.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *conference on Robot Learning*, pp. 991–1002. PMLR, 2022.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE transactions on big data*, 7(3):535–547, 2019.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Muyao Li, Zihao Wang, Kaichen He, Xiaojian Ma, and Yitao Liang. Jarvis-vla: Post-training large-scale vision language models to play visual games with keyboards and mouse. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 17878–17899, 2025.

Zaijing Li, Yuquan Xie, Rui Shao, Gongwei Chen, Dongmei Jiang, and Liqiang Nie. Optimus-1: Hybrid multimodal memory empowered agents excel in long-horizon tasks. *Advances in neural information processing systems*, 37:49881–49913, 2024.

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

Fanqi Lin, Ruiqian Nai, Yingdong Hu, Jiacheng You, Junming Zhao, and Yang Gao. Onetwovla: A unified vision-language-action model with adaptive reasoning. *arXiv preprint arXiv:2505.11917*, 2025.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, et al. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Advances in neural information processing systems*, 37:133386–133442, 2024.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision–language–action models for embodied ai. *IEEE Transactions on Neural Networks and Learning Systems*, 2026.

Loïc Magne, Anas Awadalla, Guanzhi Wang, Yinzhen Xu, Joshua Belofsky, Fengyuan Hu, Joohwan Kim, Ludwig Schmidt, Georgia Gkioxari, Jan Kautz, et al. Nitrogen: An open foundation model for generalist gaming agents. *arXiv preprint arXiv:2601.02427*, 2026.

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

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Junyeong Park, Junmo Cho, and Sungjin Ahn. Mrsteve: Instruction-following agents in minecraft with what-where-when memory. In *International Conference on Learning Representations*, volume 2025, pp. 16922–16960, 2025.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PmLR, 2021.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Rutav Shah, Albert Yu, Yifeng Zhu, Yuke Zhu, and Roberto Martín-Martín. Bumble: Unifying reasoning and acting with vision-language models for building-wide mobile manipulation. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13337–13345. IEEE, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on robot learning*, pp. 894–906. pmlr, 2022.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*, 2023.

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024.

Weihao Tan, Xiangyang Li, Yunhao Fang, Heyuan Yao, Shi Yan, Hao Luo, Tenglong Ao, Huihui Li, Hongbin Ren, Bairen Yi, et al. Lumine: An open recipe for building generalist agents in 3d open worlds. *arXiv preprint arXiv:2511.08892*, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

Huaxiaoyue Wang, Kushal Kedia, Juntao Ren, Rahma Abdullah, Atiksh Bhardwaj, Angela Chao, Kelly Y Chen, Nathaniel Chin, Prithwish Dan, Xinyi Fan, et al. Mosaic: Modular foundation models for assistive and interactive cooking. *arXiv preprint arXiv:2402.18796*, 2024a.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025a.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Shawn Ma, and Yitao Liang. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 36:34153–34189, 2023b.

Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, et al. Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(3):1894–1907, 2024b.

Zihao Wang, Muyao Li, Kaichen He, Xiangyu Wang, Zhancun Mu, Anji Liu, and Yitao Liang. Openha: A series of open-source hierarchical agentic models in minecraft. *arXiv preprint arXiv:2509.13347*, 2025b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024c.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Longrong Yang, Zhixiong Zeng, Yufeng Zhong, Jing Huang, Liming Zheng, Lei Chen, Haibo Qiu, Zequn Qin, Lin Ma, and Xi Li. Omniactor: A generalist gui and embodied agent for 2d&3d worlds. In *International Conference on Learning Representations*, volume 2026, pp. 97248–97269, 2026.

Yi Yang, Jiaxuan Sun, Siqi Kou, Yihan Wang, and Zhijie Deng. Lohovla: A unified vision-language-action model for long-horizon embodied tasks. *arXiv preprint arXiv:2506.00411*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Kuan Zhang, Dongchen Liu, Qiyue Zhao, Tianyu Xin, Yue Su, Haisheng Wang, Han Yin, Hongbo Ma, Peize Li, Tianjun Gu, et al. Towards generalist game players: An investigation of foundation models in the game multiverse. *arXiv preprint arXiv:2605.09965*, 2026a.

Yi Zhang, Yinda Chen, Che Liu, Zeyuan Ding, Jin Xu, Shilong Zou, Junwei Liao, Jiayu Hu, Xiancong Ren, Xiaopeng Zhang, et al. Pelican-unify 1.0: A unified embodied intelligence model for understanding, reasoning, imagination and action. *arXiv preprint arXiv:2605.15153*, 2026b.

Xinyue Zheng, Haowei Lin, Kaichen He, Zihao Wang, Zilong Zheng, and Yitao Liang. Mcu: An evaluation framework for open-ended game agents. *arXiv preprint arXiv:2310.08367*, 2023.

Peiyuan Zhi, Zhiyuan Zhang, Yu Zhao, Muzhi Han, Zeyu Zhang, Zhitian Li, Ziyuan Jiao, Baoxiong Jia, and Siyuan Huang. Closed-loop open-vocabulary mobile manipulation with gpt-4v. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4761–4767. IEEE, 2025.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Zheyuan Zhou, Liang Du, Zixun Sun, Xiaoyu Zhou, Ruimin Ye, Qihao Chen, Yinda Chen, and Lemiao Qiu. Main-vla: Modeling abstraction of intention and environment for vision-language-action models. *arXiv preprint arXiv:2602.02212*, 2026.

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

A AGENT HARNESS DETAILS

This section provides additional implementation details for the agent harness introduced in Section 3. We describe the runtime interaction between the VLM planner and the native-control VLA, the capability-constrained atomic interface, evidence-driven verification and recovery, and the structured memory used during long-horizon execution.

A.1 HARNESS RUNTIME

HiSTEVE executes a long-horizon objective through repeated bounded interactions between a high-level semantic planner and a continuously acting VLA policy. At high-level decision step j, the harness maintains a structured planning state

$$h_j = (G, p_j, r_j, v_{<j}), \tag{6}$$

where G denotes the final task objective, p_j denotes visually verified progress, r_j denotes unresolved requirements, and $v_{<j}$ summarizes previous execution outcomes.

Given h_j, the harness determines which parameterized atomic subtask schemas are currently applicable and constructs the admissible set

$$\mathcal{S}_j = \{s \in \mathcal{S} \mid \text{Applicable}(s, h_j) = 1\}. \tag{7}$$

Task-relevant information retrieved from episodic, procedural, and world memory is denoted by R_j. The planner then selects and instantiates the next atomic subtask:

$$u_j = \mathcal{P}_{\text{VLM}}(G, O_j, h_j, \mathcal{S}_j, R_j), \tag{8}$$

where O_j denotes the current and recent visual observations.

The planner output u_j specifies a semantic objective rather than an executable Minecraft function or scripted action sequence. The harness converts it into a fixed natural-language instruction I_j, which is executed by the same VLA during one bounded invocation. At control step t within the invocation, the VLA predicts

$$\mathbf{a}_t \sim \pi_\theta(\cdot \mid o_t, I_j), \tag{9}$$

where o_t is the current first-person observation and $\mathbf{a}_t$ is a native keyboard-and-mouse action.

The instruction I_j remains fixed throughout the invocation, while the visual observation changes after every environment step. This separation yields two interacting timescales: the VLA performs continuous control at the action timescale, whereas the harness updates the task state only at semantic execution boundaries.

Control is returned to the harness when either the execution budget is exhausted or the completion detector produces a candidate completion signal. Let p_t denote the detector probability that the current subtask has completed. A detector-triggered handoff is produced when

$$p_t > \tau \quad \wedge \quad p_{t-1} > \tau, \tag{10}$$

where τ is the detector operating threshold. Requiring two consecutive positive predictions reduces sensitivity to isolated detector errors.

Importantly, a handoff does not imply successful task completion. It only indicates that the current bounded invocation should stop and return control to the harness for verification.

A.2 CAPABILITY-CONSTRAINED ATOMIC INTERFACE

Direct free-form decomposition can produce subtasks that are too abstract for reliable execution by the VLA, or conversely, subtasks that are so narrow that they require unnecessarily frequent high-level decisions. HiSTEVE therefore exposes the lower-level VLA to the planner through a parameterized atomic interface.

Each atomic schema $s \in \mathcal{S}$ is represented as

$$s = (\mathcal{A}_s, \mathcal{P}_s, \mathcal{C}_s, \mathcal{F}_s, B_s), \tag{11}$$

where $\mathcal{A}_s$ denotes the semantic arguments of the subtask, $\mathcal{P}_s$ its applicability conditions, $\mathcal{C}_s$ its visually observable completion criteria, $\mathcal{F}_s$ recognizable failure conditions, and B_s the execution budget.

The schema library defines the capability boundary presented to the planner. It does not contain scripted implementations of the corresponding subtasks. Once a schema is instantiated, all physical interactions required to realize the semantic objective are produced by the VLA through native keyboard-and-mouse actions.

For count-based objectives, the harness does not issue an indefinitely running instruction such as "collect N items." Instead, it repeatedly invokes bounded acquisition subtasks. After invocation j, the remaining requirement is updated according to the visually verified progress:

$$r_{j+1} = \max\left(0, r_j - \Delta_j\right), \tag{12}$$

where Δ_j denotes newly verified progress during the current invocation.

This decomposition prevents already acquired resources from being repeatedly counted as new progress and avoids continuing a stale local instruction after the corresponding requirement has already been satisfied.

A.3 Evidence-Driven Verification and Recovery

HiSTEVE explicitly separates execution handoff from task verification. A detector-triggered handoff only indicates a candidate semantic boundary. After control returns, the harness reassesses the current and recent visual observations and evaluates the completion and failure predicates associated with the active subtask.

The resulting execution outcome is represented as

$$v_j \in \{\text{success}, \text{failure}, \text{interrupted}\}, \tag{13}$$

where interrupted indicates that control has returned to the harness but neither success nor failure can yet be visually verified.

The updated planning state is then

$$h_{j+1} = \mathcal{U}\left(h_j, u_j, v_j, O_{j+1}\right), \tag{14}$$

where $\mathcal{U}$ denotes the harness state-update operation and O_{j+1} contains the visual evidence available after the invocation.

During deployment, the planner and VLA interact with the environment only through first-person observations and native keyboard-and-mouse actions. Simulator-side state used during training for reward construction or trajectory certification is not provided to the deployed planner or VLA.

For example, resource acquisition is accepted only after the corresponding inventory change is visually confirmed. Likewise, crafting or smelting is not considered complete merely because an intermediate GUI state is reached; the resulting item must be visually observed in the player inventory.

After verification, the harness follows three possible transitions:

$$\text{Next}(v_j) = \begin{cases} \text{advance}, & \text{if the completion condition is verified,} \\ \text{recover}, & \text{if the subtask remains locally recoverable,} \\ \text{replan}, & \text{otherwise.} \end{cases} \tag{15}$$

Recovery is used for local execution deviations that do not invalidate the high-level objective, such as temporary lack of progress, inconsistent GUI state, target loss, or a produced item that has not yet been collected. If recovery is infeasible or repeatedly fails, the current subtask becomes inapplicable, or the observed state changes substantially, the harness invokes the planner again from the updated task state.

A.4 STRUCTURED MEMORY

Long-horizon execution requires persistent task state, but retaining the complete raw interaction history rapidly increases context length and preserves many transient or unverified intermediate states. HiSTEVE instead maintains a compact memory centered on verified subtask-level state transitions.

The memory system consists of four complementary stores:

- **Working memory** records the current objective, active subtask, verified progress, unresolved requirements, and recent failure or no-progress states.
- **Episodic memory** records the state transition associated with each bounded VLA invocation, together with the selected subtask, harness-verified outcome, execution cost, and diagnostic information.
- **Procedural memory** stores reusable dependencies among parameterized subtasks extracted from verified successful executions.
- **World memory** stores visually inferred landmarks and spatial relations that may support later planning within the current world.

The episodic record generated by bounded invocation j is represented as

$$m_j = (h_j, u_j, v_j, h_{j+1}, c_j)\,, \tag{16}$$

where c_j contains execution metadata such as execution cost and diagnostic information.

At each high-level decision step, the current working memory h_j is directly included in the planning context. Task-relevant entries from the other memory stores are retrieved according to

$$R_j = \mathrm{Retrieve}\,(G, h_j, \mathcal{M}_{\mathrm{epi}}, \mathcal{M}_{\mathrm{proc}}, \mathcal{M}_{\mathrm{world}})\,. \tag{17}$$

Only visually verified outcomes are committed as successful task progress. Procedural memory is updated only from verified successful executions.

To reduce memorization of specific evaluation trajectories, procedural memory does not retain absolute coordinates or fixed camera orientations. Memory accumulated during the current episode remains available for online planning, whereas episodic records from previous runs of the same evaluation seed are excluded from retrieval. This prevents direct replay of routes discovered during earlier evaluations of the same seed.

A.5 MATCHED HARNESS COMPARISON

To isolate the contribution of agentic orchestration from lower-level policy quality, we compare harness variants while holding the lower-level VLA and upper-level semantic model fixed.

The fixed-horizon baseline executes the same atomic instruction for a predetermined control horizon before returning control to the upper level. Full HiSTEVE instead uses state-dependent handoff, visual verification, structured memory update, local recovery, and replanning.

Let π_θ denote the shared VLA policy and $\mathcal{H}$ the high-level harness. The comparison can therefore be written as

$$\pi_{\mathrm{agent}} = \mathcal{H}\,(\pi_\theta)\,, \tag{18}$$

where π_θ is held fixed and only $\mathcal{H}$ changes between the compared systems.

This controlled comparison produces only a small change in atomic-task performance: MineStudio All changes from 81.2% to 85.0%, while OpenHA Avg. changes from 37.8% to 37.4%. In contrast, LongBench goal success increases from 10.7% to 25.0%, and diagnostic progress increases from 17.9% to 28.6%.

The contrast between nearly unchanged atomic performance and substantially improved long-horizon performance indicates that the primary gain arises from how the same controller is scheduled, monitored, verified, and recovered over extended interaction rather than from a stronger underlying VLA.

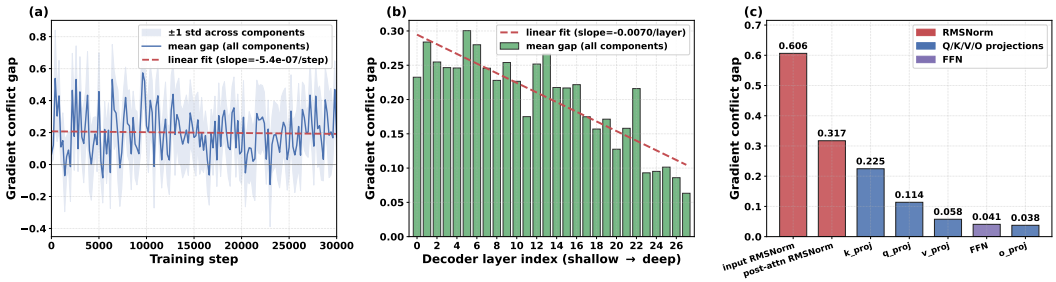

Figure 4: Overview of the cross-regime gradient-conflict analysis. (a) Mean conflict gap over all layers and components throughout training; the shaded region denotes ± 1 standard deviation across components, and the dashed line is a linear fit. (b) Mean conflict gap by decoder depth, averaged over training steps and components. (c) Mean conflict gap by shared component, averaged over training steps and decoder layers. All panels are computed from the same set of gradient probes.

B VLA Architecture and Action Representation Details

B.1 Cross-Regime Gradient Conflict

This subsection provides the complete experimental setup and results for the cross-regime gradient-conflict analysis summarized in the main text. We train a fully shared baseline without regime-specific parameter branches, keeping the vision encoder frozen while jointly optimizing the remaining decoder parameters on the union of non-GUI and GUI data.

Gradient Conflict Gap Measurements. During the $30{,}000$-step training run, we establish a probe point every 200 optimizer steps, including at initialization. At each probe point, we hold the current parameters fixed and compute gradients separately on pure non-GUI and pure GUI mini-batches without applying an optimizer update. At each of the 28 decoder layers, we extract the gradients of the four attention projections, the two RMSNorm modules, and the feed-forward network, and flatten the gradient of each component into a single vector. Each gradient estimate is accumulated over 8 mini-batches to reduce batch-level noise. Data augmentation is disabled during probing, and samples are drawn from a fixed probe split excluded from parameter optimization.

Let g and h denote the gradients of the same model component obtained from GUI and non-GUI data, respectively. Their cross-regime cosine similarity is

$$c_{\text{cross}} = \cos(g, h) = \frac{g^\top h}{\|g\|_2 \|h\|_2}. \tag{19}$$

The cross-regime cosine alone does not distinguish regime differences from mini-batch variability. We therefore divide the probe data for each regime into two disjoint subsets, A and B, and compute the corresponding within-regime similarities

$$c_{\text{GUI}} = \cos(g_A, g_B), \qquad c_{\text{NG}} = \cos(h_A, h_B). \tag{20}$$

We define the conflict gap as the difference between the mean within-regime similarity and the cross-regime similarity:

$$\Delta = \frac{1}{2}\left(c_{\text{GUI}} + c_{\text{NG}}\right) - c_{\text{cross}}. \tag{21}$$

A positive Δ therefore indicates that cross-regime gradients are less aligned than gradients from the same regime, relative to the within-regime reference induced by mini-batch variability.

Conflict throughout Training. As shown in Figure 4(a), the conflict gap averaged over all layers and components has an overall mean of 0.1997. The fitted slope is -5.4×10^{-7} per optimizer step, corresponding to a total change of approximately -0.016 over $30{,}000$ steps, indicating no substantial reduction in cross-regime misalignment during training. At initialization, Δ is approximately 0.06. The gap increases during the first several hundred optimizer steps and persists throughout the remainder of training, suggesting that it is not restricted to a short transient near initialization.

Conflict across Decoder Depth. Figure 4(b) shows that the conflict gap decreases with decoder depth when averaged over training steps and components, with a fitted slope of -0.007 per layer. The early layers L0–L8 have a mean gap of 0.257, approximately twice the mean of 0.124 in the deeper layers L19–L27, while the mean gap remains positive at every decoder layer. This result shows that cross-regime misalignment is stronger in the early layers but is not confined to a small subset of the decoder. The pattern is consistent with the two regimes placing different demands on lower-level visuomotor features, although the analysis alone does not establish a causal source.

Conflict across Shared Components. As shown in Figure 4(c), the two RMSNorm modules exhibit the highest mean conflict gaps: 0.606 for the input RMSNorm and 0.317 for the post-attention RMSNorm. Among the remaining components, the K projection has a gap of 0.225, while the Q, V, and O projections and the feed-forward network have gaps of 0.114, 0.058, 0.038, and 0.041, respectively. Because the learnable parameters of RMSNorm are channel-wise scale vectors, the concentration of conflict in these modules suggests that regime-specific feature scaling provides a parameter-efficient location for specialization. This observation motivates the regime-specific RMSNorm branches introduced in the main method.

B.2 Backbone and Regime Routing

The VLA policy is instantiated from Qwen2.5-VL-7B (Bai et al., 2025), retaining its original vision encoder, multimodal projector, and autoregressive decoder. The current visual observation and atomic-subtask instruction are serialized using the native Qwen2.5-VL multimodal template. Action symbols are assigned to reserved vocabulary entries that do not overlap with natural-language tokens, allowing actions to be generated through the original autoregressive output head.

Each atomic subtask is associated with an interaction regime $c_t \in \{\mathrm{NG}, \mathrm{GUI}\}$. This regime serves as an explicit routing variable that selects the corresponding RMSNorm branch and mouse-displacement token set. The same regime is serialized as the interaction-type token after the beginning-of-action token. Apart from the regime-specific RMSNorm parameters and mouse-token embeddings, the remaining backbone parameters are shared across the two regimes.

B.3 Action Representation

HiSTEVE represents native keyboard-and-mouse inputs as discrete action tokens, without using symbolic actions or arbitrary text input. Following the mouse encoding used in prior Minecraft policies (Baker et al., 2022; Li et al., 2025), we apply μ-law companding to the horizontal and vertical mouse displacements independently and quantize each transformed axis into 21 bins.

Non-GUI and GUI interactions use disjoint mouse-token sets. Mouse displacement controls camera orientation in the non-GUI regime and cursor position in the GUI regime. The two regimes share the same numerical displacement bins, but each bin is assigned a different vocabulary entry with an independently learned embedding. Table 4 summarizes the resulting native action space.

The native controls in Table 4 are serialized into a fixed-order action-token sequence. The beginning-of-action, end-of-action, and interaction-regime tokens define the sequence structure and do not introduce additional native input events. For each control step, the sequence is

$$\mathbf{a}_t = \Big[\langle \mathrm{BOA} \rangle, \langle c_t \rangle, g_t^{(1)}, \ldots, g_t^{(M)}, m_{x,b_{t,x}}^{(c_t)}, m_{y,b_{t,y}}^{(c_t)}, \langle \mathrm{EOA} \rangle\Big], \qquad c_t \in \{\mathrm{NG}, \mathrm{GUI}\}, \tag{22}$$

where $g_t^{(1)}, \ldots, g_t^{(M)}$ denote the discrete key and button groups in their fixed serialization order, and $b_{t,x}, b_{t,y} \in \{1, \ldots, 21\}$ are the quantized horizontal and vertical mouse-displacement indices.

C VLA Training Strategy Details

C.1 World Knowledge Injection

To reliably map subtasks to low-level actions, the policy model must acquire knowledge of Minecraft's entities, blocks and crafting mechanics. We therefore inject world knowledge by supervised fine-tuning. We adopt the world knowledge and alignment data released with JARVIS-VLA

Table 4: Native Minecraft action space used by HiSTEVE. Non-GUI and GUI mouse movements share the same displacement codebook but are represented by disjoint token sets.

Index	Action	Human Input	Description
1	Forward	key `W`	Move forward.
2	Back	key `S`	Move backward.
3	Left	key `A`	Strafe left.
4	Right	key `D`	Strafe right.
5	Jump	key `Space`	Jump or swim upward.
6	Sneak	key `Left Shift`	Sneak and move cautiously.
7	Sprint	key `Left Ctrl`	Sprint in the current movement direction.
8	Attack	`Left Mouse Button`	Attack an entity or break a block.
9	Use	`Right Mouse Button`	Interact with an object.
10	Inventory	key `E`	Open or close the inventory.
11	Hotbar: [1–9]	keys `1--9`	Select the corresponding hotbar slot.
12	Non-GUI Mouse X	move `Mouse X`	Rotate the camera horizontally.
13	Non-GUI Mouse Y	move `Mouse Y`	Rotate the camera vertically.
14	GUI Mouse X	move `Mouse X`	Move the GUI cursor horizontally.
15	GUI Mouse Y	move `Mouse Y`	Move the GUI cursor vertically.

(Li et al., 2025), and deduplicate it to remove near-duplicate samples. We freeze the vision encoder, and fully finetune the language decoder, so that it acquires game world knowledge and aligns it with first-person visual observations.

Training Sample Construction. The world knowledge data contain three supervision formats: text-only knowledge question answering, captioning of in-game screenshots, and visual question answering over the screenshots. Table 5 illustrates the model input and supervised target.

Data Curation. Before training, we apply format-specific normalization, deduplication, and quality filtering; for knowledge-QA targets, we capitalize the initial letter and add missing sentence-final punctuation. For exact deduplication, we lowercase and whitespace-normalize questions and retain the longest target for each normalized question. For knowledge QA, we use FAISS (Johnson et al., 2019) to retrieve candidate neighbors from `all-MiniLM-L6-v2` sentence embeddings (Reimers & Gurevych, 2019; Wang et al., 2020) and merge candidate pairs with cosine similarity of at least 0.95 via union–find; for VQA, both text similarity from the same sentence encoder and image similarity computed from CLIP ViT-B/32 embeddings (Radford et al., 2021) must be at least 0.85. Finally, we remove QA samples with empty questions or targets, or targets containing at most five words excluding punctuation.

Training Objective. Let x denote either a text-only context or an image–text context, and let $y = (y_1, \ldots, y_{|y|})$ denote the corresponding target answer or caption. World knowledge injection uses an autoregressive cross-entropy objective evaluated only over the target-output tokens:

$$\mathcal{L}_{\mathrm{WK}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\mathrm{WK}}}\left[\sum_{j=1}^{|y|}\log p_\theta\left(y_j \mid x, y_{<j}\right)\right], \tag{23}$$

where $\mathcal{D}_{\mathrm{WK}}$ denotes the mixed world knowledge dataset after filtering and deduplication. Input-prompt and visual tokens are excluded from the loss.

C.2 Imitation Learning

Starting from the world knowledge checkpoint, we train the policy using VPT gameplay data (Baker et al., 2022), trajectories released with JARVIS-VLA (Li et al., 2025), and script-generated trajectories that supplement crafting and smelting recipes. After deduplication, we resample trajectories by task to balance the task distribution. Each training example consists of the subtask instruction, the single-frame observation, and the corresponding keyboard-and-mouse action. At this stage, we duplicate each RMSNorm scale vector to initialize the GUI and non-GUI branches, freeze the vision encoder, and finetune all remaining policy parameters.

Table 5: Representative examples of the three data formats used for world knowledge injection.

Text QA Example

Question: How do you use enchantments to enhance items and equipment in Minecraft?

Target answer: By placing the item in an Enchantment Table to apply random enchantments, or by using an Anvil to combine items or apply enchanted books.

Caption Example



Instruction: Please provide a caption for this Minecraft screenshot.

Target caption: A zombie stands centrally in front of the player in a cave. Stone walls surround the player and the zombie. A pool of lava is visible in the background of the cave.

Visual Question Answering Example



Question: What type of biome is visible in the image?

Target answer: The biome visible in the image is a Desert biome.

Training Sample Construction. Starting from the world knowledge checkpoint, we divide demonstration trajectories into atomic-task clips and construct each training example from a single-frame visual observation, a subtask instruction, an interaction regime, and a target action sequence. Table 6 shows the corresponding formats for non-GUI and GUI training examples.

Data Curation. To improve recipe coverage, we first add 10 programmatically generated training trajectories for each `craft_item` or `smelt_item` recipe absent from the collected trajectories. We then resample the augmented dataset at the atomic-task-clip level to obtain a 1:1 GUI-to-non-GUI ratio, while ensuring that each designated major task family (e.g., `kill_entity` and `smelt_item`) accounts for at least 10% of all training samples.

Training Objective. Let o_t denote the current visual observation, u_t the subtask instruction, $c_t \in \{\mathrm{NG}, \mathrm{GUI}\}$ the interaction regime, and $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,|\mathbf{a}_t|})$ the target action sequence. Imitation

Table 6: Representative imitation-learning examples from the non-GUI and GUI interaction regimes. Each example contains a single-frame observation, an atomic-subtask instruction, and its aligned target action sequence.

Non-GUI Interaction Example



Subtask instruction: Collect the pumpkin by holding left button.

Target action sequence:

$$\mathbf{a}_t = \left[\langle \mathrm{BOA} \rangle, \langle \mathrm{NG} \rangle, \mathit{Attack}, m^{(\mathrm{NG})}_{x,b_{t,x}}, m^{(\mathrm{NG})}_{y,b_{t,y}}, \langle \mathrm{EOA} \rangle\right].$$

GUI Interaction Example



Subtask instruction: Craft the fishing rod at the crafting table.

Target action sequence:

$$\mathbf{a}_t = \left[\langle \mathrm{BOA} \rangle, \langle \mathrm{GUI} \rangle, \mathit{Use}, m^{(\mathrm{GUI})}_{x,b_{t,x}}, m^{(\mathrm{GUI})}_{y,b_{t,y}}, \langle \mathrm{EOA} \rangle\right].$$

learning uses an autoregressive cross-entropy objective evaluated only over the target action tokens:

$$\mathcal{L}_{\mathrm{IL}}(\theta) = -\mathbb{E}_{(o_t,u_t,c_t,\mathbf{a}_t)\sim\mathcal{D}_{\mathrm{IL}}}\left[\sum_{k=1}^{|\mathbf{a}_t|} \log p_\theta\left(a_{t,k} \mid \mathbf{a}_{t,<k}, o_t, u_t, c_t\right)\right], \tag{24}$$

where c_t selects the corresponding RMSNorm branch and mouse-displacement token set, while the interaction-type identifier token remains part of $\mathbf{a}_t$ and participates in the training loss.

C.3 Reinforcement Learning

Rollout Construction. For each same-start branch group in Equation 3, the initial condition fixes the task, world seed, initial inventory, and GUI configuration. Each branch is executed in a separate Minecraft process to prevent cross-branch state contamination. We record visual observations, action tokens, behavior-policy log-probabilities, parsed environment actions, termination reasons, and trainer-side environment evidence. For local comparisons, we prioritize candidate segments from the same branch group whose structured state signatures match at the beginning of the segment.

Environment-Based Rewards. Post-training determines atomic-subtask completion from environment state changes rather than model-reported success. For an item-acquisition task, the completion certificate for target item g is

$$C_t^g = \mathbf{1}\left[N_t(g) - N_0(g) \geq \delta_g\right], \tag{25}$$

where $N_t(g)$ is the inventory count at step t and δ_g is the required increase relative to the initial state. Thus, breaking a target block without collecting the drop, or producing an item in a GUI without transferring it to the player inventory, is not considered successful.

For GUI tasks, required-material coverage, correct slot occupancy, process initiation, output appearance, incorrect products, and material loss are summarized by a progress potential $\Phi(s)$. The step reward combines the change in progress potential with a terminal reward issued when the completion certificate is first satisfied:

$$r_t = \Phi(s_{t+1}) - \Phi(s_t) + R_T \mathbf{1}\left[C_{t+1}^g \wedge \neg C_t^g\right], \tag{26}$$

where R_T is the terminal success reward. Taking differences in Φ prevents the policy from repeatedly receiving the same progress reward by remaining at an already reached stage.

Branch Optimization. Let x_i denote the policy-visible visual–language condition associated with sample i, and let $\mathbf{a}_i$ denote the optimized action sequence. The behavior-policy correction weight and current-policy update ratio are

$$w_i = \frac{\pi_{\text{old}}(\mathbf{a}_i \mid x_i)}{\pi_b(\mathbf{a}_i \mid x_i)}, \qquad \rho_i = \frac{\pi_\theta(\mathbf{a}_i \mid x_i)}{\pi_{\text{old}}(\mathbf{a}_i \mid x_i)}. \tag{27}$$

We detach w_i from gradient computation and use ρ_i for the clipped group-relative update:

$$\mathcal{L}_{\text{GRPO}} = -\mathbb{E}_i\left[w_i \min\left(\rho_i A_i, \text{clip}\left(\rho_i, 1-\varepsilon_{\text{clip}}, 1+\varepsilon_{\text{clip}}\right) A_i\right)\right]. \tag{28}$$

For locally matched segments, let $(x^+, \mathbf{a}^+)$ and $(x^-, \mathbf{a}^-)$ denote the preferred and rejected segments from successful and failed branches, respectively. The corresponding pairwise objective is

$$\mathcal{L}_{\text{pair}} = -\log\sigma\left(\beta_{\text{pair}}\left[\log\frac{\pi_\theta(\mathbf{a}^+ \mid x^+)}{\pi_{\text{ref}}(\mathbf{a}^+ \mid x^+)} - \log\frac{\pi_\theta(\mathbf{a}^- \mid x^-)}{\pi_{\text{ref}}(\mathbf{a}^- \mid x^-)}\right]\right). \tag{29}$$

Equations 28 and 29 instantiate $\mathcal{L}_{\text{branch}}$ in Equation 4 according to the available comparison.

D Task Completion State Detector Details

D.1 Dataset Construction and Evaluation Split

We construct the completion detection dataset from VPT human gameplay trajectories, using offline event and inventory-state detection to identify the task category and completion time of each atomic-task segment. Source trajectories are partitioned before temporal observation windows are sampled, ensuring that no windows from the same trajectory appear across data splits.

For each atomic-task segment, let $t^\star$ denote the detected completion time. A temporal sample queried at time t receives the binary label

$$y_t = \mathbf{1}\left[t \geq t^\star\right]. \tag{30}$$

Samples before $t^\star$ are labeled incomplete, while samples at or after $t^\star$ are labeled complete. Incomplete samples are downsampled to balance the two classes. The reported validation set contains $4{,}674$ samples spanning mining, killing, crafting, and smelting. All compared models receive the same temporal observations and subtask instruction.

Cross-Attention Implementation. Cross-attention modules are inserted after detector decoder layers $\ell \in \{4, 8, 12, 16, 20, 24, 28, 32, 36\}$. Expanding the multi-head attention in Equation 5, the output of attention head r and the resulting residual update are

$$\begin{aligned} U_{\ell,r} &= \text{softmax}\left(\frac{\left(\text{LN}_\ell(Z_t^{(\ell)})W_{\ell,r}^Q\right)\left(H_t^\pi W_{\ell,r}^K\right)^\top}{\sqrt{d_h}}\right)\left(H_t^\pi W_{\ell,r}^V\right), \\ \widetilde{Z}_t^{(\ell)} &= Z_t^{(\ell)} + \text{Concat}_{r=1}^{n_h}\left(U_{\ell,r}\right) W_\ell^O, \end{aligned} \tag{31}$$

where n_h and d_h denote the number of attention heads and the dimension of each head, respectively.

Channel Pruning. We estimate feed-forward channel importance using intermediate activations collected from calibration samples and the corresponding down-projection weights. Let $A_\ell \in \mathbb{R}^{N\times d_{\text{ff}}}$ denote the intermediate activations at layer ℓ, and let $W_\ell^{\text{down}} \in \mathbb{R}^{d\times d_{\text{ff}}}$ denote its down-projection matrix. The channel score and retained set are defined as

$$s_{\ell,j} = \|A_{\ell,:,j}\|_2 \left\|W^{\text{down}}_{\ell,:,j}\right\|_2, \qquad \mathcal{I}_\ell = \text{TopK}\left(\{s_{\ell,j}\}_{j=1}^{d_{\text{ff}}}, d'_{\text{ff}}\right), \quad d'_{\text{ff}} \approx \frac{d_{\text{ff}}}{3}. \tag{32}$$

The same channel indices are retained in the output dimensions of the gate and up projections and the corresponding input dimensions of the down projection.

Training Objective. During training, we freeze the VLA and stop gradients through H_t^π. We optimize only the cross-attention modules, classification head, and LoRA adapters applied to the detector's self-attention Q/K/V/O projections. Let $y_t \in \{0,1\}$ denote the completion label and $p_{t,c}$ the predicted probability of class c. The detector is trained using label-smoothed two-class cross-entropy:

$$\mathcal{L}_{\text{det}} = -\frac{1}{|\mathcal{B}|}\sum_{t\in\mathcal{B}}\sum_{c\in\{0,1\}}\left[(1-\epsilon)\mathbf{1}\{c=y_t\} + \frac{\epsilon}{2}\right]\log p_{t,c}, \tag{33}$$

where ϵ is the label-smoothing coefficient and $\mathcal{B}$ denotes a training batch.

D.2 Deployment Decision Rule

The detector is evaluated before the VLA decodes the next action. Let p_t denote the predicted probability of the completion class. A single positive prediction does not immediately terminate the current invocation. Instead, the detector emits a candidate completion signal only if

$$p_{t-1} > \tau \quad \wedge \quad p_t > \tau. \tag{34}$$

The two-query confirmation suppresses isolated false-positive predictions. If the condition is not satisfied, the VLA proceeds to decode and execute the next native action.

A candidate completion signal only returns control to the harness. It is not treated as direct task-success evidence. The harness subsequently verifies the corresponding visual completion conditions before updating the planning state.

E Experimental Protocol Details

E.1 Atomic-Task Evaluation

We evaluates the lower-level VLA independently, while the harness ablation in Table 3 evaluates system-level execution on the same atomic tasks.

MineStudio Bench. MineStudio Bench contains 20 atomic tasks, evenly divided among Mine, Kill, Craft, and Smelt. Each task is evaluated with three independent trials. We report strict pass@1, which measures whether the policy succeeds on its first execution attempt. Mine and Kill primarily evaluate first-person world interaction, whereas Craft and Smelt additionally require GUI-mediated manipulation.

For category scores S_{Mine}, S_{Kill}, S_{Craft}, and S_{Smelt}, the overall score is

$$S_{\text{MineStudio}} = \frac{S_{\text{Mine}} + S_{\text{Kill}} + S_{\text{Craft}} + S_{\text{Smelt}}}{4}. \tag{35}$$

E.2 LongBench Benchmark and Evaluation

LongBench evaluates whether atomic native-control capabilities can be composed into persistent behavior over extended interaction. In contrast to atomic benchmarks, each LongBench episode provides only a final goal: the agent must determine the required intermediate objectives online from its visual observations and execution history. Neither a reference task decomposition nor the benchmark's diagnostic annotations are exposed to the agent.

Table 7: LongBench task suite. Each episode exposes only the final goal to the agent; the decomposition and diagnostic checkpoints are withheld during execution.

Family	Task	Primary long-horizon requirement
Resource & Crafting	`wooden_age`	Resource acquisition and tool progression
	`stone_age`	Multi-stage gathering and crafting
	`iron_tool`	Resource, furnace, and tool-production chain
	`glass_maker`	Gathering, smelting, and output retrieval
Exploration & Memory	`cave_survey`	Exploration and persistent environmental state
	`camp_return`	Landmark memory and return navigation
	`field_camp`	Exploration with stateful task continuation
	`resource_revisit`	Remembering and revisiting a prior location
Recovery	`tool_break_recovery`	Recovery after tool failure
	`lost_drop_recovery`	Recovery from lost task resources
	`fuel_alternative`	Substitution after unavailable intended resource
Open-ended	`cook_dinner`	Flexible multi-stage goal completion
	`light_up`	Open-ended resource acquisition and execution
	`survival_iron_ingot`	Extended survival and resource progression

Task suite. LongBench contains 14 goal-only tasks organized into four families that emphasize different long-horizon requirements: Resource & Crafting, Exploration & Memory, Recovery, and Open-ended interaction. Table 7 summarizes the task suite.

The four families serve complementary purposes. Resource & Crafting tasks require the agent to compose multiple native-control skills into a dependency chain whose final object is unavailable at initialization. Exploration & Memory tasks require information gathered earlier in the episode to affect later decisions. Recovery tasks introduce situations in which the current execution path can no longer be continued directly and therefore require local recovery or replanning. Open-ended tasks admit multiple valid strategies and provide less structure about the sequence of intermediate objectives.

Benchmark initialization. Eleven of the fourteen tasks start from an empty inventory in a natural survival world. Formal evaluation uses fixed environment seeds, while terrain, resource distribution, and naturally occurring entities remain part of the Minecraft environment.

The three Recovery tasks use controlled initial states to reproducibly instantiate low-frequency recovery scenarios involving tool breakage, alternative fuel, and missed resource pickup.

These states are instantiated only during reset; subsequent evaluation follows the same native visual and keyboard-and-mouse interface used throughout the paper.

Hidden diagnostic nodes. LongBench additionally defines 49 hidden diagnostic nodes across the 14 tasks. A diagnostic node denotes an interpretable intermediate state, such as obtaining a prerequisite resource, constructing an intermediate tool, reaching a relevant landmark, or entering a required processing stage.

These nodes are never included in the planner prompt and are not used to select the next subtask. They are recorded only by the evaluation procedure after execution and therefore provide a diagnostic view of trajectory progress without supplying privileged guidance to the agent.

For task i, let $\mathcal{D}_i$ denote its diagnostic-node set and $\mathcal{R}_i \subseteq \mathcal{D}_i$ the nodes reached during execution. We define diagnostic progress as

$$P_i = \frac{|\mathcal{R}_i|}{|\mathcal{D}_i|}. \tag{36}$$

Across an evaluation set of N instances, the aggregate progress metric is

$$\text{Progress} = \frac{\sum_{i=1}^{N} |\mathcal{R}_i|}{\sum_{i=1}^{N} |\mathcal{D}_i|}. \tag{37}$$

Diagnostic progress is not a success criterion. A trajectory may reach most intermediate diagnostic states but is still counted as unsuccessful if the final-goal predicate is not satisfied.

Evaluation interface. LongBench follows the same human-interface execution constraint used throughout HiSTEVE. At the beginning of each episode, the agent receives a natural-language final goal, while the deployed planner and VLA operate from first-person visual observations and interact through native keyboard-and-mouse actions. Reference decompositions, hidden diagnostic nodes, and privileged environment state are not exposed to the planner or VLA.

Final-goal evaluation. The primary LongBench metric is final-goal success. For each task family, Goal SR is computed over its formal evaluation episodes. For evaluation instance i, let $g_i \in \{0, 1\}$ denote whether the terminal task predicate is satisfied. We report

$$\text{GoalSR} = \frac{1}{N} \sum_{i=1}^{N} g_i. \tag{38}$$

Table 1 reports the family-level Goal SR, while the overall Goal SR in Table 3 is the macro-average over the four task families.

Final-goal success is authoritative: reaching intermediate milestones or producing partial progress does not convert an unsuccessful trajectory into a successful one. In addition to Goal SR and diagnostic progress, we record environment steps, upper-level planner calls, detector-triggered handoffs, local recovery attempts, and replanning events. These quantities are used for trajectory analysis rather than as optimization objectives.

Matched harness evaluation. To isolate the effect of agent orchestration from lower-level policy quality, the harness ablation keeps the native-control VLA and upper-level semantic model fixed. The compared systems therefore share the same local controller and differ only in how repeated VLA invocations are organized over time.

The fixed-horizon variant returns control after a predetermined execution boundary. Full HiSTEVE instead uses state-dependent handoff followed by visual verification and planning-state update; incomplete executions may trigger local recovery or replanning. Because the lower-level VLA is unchanged, this comparison isolates the effect of runtime orchestration from changes in policy weights. The corresponding controlled results are reported in Table 3.

Diagnostic failure patterns. LongBench exposes failures that are less visible in atomic benchmarks because successful completion depends on preserving usable state over many policy invocations. We observe three recurring patterns.

First, multi-step GUI interactions can leave the environment in a partially correct state that is difficult to continue from reliably. For example, an invocation may correctly place part of a recipe but fail to retrieve the output or recover from an incorrect slot configuration. A subsequent invocation then starts from a state that differs from the demonstrations seen by the policy.

Second, exploration tasks amplify errors in spatial memory. The agent may successfully discover a relevant area but later fail to recover sufficiently precise visual or spatial context for a return objective. These cases distinguish reaching a useful location once from maintaining information that remains actionable several planning steps later.

Third, recovery itself consumes execution budget. A local failure may be recoverable, yet repeated attempts can leave insufficient budget for the remaining dependency chain. LongBench therefore measures not only whether recovery is possible, but whether recovery is efficient enough to preserve the feasibility of the overall objective.

Table 8: Ablation of the state detector design. Each group changes one design factor while holding the remaining settings fixed.

Variant	Acc ↑	Prec ↑	Rec ↑	FPR ↓	FNR ↓	F1 ↑
Ours	0.863	0.861	0.865	0.140	0.135	0.863
FFN intermediate width						
Wanda, width $\frac{1}{2}$	0.865	0.863	0.867	0.138	0.133	0.865
Wanda, width $\frac{1}{4}$	0.739	0.739	0.740	0.262	0.260	0.739
FFN channel selection						
Prefix pruning, width $\frac{1}{3}$	0.768	0.768	0.768	0.232	0.232	0.768
Cross-attention density						
Every 2 layers	0.869	0.867	0.873	0.134	0.127	0.870
Every layer	0.859	0.859	0.859	0.141	0.141	0.859
Policy-context injection						
Layer-wise pairing	0.690	0.690	0.690	0.310	0.310	0.690

These observations explain why diagnostic progress can substantially exceed final-goal success: HiSTEVE often completes meaningful portions of a long trajectory before a later execution or recovery failure prevents satisfaction of the terminal predicate.

E.3 Controlled Harness Evaluation

The harness ablation is designed to separate improvements in agent orchestration from changes in lower-level policy quality. Across compared variants, we hold the native-control VLA, upper-level semantic model, task definitions, and evaluation protocol fixed.

The fixed-horizon variant executes the same atomic instruction until a predetermined execution boundary before returning control to the upper level. Full HiSTEVE instead uses state-dependent completion signals followed by visual verification, structured task-state update, local recovery, and replanning.

Because the underlying VLA is fixed, differences in long-horizon performance primarily reflect how the same local controller is organized across repeated invocations. This is also why we report atomic metrics for the same comparison: large changes in long-horizon behavior accompanied by only small changes in atomic performance provide evidence that the gain arises from orchestration rather than from a stronger controller.

F Ablation of the State Detector Design

We study three design choices in the state detector: FFN pruning, cross-attention density, and policy-context injection. Our final configuration uses Wanda-inspired scores (Sun et al., 2024) to retain approximately one-third of each FFN, inserts cross-attention after every fourth detector layer, and supplies the same final-layer policy context H_t^π to all cross-attention modules. All variants use the Qwen2.5-VL-3B backbone and are evaluated on the same class-balanced validation set of $4{,}674$ samples. Each variant changes only one factor while holding the remaining settings fixed.

Effect of FFN Width. Reducing the FFN width from one-half to one-third changes accuracy only from 0.865 to 0.863, whereas reducing it further to one-quarter decreases accuracy to 0.739. We therefore use one-third as the most compact tested width that preserves detection performance.

Effect of Channel Selection. At the same one-third width, naive prefix truncation reduces accuracy from 0.863 to 0.768. This result demonstrates the importance of selecting retained channels according to their activation–weight importance. We therefore use Wanda-inspired activation–weight scores for channel selection in the final detector.

Effect of Cross-Attention Density. Inserting cross-attention every two layers achieves an accuracy of 0.869, whereas inserting it after every layer yields 0.859, showing that greater density does not provide gains. The every-fourth-layer configuration uses only 9 modules, compared with 18 and 36 in the denser variants, while remaining within 0.006 accuracy of the best result. We therefore adopt the every-fourth-layer placement to balance detection performance and overhead.

Final-Layer Broadcast versus Layer-Wise Pairing. Using depth-matched intermediate policy representations reduces accuracy from 0.863 to 0.690. This substantial gap supports broadcasting the final-layer policy context to all cross-attention modules, as it integrates information from the current observation and subtask. We therefore broadcast the same final-layer policy context H_t^π to all cross-attention modules in the final detector.

G Discussion

HiSTEVE suggests that a central challenge in long-horizon agency is maintaining task state across planning and control decisions. The agent harness provides this continuity by preserving task state, aligning plans with executable capabilities, and using execution outcomes to make decisions.

With the lower-level VLA fixed, the complete harness changes aggregate atomic performance only modestly while substantially improving LongBench goal success. This suggests that long-horizon capability depends not only on the strength of the local controller, but also on how existing capabilities are organized across repeated invocations, with task state preserved and subsequent decisions adapted to execution outcomes. A further design insight is to decouple control handoff from task-success verification. The completion detector provides only a candidate handoff signal, while the harness subsequently verifies completion from visual evidence, preventing individual detector errors from being directly committed to persistent task state.

The current system has two main limitations. First, its atomic subtask interfaces are manually defined, and its capabilities remain limited by the existing VLA policy; adding new atomic capabilities requires additional training data and policy updates, while neither the harness nor the VLA currently performs parametric continual learning from deployment experience. In addition, high-level planning relies on an off-the-shelf VLM, making overall system performance dependent on the capabilities of the selected VLM backend.

Future work could enable independent or joint evolution of the VLA and harness: the VLA could learn continually from failure and recovery trajectories, while the harness updates its planning constraints and recovery strategies from experience. A complementary direction is to use Minecraft as training an environment for agent intelligence, examining whether planning, memory, and recovery capabilities can transfer across games and ultimately to interactive domains beyond games.