AHR: AMPLITUDE-HORIZON REFINEMENT FOR VISION-LANGUAGE-ACTION MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) policies provide strong semantic grounding and coherent motion proposals for robotic manipulation. However, small execution errors can prevent these capabilities from translating into reliable task completion. We introduce Amplitude–Horizon Refinement (AHR), which improves the execution of frozen VLA policies by jointly learning how to correct their action proposals and how long to execute each correction before re-observation. The key insight is that the intended correction horizon can change the appropriate correction from the first command. AHR therefore represents each refinement decision as a correction unit: a horizon-conditioned residual sequence paired with its execution length. Given RGB observations, proprioception, and a base proposal, a lightweight actor generates bounded residual candidates and selects a complete unit, allowing different corrections to overlapping commands across horizons. An off-policy actor–critic objective jointly optimizes residual generation and horizon selection using shared unit-level value estimates, duration-aware semi-Markov targets, and mixed demonstration and online replay. With a frozen GR00T N1.6 base policy, AHR achieves average success rates of 96.1% on RoboMimic-Pixel and 96.8% on DexMimicGen, outperforming the compared methods on both benchmarks. Across two real-world tasks, AHR achieves a 95.6% average success rate, compared with 65.0% for the frozen base policy. AHR thus provides an effective framework for refining frozen VLA policies by jointly learning action corrections and correction horizons.

1 INTRODUCTION

Vision-language-action (VLA) models provide a strong foundation for general-purpose manipulation by grounding language instructions in visual observations and learning diverse motor skills from demonstrations (Kim et al., 2025b; Black et al., 2025a). Yet semantically appropriate behavior does not guarantee precise physical execution. A robot may approach the correct object and follow the intended task sequence, but fail during insertion or stacking because of small alignment errors or unexpected contact (Ankile et al., 2025; 2026). These deviations can also lead to states poorly represented in the training data (Ross et al., 2011). Closing this gap requires refining how the robot executes an otherwise useful motion proposal.

Collecting corrective demonstrations for these states requires additional expert supervision. Online reinforcement learning (RL) enables refinement through the robot's own attempts and task outcomes, without expert action labels at newly encountered states (Yuan et al., 2025; Ankile et al., 2026). Residual RL provides a natural architecture for reusing pretrained capabilities while learning these improvements: a lightweight controller adjusts the actions of a frozen base policy (Silver et al., 2018; Johannink et al., 2019). Bounded residuals concentrate exploration on local changes to the proposed motion (Yuan et al., 2025). When the base policy generates action chunks (Black et al., 2025b; NVIDIA et al., 2025), however, refinement must specify more than individual adjustments. It must also determine which consecutive adjustments constitute one correction.

The length of a base proposal need not determine how many commands are executed before replanning (Zhao et al., 2026). In residual refinement, a fixed-length proposal can similarly support shorter or longer corrections because only the corrected prefix is executed. Mixture of Horizons learns and fuses action predictions at different spans within the action generator (Jing et al., 2026); we instead

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.

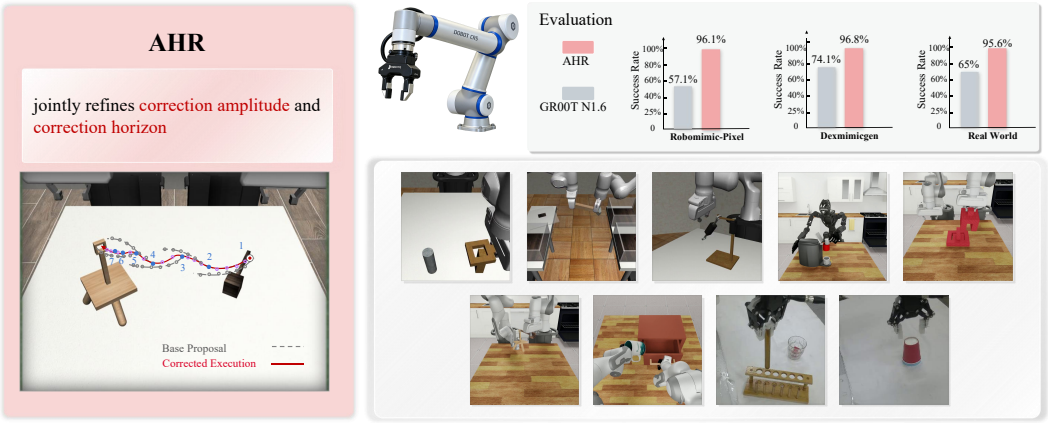


Figure 1: **Overview of AHR. Left:** AHR jointly learns horizon-conditioned residual corrections and execution horizons for a frozen VLA. Gray dashed lines denote base proposals, and the red solid line shows corrected execution. **Right:** Task-averaged success rates comparing AHR with the frozen GR00T N1.6 base on RoboMimic-Pixel, DexMimicGen, and real-world manipulation, with representative task scenes below.

keep that generator fixed and learn how to organize corrections to its proposals. We call the number of consecutive commands covered by a correction its *correction horizon*. Short corrections allow earlier feedback, whereas longer corrections can coordinate adjustments across more commands. Fixing this horizon predetermines how adjustments are grouped. Selecting variable-length prefixes of one horizon-independent residual sequence adapts the grouping but assigns identical residuals to overlapping steps. Neither representation allows the same initial commands to receive different corrections according to a variable intended span.

Our key insight is that the intended horizon can change the appropriate correction from its first step. Consider insertion under uncertain lateral alignment. A short correction can reposition the robot before a fresh observation guides advancement. A longer correction spanning alignment and entry must coordinate lateral and forward adjustments within the same segment. Its initial residuals may therefore differ, even for the same observation and base proposal. Conversely, better coordinated residuals can increase the return of a longer correction and change which horizon is preferred. We propose **Amplitude–Horizon Refinement (AHR)**, which reformulates residual refinement around a *correction unit*: a horizon-conditioned residual sequence paired with its length. By learning and selecting complete units, AHR couples how the proposal is adjusted with how many commands each correction covers.

Given RGB observations, proprioception, and a frozen VLA proposal, AHR's lightweight actor generates bounded residual sequences conditioned on a small set of candidate horizons. Candidates share an actor representation but specify adjustment magnitudes and directions for each span, allowing different corrections to overlapping steps. A discrete selection policy assigns probabilities to complete units, selecting a corrected motion together with its execution length. The robot adds the selected residuals to the corresponding base commands and executes the corrected prefix. It then constructs the next unit from a fresh observation and VLA proposal. The primitive control rate remains unchanged.

AHR jointly trains residual candidates and unit selection under a shared off-policy actor–critic objective. The critic evaluates each complete unit through the return of its corrected motion and subsequent decisions, so the same estimates guide residual optimization and horizon preference. Duration-aware semi-Markov targets account for the number of control steps actually executed, comparing different horizons under a common primitive-step discounted-return objective (Sutton et al., 1999). Following the use of offline data in online off-policy learning (Ball et al., 2023; Ankile et al., 2026), training combines demonstration and online segment transitions with sparse task-completion rewards, while the base VLA remains frozen.

We evaluate AHR on seven tasks across RoboMimic-Pixel and DexMimicGen and two real-world manipulation tasks. With a frozen GR00T N1.6, AHR achieves the highest mean success rate among

the compared methods on both simulation benchmarks: 96.1% on RoboMimic-Pixel and 96.8% on DexMimicGen. On real robots, AHR raises success from 72.0% to 98.0% on cup stacking and from 58.0% to 93.2% on test-tube insertion—gains of 26.0 and 35.2 percentage points over the frozen base policy. In sum, our contributions can be summarized as:

- A correction-unit formulation that treats residual values and their intended temporal span as one action, allowing the same base proposal to be refined differently for different horizons.
- An off-policy refinement architecture that jointly learns horizon-conditioned residual candidates and a discrete policy over complete units, optimizing their task returns while preserving the pretrained VLA.
- An empirical study of residual action structure in simulated and real-world manipulation, combining task-success evaluation, controlled ablations, and analysis of phase-dependent corrections.

2 Related Work

Vision-language-action policies. Vision-language-action (VLA) policies adapt pretrained multimodal knowledge to language-conditioned robot control, enabling large-scale visual and semantic priors to support generalizable manipulation (Zitkovich et al., 2023; Kim et al., 2025b). Recent VLAs increasingly pair such representations with expressive continuous-action generators. π_0, GR00T N1, DexVLA, and DiffusionVLA employ dedicated flow- or diffusion-based action modules to generate temporally structured robot motions (Black et al., 2025b; NVIDIA et al., 2025; Wen et al., 2025a;b). A complementary line improves downstream adaptation: OpenVLA-OFT develops efficient action decoding and fine-tuning, while InstructVLA and VLM2VLA study instruction-based adaptation and the preservation of pretrained multimodal capabilities during robot fine-tuning (Kim et al., 2025a; Yang et al., 2026; Hancock et al., 2026). Other work augments VLAs with long-horizon memory and more efficient inference for deployment (Shi et al., 2026; Li et al., 2026b). Beyond supervised adaptation, iRe-VLA and ConRFT use online reinforcement learning to update trainable action-generation components through interaction, including on real robots, while SimpleVLA-RL further explores reward-driven VLA post-training at scale (Guo et al., 2025; Chen et al., 2025; Li et al., 2026a; Xu et al., 2026). AHR instead improves the VLA at the execution level: it keeps the task-adapted VLA frozen and applies online RL through a base-policy-agnostic refinement interface over its proposed actions, improving execution without updating the underlying VLA parameters.

Residual reinforcement learning for manipulation. Residual reinforcement learning augments a base controller with additive corrective actions, retaining useful prior behavior while using task interaction to improve execution (Silver et al., 2018; Johannink et al., 2019). This paradigm has increasingly been applied to learned manipulation policies. ResiP combines action-chunked behavior cloning with closed-loop residual correction for precise assembly, while ResFiT learns lightweight per-step residuals from visual observations and proprioception using off-policy updates and demonstration replay (Ankile et al., 2025; 2026). Policy Decorator learns bounded residual action chunks with controlled online exploration (Yuan et al., 2025), and DICE-RL optimizes chunk-level residual actions and values with selective behavior regularization (Sun & Song, 2026). Further work improves residual learning through uncertainty-aware exploration and more effective value estimation (Dodeja et al., 2026; Ma et al., 2026). Residual refinement has also been integrated directly with pretrained VLAs: PLD uses residual specialists to generate recovery data for subsequent distillation, object-centric residual RL enables sim-to-real transfer of corrective policies, and BORA performs chunk-wise online residual adaptation for dexterous manipulation (Xiao et al., 2026; Kim et al., 2026; Chen et al., 2026). Human guidance and downstream-success objectives further support real-world adaptation and long-horizon coordination (Liu et al., 2026a;b). In residual refinement, the correction horizon is typically prescribed by the execution interface. A parallel line of work adapts proposal execution. AAC selects the executed prefix at inference time (Liang et al., 2026), while DEHP learns the horizon of a frozen chunk policy (Zhao et al., 2026). AHR reformulates residual refinement around correction units. Each unit pairs a horizon-conditioned residual sequence with its execution length and is learned and selected as a complete action. The correction horizon is thus part of the residual action itself, allowing the same initial base commands to receive different corrections for different intended spans. The underlying VLA remains frozen.

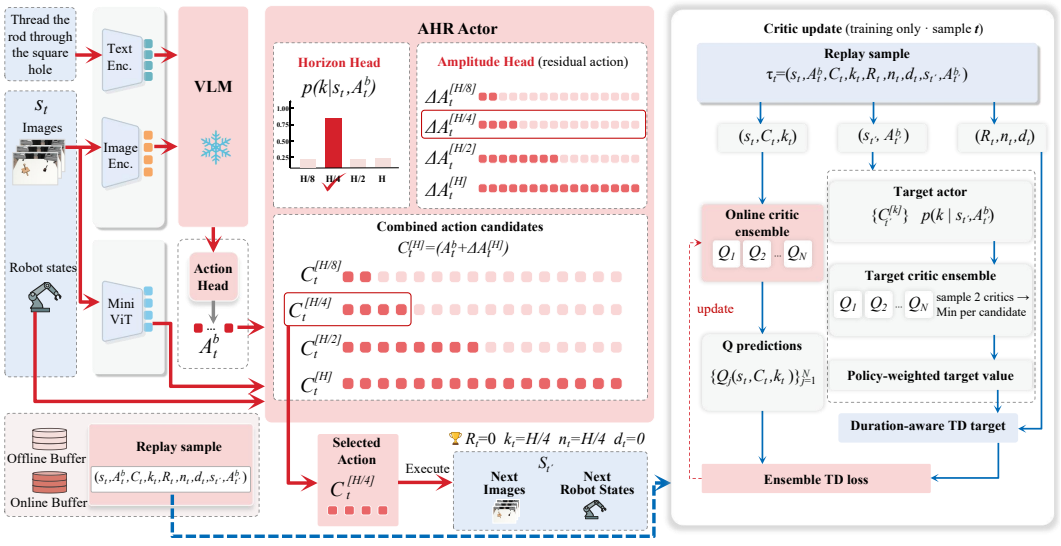


Figure 2: AHR architecture and learning pipeline. The frozen VLA supplies a base proposal, and the actor selects a horizon-conditioned correction unit. During training, sampled transitions are evaluated by the critic ensemble and fitted to a common duration-aware target with a policy-weighted successor value. The selected corrected prefix is executed before refreshing the VLA proposal.

3 METHOD

Amplitude–Horizon Refinement (AHR) learns *correction amplitudes* and *correction horizons* for a frozen VLA. A correction unit specifies both how a base action proposal is modified and how many corrected commands are executed before the next refinement decision. We learn these two components jointly from the task return of the resulting corrected motion.

3.1 PROBLEM SETUP

We first fine-tune a base VLA π_b on target-task demonstrations and then freeze it. At each refinement decision, the robot receives an observation o_t comprising RGB images and proprioception, together with a task instruction ℓ. We use s_t as shorthand for the available observation; full-state observability is not assumed. The frozen base policy produces an action proposal

$$A_t^b = \pi_b(s_t, \ell) = \left(a_{t,1}^b, \ldots, a_{t,H}^b\right), \tag{1}$$

where H is the proposal length and $a_{t,j}^b \in \mathbb{R}^{d_a}$ denotes its j-th control command.

The refiner modifies a prefix of A_t^b and determines how many corrected commands to execute before the next refinement decision. We optimize the expected discounted task return using sparse task-completion rewards and a primitive-step discount factor γ. The primitive control rate remains fixed; the correction horizon changes only the interval between successive refinement decisions.

3.2 AMPLITUDE–HORIZON CORRECTION UNITS

A correction unit contains two components: correction amplitudes and a correction horizon. Let $\mathcal{K} \subseteq \{1, \ldots, H\}$ denote a finite set of candidate correction horizons fixed for a given training configuration. Unless otherwise specified, we use $\mathcal{K} = \{\lceil H/8 \rceil, \lceil H/4 \rceil, \lceil H/2 \rceil, H\}$ (Appendix B.3). At each refinement decision, the policy selects $k_t \in \mathcal{K}$ according to the current execution context.

For a candidate horizon k, the correction amplitudes are represented by a residual sequence $\Delta A_t^{[k]} \in \mathbb{R}^{k \times d_a}$. Here, "amplitudes" refer to signed adjustments along each action dimension at each control step, rather than to a single scalar gain. Following the additive interface of residual RL (Johannink et al., 2019), we define the correction unit and its corrected segment as

$$u_t^{[k]} = \left(\Delta A_t^{[k]}, k\right), \qquad C_t^{[k]} = \mathrm{clip}\left(A_{t,1:k}^b + \Delta A_t^{[k]}\right). \tag{2}$$

The base commands and correction amplitudes use the same normalized action coordinates, and clipping acts elementwise on $[-1, 1]$. The corrected segment specifies the full k-step command sequence selected at the decision point; execution may stop earlier if the episode ends.

The appropriate correction amplitudes can depend on the correction horizon. For example, a short insertion correction may primarily resolve alignment, whereas a longer correction may need to coordinate alignment and forward motion before the next observation. Their initial corrections can therefore differ even under the same observation and base proposal. AHR captures this dependence by not constraining residual sequences for different horizons to share a common prefix. In contrast, truncating a single horizon-independent residual sequence changes only its execution length while preserving identical corrections on overlapping steps.

The actor fuses features from a lightweight RGB encoder with proprioception and the base proposal into a shared representation. A shared residual decoder uses this representation and a learned embedding of each candidate horizon to generate bounded residuals $\Delta A^{[k]}_{\theta,t} = f_\theta(s_t, A^b_t; k)$, while a selection head predicts categorical probabilities $p_\theta(k \mid s_t, A^b_t)$ over $\mathcal{K}$. Each residual is paired with its horizon as a correction unit; selecting k_t chooses the corresponding corrected prefix in Eq. 2.

During collection, we sample $k_t \sim p_\theta(\cdot \mid s_t, A^b_t)$ and perturb the corresponding residual sequence for exploration (Appendix A.1). The corrected prefix is mapped to the environment action representation and executed. After the segment completes, AHR obtains a fresh observation and base proposal and discards the unused suffix. Evaluation selects the highest-probability unit without exploration. Deployment requires only the frozen VLA and the refinement actor.

3.3 Joint Off-Policy Learning

AHR jointly learns residual generation and horizon selection using shared value estimates for complete correction units. The base VLA remains frozen; online updates are restricted to the actor–critic and visual encoder within the AHR refinement module.

Segment replay. Let C_t denote the complete k_t-step corrected segment selected for collection, after residual exploration and command clipping. Execution lasts for $1 \le n_t \le k_t$ primitive steps, with $n_t < k_t$ only if the episode ends early. We store the transition at the primitive-step index t; refinement decisions occur at a subset of these indices.

$$\begin{aligned} R_t &= \sum_{j=0}^{n_t-1} \gamma^j r_{t+j}, \qquad t' = t + n_t, \\ \tau_t &= \left(s_t, A^b_t, C_t, k_t, R_t, n_t, d_t, s_{t'}, A^b_{t'}\right). \end{aligned} \tag{3}$$

The selected segment C_t and horizon k_t are retained even when $n_t < k_t$; rewards and the bootstrap discount use only the realized duration n_t. The task objective ends at success, failure, or the prescribed task deadline. We set $d_t = 1$ at these boundaries and do not bootstrap beyond the end of the attempt.

Each minibatch $\mathcal{B}$ contains equal numbers of transitions from a fixed demonstration buffer and a growing online buffer (Ball et al., 2023; Ankile et al., 2026). Offline replay reuses the demonstrations used to adapt the base VLA: for each candidate horizon, we extract complete expert-action windows and query the frozen base policy at their start and nonterminal successor observations. We set C_t to the demonstrated command window with $n_t = k_t$. Both buffers use Eq. 3 and the same objectives; construction and sampling details are given in Appendix A.2.

Duration-aware value learning. We use N scalar value heads $Q_{\phi_i}(s, C, k)$ with a shared critic trunk, whose collective parameters are denoted by Φ. Each head approximates the return from executing the selected segment until completion or episode end, and following the refinement policy thereafter. All command inputs use the same clipping and padding convention (Appendix E.2).

For each nonterminal successor, the target actor regenerates all candidate units with target smoothing, yielding $\widetilde{C}^{[k]}_{\bar{\theta},t'}$. Following REDQ's randomized in-target minimization (Chen et al., 2021), we uniformly sample a two-element subset $\mathcal{I}$ of target-head indices without replacement and define

$Q_{\mathrm{tgt}}(s, C, k) = \min_{i \in \mathcal{I}} Q_{\bar{\phi}_i}(s, C, k)$. The shared target and ensemble loss are

$$y_t = R_t + (1 - d_t)\gamma^{n_t} \sum_{k \in \mathcal{K}} p_{\bar{\theta}}\left(k \mid s_{t'}, A^b_{t'}\right) Q_{\mathrm{tgt}}\left(s_{t'}, \widetilde{C}^{[k]}_{\bar{\theta},t'}, k\right),$$
$$\mathcal{L}_Q(\Phi) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathcal{B}}\left[\left(Q_{\phi_i}(s_t, C_t, k_t) - y_t\right)^2\right]. \tag{4}$$

Here, $\mathbb{E}_{\mathcal{B}}$ denotes the minibatch average, and y_t is held fixed during critic optimization. The factor γ^{n_t} accounts for the realized duration under a common primitive-step discounted-return objective (Sutton et al., 1999).

Joint amplitude–horizon optimization. The current actor regenerates all correction units at replayed observations and base proposals. Critic parameters are held fixed and visual features are detached, while gradients through the critic's command inputs are retained. Let

$$Q_{\mathrm{avg}}(s, C, k) = \frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i}(s, C, k).$$

We optimize

$$\mathcal{L}_\pi(\theta) = -\mathbb{E}_{\mathcal{B}}\left[\sum_{k \in \mathcal{K}} p_\theta(k \mid s_t, A^b_t) Q_{\mathrm{avg}}\left(s_t, C^{[k]}_{\theta,t}, k\right)\right]. \tag{5}$$

The sum is evaluated over all candidate correction units. Gradients through $C^{[k]}_{\theta,t}$ optimize the correction constructed for horizon k, while gradients through $p_\theta(k \mid s_t, A^b_t)$ optimize the probability of selecting that unit. Each selection probability also weights its candidate's residual gradient. Thus, both components are optimized using the same unit-level values, while horizon conditioning allows overlapping commands to receive different corrections.

Training begins with online replay collection and critic warm-starting, followed by alternating collection and optimization. We allocate g critic updates per newly collected primitive environment step and update the actor every d_π critic updates. Algorithm 1 specifies the sampling, update schedule, and target-network updates.

4 Experiments

4.1 Comparison with Policy Improvement Methods

Simulation benchmarks. We evaluate on seven image-based manipulation tasks from RoboMimic-Pixel and DexMimicGen. RoboMimic-Pixel includes Square, Transport, and Tool Hang, while DexMimicGen includes Coffee, Piece Assembly, Threading, and Drawer Cleanup. We fine-tune GR00T N1.6 separately for each task using 100 demonstrations for RoboMimic-Pixel and 500 for DexMimicGen, and keep the resulting task-specific VLA fixed during online refinement.

Baselines. We compare AHR with complementary policy-improvement baselines. DSRL (Wagenmaker et al., 2025) steers frozen generative policies in latent space. ResFiT (Ankile et al., 2026) applies stepwise residuals while retaining the base proposal until its scheduled refresh. DICE-RL (Sun & Song, 2026) learns fixed-horizon residual chunks with selective behavior regularization and value-guided action selection. Our **Full-Chunk Residual + AAC (FCR+AAC)** baseline learns a horizon-independent residual over the full 16-step VLA proposal and applies training-free AAC (Liang et al., 2026) at inference to select the executed prefix, combining fixed-horizon residual learning with adaptive execution.

Comparison protocol. All methods use the same task-specific frozen GR00T N1.6 checkpoint and common task interface under matched primitive-environment-step budgets. No additional expert trajectories are collected: the original base-adaptation demonstrations are reprocessed into each method's offline replay representation. Baselines retain their method-specific learning procedures, with execution horizons and adaptations to the GR00T interface specified in Appendix B.6.

Simulation performance. AHR achieves the highest mean success rate on all seven tasks, averaging 96.1% on RoboMimic-Pixel and 96.8% on DexMimicGen (Table 1). DSRL steers the frozen generator through its latent-noise space rather than directly correcting action proposals. It reaches 98.0% on COFFEE, but 75.0% on TRANSPORT and 61.0% on TOOL HANG. ResFiT instead applies feedback-driven residual corrections, reaching 98.0% on SQUARE, but 76.0% on TRANSPORT and 55.2% on TOOL HANG. In our implementation, it retains the base proposal between scheduled refreshes; earlier corrections can therefore leave the remaining suffix mismatched to the realized trajectory.

DICE-RL (Sun & Song, 2026) uses fixed-horizon residual refinement; we follow its released RoboMimic-Pixel setup and validation-select the fixed horizon for our DexMimicGen extension (Appendix B.6). With selective behavior regularization and value-guided selection, it reaches 90.0% on TRANSPORT and 88.0% on TOOL HANG, whereas AHR selects horizon-conditioned correction units and reaches 95.0% and 94.0%, respectively. FCR+AAC adapts execution length at inference after generating horizon-independent corrected proposals. AHR exceeds this composition by 4.6, 4.4, and 5.4 percentage points on TOOL HANG, THREADING, and DRAWER CLEANUP, respectively. These results favor joint amplitude–horizon refinement over adding adaptive execution to a fixed-horizon refiner.

Table 1: Success rates (%) on the selected RoboMimic-Pixel and DexMimicGen tasks. We report the mean ± standard deviation over five random seeds, with 100 evaluation rollouts per seed. For each benchmark, we first average task success rates within each training seed and then report the mean and standard deviation across seeds.

Method	RoboMimic-Pixel			RoboMimic-Pixel Avg.	DexMimicGen				Dex Avg.
	Square	Transport	Tool Hang		Coffee	Piece Assembly	Threading	Drawer Cleanup	
GR00T N1.6	81.0± 2.6	52.4± 3.4	38.0± 4.0	57.1± 3.3	84.0± 3.1	79.2± 3.6	72.0± 3.8	61.0± 3.2	74.1± 3.4
DSRL	95.8± 0.4	75.0± 1.2	61.0± 1.2	77.3± 0.9	98.0± 1.4	88.8± 3.9	91.2± 2.7	86.4± 2.1	91.1± 2.5
ResFiT	98.0± 0.7	76.0± 3.5	55.2± 4.1	76.4± 2.7	94.0± 2.8	88.0± 3.7	83.2± 3.3	83.2± 3.3	87.1± 2.7
DICE-RL	97.2± 0.8	90.0± 1.6	88.0± 1.7	91.7± 1.0	97.2± 1.5	92.0± 2.6	90.0± 2.8	88.0± 2.6	91.8± 2.0
FCR+AAC	97.6± 1.1	91.8± 0.8	89.4± 1.1	92.9± 1.0	98.2± 0.8	93.4± 1.1	91.6± 1.1	89.8± 1.3	93.3± 1.1
AHR	**99.2**± 0.4	**95.0**± 0.7	**94.0**± 1.2	**96.1**± 0.8	**99.2**± 1.1	**96.8**± 2.3	**96.0**± 2.4	**95.2**± 1.8	**96.8**± 1.9

4.2 UNDERSTANDING AMPLITUDE–HORIZON REFINEMENT

4.2.1 DISSECTING THE CORRECTION UNIT

We examine how correction amplitudes and horizons should be combined through controlled comparisons on THREADING (Table 2). Best Fixed is selected on validation data from nine independently trained fixed-horizon residual policies with $k \in \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$; each executes k corrected commands before re-observing and querying a fresh base proposal. Horizon-only adapts the horizon without modifying the base proposal. Shared Residual jointly learns residual correction and horizon selection, but constrains all candidate horizons to prefixes of a single horizon-independent residual sequence. AHR instead conditions each candidate's correction on its intended horizon. Appendix C.1.1 details the variant configurations and fixed-horizon selection protocol.

Table 2: Correction-unit variants on THREADING.

Variant	Residual correction	Adaptive horizon	Horizon-conditioned residual	Success (%)
Best Fixed ($k^\star = 4$)	✓	×	×	90.2± 2.8
Horizon-only	×	✓	×	83.8± 3.3
Shared Residual	✓	✓	×	91.8± 1.9
AHR	✓	✓	✓	96.0± 2.4

Among independently trained fixed-horizon residual policies, THREADING success peaks at $k = 4$ (90.2%), compared with 84.6% for Fixed-1, which applies a one-step residual correction before re-observing and querying a fresh VLA proposal (Appendix C.1.1). Success declines at longer horizons, consistent with a trade-off between feedback frequency and temporally coordinated correction. The validation-selected fixed correction horizons are $k = 4, 8, 4$, and 4 for SQUARE, TRANSPORT, TOOL HANG, and THREADING, respectively (Appendix C.1.2). AHR reaches 96.0% on THREADING, improving over Best Fixed and Horizon-only by 5.8 and 12.2 percentage points. Within the same joint learning framework, Shared Residual adapts the correction horizon but assigns identical residuals to overlapping commands. AHR instead conditions these residuals on their intended horizon, improving over Shared Residual by 4.2 points on THREADING and 1.2–4.2 points across the

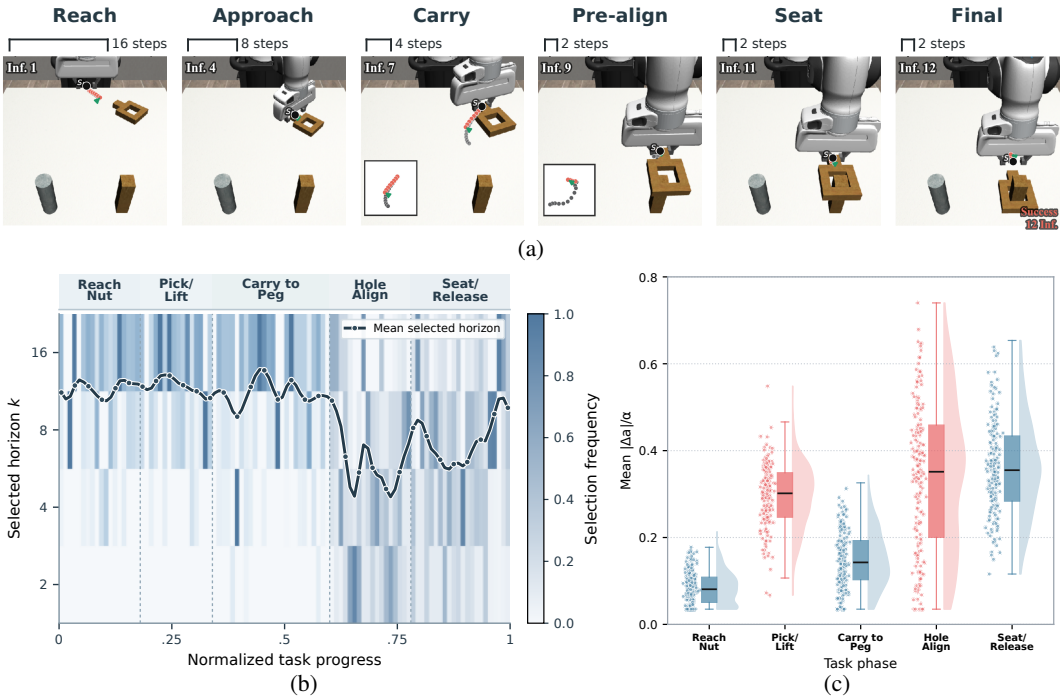


Figure 3: Phase-dependent correction behavior of AHR on SQUARE. (a) Representative rollout showing adaptive correction horizons during different task phases. (b) Horizon-selection frequencies across evaluation rollouts, normalized by the number of refinement decisions in each progress bin. (c) Mean absolute residual magnitude over action dimensions and commands, normalized by the residual bound $\alpha = 0.1$.

four tasks. Appendix C.1.3 characterizes this distinction: for a fixed selector distribution, shared-prefix constraints are lossless exactly when the active horizon-specific optima are prefix-compatible.

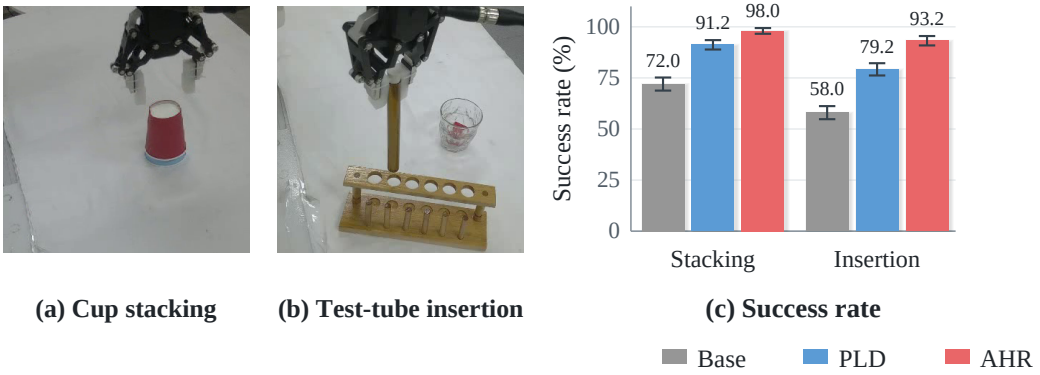
4.2.2 PHASE-DEPENDENT CORRECTION BEHAVIOR

Fig. 3(a) shows horizon choices at selected SQUARE execution states. The heatmap in Fig. 3(b) shows a shift from longer horizons during reach, pick/lift, and carry to shorter horizons during alignment. The phase-averaged horizon falls from approximately 12 to 6.5 steps, then rises to 7.8 during seat/release, where selection spans multiple horizons. Fig. 3(c) reports mean absolute residuals across action dimensions and commands in each selected correction unit. Corrections are small during reach and carry; alignment and seating show broader distributions, consistent with state-dependent refinement around precision-sensitive interactions. This phase-dependent refinement uses a lightweight AHR module. With the final trained weights on an RTX 5090, mean inference latency is 4.7 ms per AHR decision and 61.5 ms per GR00T N1.6 proposal.

4.3 REAL-WORLD EXPERIMENTS

We evaluate cup stacking and test-tube insertion on a Dobot CR5 using RGB observations and proprioception (Fig. 4(a,b)). For each task, Base, PLD, and AHR share the same frozen GR00T N1.6 checkpoint, adapted on 110 demonstrations for stacking and 170 for insertion. We evaluate PLD's single-step residual specialist before distillation (Xiao et al., 2026). AHR and PLD use matched per-run online-interaction budgets of 60k primitive steps for stacking and 150k for insertion (Appendix D).

AHR and PLD each undergo five independent training runs per task, with 50 matched evaluation trials per model. AHR achieves 98.0% success on stacking, compared with 91.2% for PLD and 72.0% for Base. On insertion, AHR reaches 93.2%, compared with 79.2% for PLD and 58.0% for Base. The gains over Base show that online refinement improves task success without updating the VLA parameters. AHR also improves on stepwise residual refinement, with a larger margin on insertion



(a) Cup stacking **(b) Test-tube insertion** **(c) Success rate**

Figure 4: Real-world evaluation. (a,b) Cup stacking and test-tube insertion. (c) Mean success and sample standard deviation over five independent training runs for AHR and the PLD specialist (50 trials per model); Base uses five matched evaluation batches.

(14.0 percentage points) than on stacking (6.8 points). Together with the controlled ablations in Section 4.2, these results support learning residual corrections jointly with their execution horizons for precise manipulation. Representative executions appear in Figs. 11 and 12.

4.4 Ablations

Figure 5 compares frozen base checkpoints and candidate horizon sets. AHR consistently improves the tested base policies. Better-trained bases reach high success with fewer environment interactions and generally converge to slightly higher final success; weaker checkpoints need more online refinement but still improve substantially. Using only $\{\lceil H/2 \rceil, H\}$ slightly reduces performance, whereas denser candidate sets converge more slowly and approach similar performance with longer training. This suggests that covering relevant correction timescales matters more than candidate density. Algorithm 1 and Appendix E detail the training procedure and update schedule.

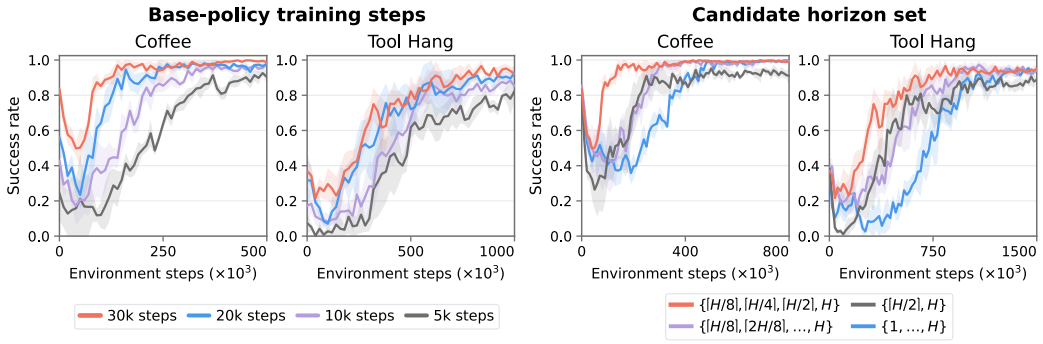


Figure 5: Ablations on Coffee and Tool Hang. The left panels vary base-policy training steps, while the right panels vary candidate-horizon sets. Candidate-horizon runs continue to 0.8M steps on Coffee and 1.5M on Tool Hang.

5 Conclusion

We introduced AHR, a refinement architecture for frozen VLAs based on horizon-conditioned correction units. Each unit couples correction content with its intended execution duration, allowing refinement to adapt to its horizon. Experiments in simulation and on real robots show that AHR improves task success while keeping the VLA parameters frozen. Future work could combine this unit-based refinement with asynchronous real-time chunking to reduce inference pauses and smooth transitions between units.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation, searching for related literature, code generation and editing, and suggesting candidate hyperparameter settings. All AI-assisted code was manually reviewed and tested by the authors, and the final hyperparameter settings were determined and validated through our own experiments. All AI-assisted outputs were reviewed by the authors, who take responsibility for the final content of this work.

REFERENCES

Lars Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne, and Pulkit Agrawal. From imitation to refinement—residual RL for precise assembly. In *IEEE International Conference on Robotics and Automation*, pp. 1–8. IEEE, 2025.

Lars Ankile, Zhenyu Jiang, Rocky Duan, Guanya Shi, Pieter Abbeel, and Anusha Nagabandi. Residual off-policy RL for finetuning behavior cloning policies. In *IEEE International Conference on Robotics and Automation*, 2026.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 1577–1594. PMLR, 2023.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. In *Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert 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, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025b.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

Yuhui Chen, Shuai Tian, Shugao Liu, Yingting Zhou, Haoran Li, and Dongbin Zhao. ConRFT: A reinforced fine-tuning method for VLA models via consistency policy. In *Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025.

Zhongxi Chen, Yifan Han, Yanming Shao, Huanming Liu, Congsheng Xu, Xiaoyu Chen, Yao Mu, and Wenzhao Lian. BORA: Bridging offline reinforcement learning and online residual adaptation for real-world dexterous VLA models, 2026. URL https://arxiv.org/abs/2605.30226.

Lakshita Dodeja, Karl Schmeckpeper, Shivam Vats, Thomas Weng, Mingxi Jia, George Konidaris, and Stefanie Tellex. Accelerating residual reinforcement learning with uncertainty estimation. *IEEE Robotics and Automation Letters*, 11(1):970–977, 2026.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1587–1596. PMLR, 2018.

Yanjiang Guo, Jianke Zhang, Xiaoyu Chen, Xiang Ji, Yen-Jen Wang, Yucheng Hu, and Jianyu Chen. Improving vision-language-action model with online reinforcement learning. In *IEEE International Conference on Robotics and Automation*, pp. 15665–15672. IEEE, 2025.

Asher Hancock, Xindi Wu, Lihan Zha, Olga Russakovsky, and Anirudha Majumdar. Actions as language: Fine-tuning VLMs into VLAs without catastrophic forgetting. In *International Conference on Learning Representations*, 2026.

Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. DexMimicGen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *IEEE International Conference on Robotics and Automation*, 2025.

Dong Jing, Gang Wang, Jiaqi Liu, Weiliang Tang, Zelong Sun, Yunchao Yao, Zhenyu Wei, Yunhui Liu, Zhiwu Lu, and Mingyu Ding. Mixture of horizons in action chunking. In *International Conference on Machine Learning*, 2026.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *IEEE International Conference on Robotics and Automation*, pp. 6023–6029. IEEE, 2019.

Kinam Kim, Namiko Saito, Heecheol Kim, Katsushi Ikeuchi, Jaegul Choo, and Yasuyuki Matsushita. Object-centric residual RL for zero-shot sim-to-real VLA enhancement, 2026. URL https://arxiv.org/abs/2606.18953.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Robotics: Science and Systems*, Los Angeles, CA, USA, June 2025a.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b.

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Yang Zhaohui, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations*, 2026a.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A joint model scheduling and token pruning approach for VLA model acceleration. In *International Conference on Learning Representations*, 2026b.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Jingyi Liu, Zhaohong Mai, ShunSen He, Hang Ren, Chao Wang, Shunbo Zhou, XiaoDong Wu, and Heng Zhang. HiL-ResRL: A model-agnostic finetuning adapter via human-in-the-loop residual reinforcement learning, 2026a. URL https://arxiv.org/abs/2606.22860.

Yuhan Liu, Xinyu Zhang, Litao Liu, and Abdeslam Boularias. Foresight residual RL for long-horizon robot manipulation with vision-language-action models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2026b.

Guozheng Ma, Lu Li, Haoyu Wang, Zixuan Liu, Pierre-Luc Bacon, and Dacheng Tao. What makes value learning efficient in residual reinforcement learning? In *International Conference on Machine Learning*, 2026.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2022.

NVIDIA. Isaac-GR00T. GitHub repository, 2025a. URL https://github.com/NVIDIA/Isaac-GR00T/tree/n1d6. GR00T N1.6 implementation, n1d6 branch. Accessed 21 September 2026.

NVIDIA. GR00T N1.6-3B model card. Hugging Face, 2025b. URL https://huggingface.co/nvidia/GR00T-N1.6-3B. Accessed 21 September 2026.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Jim Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

Physical Intelligence. openpi. GitHub repository, 2025. URL https://github.com/Physical-Intelligence/openpi/blob/main/src/openpi/training/config.py. $\pi_{0.5}$ fine-tuning, pi05_libero configuration. Accessed 21 September 2026.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635. PMLR, 2011.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 18567–18602, 2026.

Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Pack Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018. URL https://arxiv.org/abs/1812.06298.

Zhanyi Sun and Shuran Song. From prior to pro: Efficient skill mastery via distribution contractive RL finetuning. In *International Conference on Machine Learning*, 2026.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Andrew Wagenmaker, Yunchu Zhang, Mitsuhiko Nakamoto, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 258–282. PMLR, 2025.

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. DexVLA: Vision-language model with plug-in diffusion expert for general robot control. In *Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 3094–3114. PMLR, 2025a.

Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. DiffusionVLA: Scaling robot foundation models via unified diffusion and autoregression. In *International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66558–66574. PMLR, 2025b.

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Zhengyi Luo, Yuqi Xie, Fengyuan Hu, Jim Fan, Guanya Shi, and Yuke Zhu. Self-improving vision-language-action models with data generation via residual RL. In *International Conference on Learning Representations*, volume 2026, pp. 13209–13236, 2026.

Charles Xu, Jost Tobias Springenberg, Michael Equi, Ali Amin, Adnan Esmail, Sergey Levine, and Liyiming Ke. RL Token: Bootstrapping online rl with vision-language-action models, 2026. URL https://arxiv.org/abs/2604.23073v2. arXiv preprint arXiv:2604.23073v2.

Shuai Yang, Hao Li, Bin Wang, Yilun Chen, Yang Tian, Tai Wang, Hanqing Wang, Feng Zhao, Yiyi Liao, and Jiangmiao Pang. Vision-language-action instruction tuning: From understanding to manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 152896–152943, 2026.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model. In *International Conference on Learning Representations*, 2025.

Yuchi Zhao, Miroslav Bogdanovic, Arjun Sohal, Liyu Tao, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Dynamic execution horizon prediction for chunk-based robot policies, 2026. URL https://arxiv.org/abs/2606.11408.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023.

APPENDIX CONTENTS

A AHR Algorithm and Replay Construction 15

A.1 AHR Algorithm 15

A.2 Offline Replay Construction 16

B Simulation Experimental Details 17

B.1 Benchmarks and Tasks 17

B.2 Observation, Action, and Execution Interface 17

B.3 Base VLA Policy 17

B.4 Common Online Training Protocol 19

B.5 Evaluation Protocol 20

B.6 Baseline Implementations 20

B.7 Generalization across Base VLA Backbones 21

C Controlled Analyses and Additional Ablations 22

C.1 Controlled Analyses of the Correction Unit 22

C.1.1 Variant Configurations and Fixed-Horizon Selection 22

C.1.2 Task-Specific Controlled Correction-Unit Analyses 24

C.1.3 Shared-Prefix Compatibility in Joint Amplitude–Horizon Learning 26

C.2 Additional Ablation Studies 29

C.3 Inference Cost Analysis 30

D Real-World Experimental Details 32

D.1 Hardware, Tasks, and Reset Procedure 32

D.2 Base-Policy Adaptation and Online Refinement 32

D.3 Evaluation Protocol and Success Metric 33

D.4 Real-World Task Sequences 33

E Reproducibility Details 34

E.1 Training Losses 34

E.2 Network Architecture 34

E.3 AHR Hyperparameters 36

A AHR ALGORITHM AND REPLAY CONSTRUCTION

A.1 AHR ALGORITHM

Algorithm 1 summarizes AHR training and deployment. Training first collects an initial online buffer and warm-starts the critics using mixed offline and online replay. It then alternates online collection with critic and delayed actor updates. The base VLA remains frozen throughout.

Algorithm 1 Amplitude–Horizon Refinement (AHR)

Input: frozen VLA π_b, offline segment replay $\mathcal{D}_{\text{off}}$ (Appendix A.2), and candidate horizons $\mathcal{K}$
Initialize: actor $\{f_\theta, p_\theta\}$, critics $\{Q_{\phi_i}\}_{i=1}^{N}$, target networks, and online replay $\mathcal{D}_{\text{on}}$
Notation: s_t comprises RGB images and proprioception
Initial collection
Build input: observe s_t and query $A_t^b \leftarrow \pi_b(s_t, \ell)$
for $\text{step} = 1;\ \text{step} \le T_{\text{warm}};\ \text{step} \leftarrow \text{step} + n_t$ **do**
  Build (s_t, A_t^b)
  Sample k_t uniformly from $\mathcal{K}$ and residual entries uniformly from $[-\eta, \eta]$
  Form $C_t = \text{clip}(A^b_{t,1:k_t} + \Delta A_t^{[k_t]})$ and execute until completion or episode end
  Observe $s_{t'}$, R_t, n_t, and d_t, with $t' = t + n_t$ as in Eq. 3
  Retain the complete selected segment C_t and horizon k_t
  If $d_t = 0$, query $A_{t'}^b \leftarrow \pi_b(s_{t'}, \ell)$; otherwise set $A_{t'}^b \leftarrow 0$
  Add $(s_t, A_t^b, C_t, k_t, R_t, n_t, d_t, s_{t'}, A_{t'}^b)$ to $\mathcal{D}_{\text{on}}$
  Use $(s_{t'}, A_{t'}^b)$ for the next decision, or reset and re-query if $d_t = 1$
end for
Critic warm-start: use the replay sampling rule below and update critics with Eq. 4
Initialize update credit $b \leftarrow 0$ and critic-update counter $m \leftarrow 0$
Online training
repeat
  Build (s_t, A_t^b)
  Generate $\Delta A_{\theta,t}^{[k]} = f_\theta(s_t, A_t^b; k)$ for each $k \in \mathcal{K}$; compute $p_\theta(k \mid s_t, A_t^b)$
  Sample $k_t \sim p_\theta(\cdot \mid s_t, A_t^b)$ and perturb $\Delta A_t^{[k_t]}$ with collection noise to obtain $\widetilde{\Delta A}_t^{[k_t]}$
  Form $C_t = \text{clip}(A^b_{t,1:k_t} + \widetilde{\Delta A}_t^{[k_t]})$ and execute until completion or episode end
  Observe $s_{t'}$, R_t, n_t, and d_t, with $t' = t + n_t$
  Retain the complete selected segment C_t and horizon k_t
  If $d_t = 0$, query $A_{t'}^b \leftarrow \pi_b(s_{t'}, \ell)$; otherwise set $A_{t'}^b \leftarrow 0$
  Add $(s_t, A_t^b, C_t, k_t, R_t, n_t, d_t, s_{t'}, A_{t'}^b)$ to $\mathcal{D}_{\text{on}}$
  Use $(s_{t'}, A_{t'}^b)$ for the next decision, or reset and re-query if $d_t = 1$
  $b \leftarrow b + g n_t$; $U_t \leftarrow \lfloor b \rfloor$; $b \leftarrow b - U_t$
  for $j = 1, \ldots, U_t$ **do**
    Sample half of $\mathcal{B}$ from offline replay using Appendix A.2 and half from $\mathcal{D}_{\text{on}}$; t below indexes sampled transitions
    For nonterminal successors, generate $\Delta A_{\bar\theta,t'}^{[k]} = f_{\bar\theta}(s_{t'}, A_{t'}^b; k)$ and $p_{\bar\theta}(k \mid s_{t'}, A_{t'}^b)$
    Apply target smoothing to form $\widetilde{C}_{\bar\theta,t'}^{[k]}$ for each $k \in \mathcal{K}$
    Randomly sample two target critics indexed by i_1, i_2
    Compute the target y_t using the sampled pair and Eq. 4
    Update Φ and the visual encoder using $\mathcal{L}_Q$ in Eq. 4
    Update critic targets $\bar\phi_i \leftarrow \rho\bar\phi_i + (1-\rho)\phi_i$
    $m \leftarrow m + 1$
    if $m \bmod d_\pi = 0$ **then**
      Regenerate $C_{\theta,t}^{[k]} = \text{clip}(A^b_{t,1:k} + f_\theta(s_t, A_t^b; k))$ for each $k \in \mathcal{K}$
      Update θ by minimizing $\mathcal{L}_\pi$ in Eq. 5
      Update actor target $\bar\theta \leftarrow \rho\bar\theta + (1-\rho)\theta$
    end if
  end for
until the training budget is exhausted
Deployment: build (s_t, A_t^b), choose $k_t = \arg\max_{k\in\mathcal{K}} p_\theta(k \mid s_t, A_t^b)$, execute $C_{\theta,t}^{[k_t]} = \text{clip}(A^b_{t,1:k_t} + f_\theta(s_t, A_t^b; k_t))$, re-observe, and repeat

Initial online collection. To populate the online replay buffer, we sample the correction horizon uniformly from $\mathcal{K}$ and each residual entry uniformly from $[-\eta, \eta]$, where η is the initial exploration bound.

Collection noise. During subsequent online collection, we sample $k_t \sim p_\theta(\cdot \mid s_t, A_t^b)$ and perturb the selected residual: $\widetilde{\Delta A} = \mathrm{clip}_{[-1,1]}(\Delta A + \epsilon)$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$ and σ is the noise standard deviation.

Target smoothing. When constructing critic targets, we perturb target-actor residuals using the same rule, but clip the Gaussian noise to $[-c, c]$ before adding it (Fujimoto et al., 2018). Target smoothing is disabled during the critic-only warm-start by setting $\sigma = 0$.

Replay storage and critic input. Each online transition retains the complete selected k_t-step corrected segment after command clipping. Only positions beyond k_t are zero-padded to length H. Early episode termination changes the realized duration n_t, reward R_t, and terminal indicator d_t, but not the stored selected command sequence. The realized duration n_t determines the reward accumulation and bootstrap discount and is not an additional critic input.

A.2 Offline Replay Construction

AHR and its controlled variants use the same demonstration episodes that are used to adapt the frozen base policy. Let episode e contain N_e primitive transitions. For each horizon k evaluated in these variants, we construct a duration-specific replay pool $\mathcal{D}_k$ by extracting all complete k-step expert-action windows with stride one:

$$t \in \{0, \ldots, N_e - k\}, \qquad |\mathcal{D}_k| = \sum_e \max(N_e - k + 1, 0).$$

Each window remains within a single episode and includes its recorded successor observation at $t + k$. An episode contributes no record to $\mathcal{D}_k$ if $N_e < k$. Starting positions with fewer than k remaining transitions are excluded; the final complete window, starting at $N_e - k$, is retained. We do not truncate windows or append synthetic transitions to complete a window.

The demonstrated commands use the same normalized action coordinates as online commands and are stored directly as C_t, with selected and realized durations $k_t = n_t = k$. The segment reward is

$$R_t^{(k)} = \sum_{j=0}^{k-1} \gamma^j r_{t+j}.$$

The terminal indicator is taken from the final transition in the window. We query the frozen base policy at the starting observation and at the successor observation for nonterminal records; the successor base proposal is set to zero for terminal records. Each record uses the transition format in Eq. 3. Command tensors are zero-padded beyond k to the common proposal length H; these padding positions do not contribute to the reward or duration. For a successful episode whose sole success reward occurs on its final transition, exactly one window in each eligible pool $\mathcal{D}_k$ contains that reward.

During training, the adaptive variants allocate an equal share of each offline minibatch to every $k \in \mathcal{K}$, sampling uniformly with replacement within each duration-specific pool. Their marginal offline sampling distribution is therefore

$$P_{\text{off}}^{\text{adapt}} = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathrm{Uniform}(\mathcal{D}_k).$$

Fixed-k draws its entire offline minibatch uniformly with replacement from the corresponding pool $\mathcal{D}_k$. Sampling does not depend on rewards or terminal labels. AHR, Shared Residual, and Horizon-only share the same candidate pools and offline sampling rule. Their actor objectives and nonterminal target computations still evaluate all candidates in $\mathcal{K}$ at each replayed state.

The notation $\mathcal{D}_{\text{off}}$ refers to the offline replay representation for the method being trained; draws from it follow the sampling rule above. The offline pools remain fixed throughout training. All

controlled variants use the same offline minibatch size, offline-to-online sampling ratio, and update budget per primitive environment step. Each training minibatch contains equal numbers of offline and online transitions, during both critic warm-starting and subsequent training. This matches the source demonstrations and optimization budget while allowing the replay representation to reflect each method's available action durations.

B Simulation Experimental Details

B.1 Benchmarks and Tasks

We evaluate AHR on seven image-based manipulation tasks from RoboMimic-Pixel (Mandlekar et al., 2022) and DexMimicGen (Jiang et al., 2025). For each task, we adapt a separate GR00T N1.6 checkpoint using a task-specific demonstration set and freeze the resulting base policy before online refinement.

RoboMimic-Pixel. We evaluate on the Proficient-Human (PH) variants of Square, Transport, and Tool Hang. Each released PH dataset contains 200 successful demonstrations collected from a single proficient operator. We use 100 demonstrations from each task for GR00T N1.6 adaptation.

DexMimicGen. We evaluate on Coffee, Piece Assembly, Threading, and Drawer Cleanup. For each task, we use 500 demonstrations from the corresponding DexMimicGen dataset to adapt a task-specific GR00T N1.6 checkpoint.

Across both benchmarks, the demonstration set used for base-policy adaptation is reprocessed by each method that employs demonstration replay during online learning. No additional demonstrations are collected for online refinement.

B.2 Observation, Action, and Execution Interface

At each refinement decision, the policy receives multi-view RGB observations and robot proprioception. We use a task-specific camera configuration consistently for GR00T N1.6 adaptation and subsequent simulation experiments. Square, Transport, and Tool Hang use two RGB views, whereas Coffee, Piece Assembly, Threading, and Drawer Cleanup use three. AHR receives the same task-specific set of RGB streams together with proprioception and the frozen VLA action proposal.

GR00T N1.6 produces a 16-step action proposal. AHR predicts residual corrections in the same normalized action coordinates as the base proposal, and the corrected commands are clipped to $[-1, 1]$ before execution. Simulation runs at a fixed primitive control rate of 20 Hz. The selected correction horizon k_t changes only how many consecutive commands are executed before re-observation and replanning. After each corrected prefix, a fresh observation is acquired and the frozen VLA is queried for a new proposal.

B.3 Base VLA Policy

GR00T N1.6. GR00T N1.6 (NVIDIA, 2025b) serves as the primary base VLA in our main experiments. We adapt a separate task-specific checkpoint for each of the seven simulation tasks using the demonstration and observation configurations defined above. Adaptation is performed entirely before online refinement. For each task, the checkpoint obtained after 30,000 optimization steps is frozen and used as the common base policy for the unrefined Base, AHR, and all refinement baselines.

We use the standard GR00T post-training structure (NVIDIA, 2025a), keeping the pretrained language and visual backbones fixed while adapting the multimodal projector and diffusion action head. During online refinement, no gradients are propagated into the base VLA. The complete adaptation configuration, including optimization, augmentation, and action-chunk settings, is summarized in Table 3.

Table 3: GR00T N1.6 adaptation settings used for the simulation experiments.

Setting	Value
Initialization	pretrained GR00T N1.6
Task-specific checkpoints	one per task
Optimization steps	30,000
Optimizer	AdamW
Global batch size	64
Gradient accumulation	1
Peak learning rate	1×10^{-4}
LR schedule	cosine
Warm-up ratio	0.05
Weight decay	1×10^{-5}
Gradient clipping	1.0
Training precision	bfloat16
LLM backbone	frozen
Visual backbone	frozen
Multimodal projector	trainable
Diffusion action head	trainable
State dropout	0.2
Color jitter	brightness 0.3, contrast 0.4 saturation 0.5, hue 0.08
Main-experiment checkpoint	30,000 steps
Proposal length H	16
AHR horizons $\mathcal{K}$	$\{2, 4, 8, 16\}$

$\pi_{0.5}$. We use $\pi_{0.5}$ (Black et al., 2025a) as an additional pretrained VLA backbone with a different action-chunk horizon. For each task, we fine-tune a separate checkpoint from the pretrained $\pi_{0.5}$ base model using the same task-specific demonstration and observation configuration defined above. We follow the OpenPI full-finetuning recipe (Physical Intelligence, 2025). The adapted checkpoint is then frozen and used as the base policy for AHR.

We configure $\pi_{0.5}$ with a proposal length of $H = 10$, matching the OpenPI simulation fine-tuning setting used for LIBERO (Physical Intelligence, 2025). Candidate horizons follow the common rule below. Table 4 summarizes the adaptation settings.

Table 4: $\pi_{0.5}$ adaptation settings used for the simulation experiments.

Setting	Value
Initialization	pretrained $\pi_{0.5}$ base
Task-specific checkpoints	one per task
Optimization steps	30,000
Fine-tuning type	full fine-tuning
Optimizer	AdamW
Global batch size	256
Peak learning rate	5×10^{-5}
LR schedule	warm-up, then constant
Warm-up steps	10,000
Adam β_1, β_2	$(0.9, 0.95)$
Adam ϵ	10^{-8}
Weight decay	10^{-10}
Gradient clipping	1.0
EMA decay	0.999
Training precision	bfloat16
Proposal length H	10
AHR horizons $\mathcal{K}$	$\{2, 3, 5, 10\}$

Backbone-relative horizons. For each base VLA, AHR derives its candidate correction horizons from the corresponding proposal length H using

$$\mathcal{K}(H) = \{\lceil H/8 \rceil, \lceil H/4 \rceil, \lceil H/2 \rceil, H\}.$$

B.4 Common Online Training Protocol

For each task, we first adapt a task-specific base VLA and then freeze the resulting checkpoint. The unrefined Base policy and all refinement methods use this same checkpoint, observation streams, sparse task reward, reset distribution, and base-adaptation demonstration set. We denote the shared raw demonstration source by $\mathcal{D}_{\text{demo}}$. Each refinement method applies its own preprocessing and transition schema to construct an offline replay representation $\mathcal{D}_{\text{off}}^{(m)}$; in the AHR equations below, the superscript is omitted for readability. During refinement, method m collects its own online transitions in $\mathcal{D}_{\text{on}}^{(m)}$. No additional expert data are collected during online refinement. The base VLA remains fixed throughout refinement, while replay and execution rules specific to each method are given in Appendix B.6.

Online interaction is counted in primitive environment steps. A segment of realized duration n_t contributes n_t steps, independent of its nominal proposal or chunk length. Offline replay updates and critic warm-start updates do not consume environment steps. This common budget controls environment interaction; it does not equalize wall-clock time or VLA-query cost. Because a fresh proposal is obtained after each executed segment, shorter horizons entail more frequent re-observation and proposal refresh. The task-specific budgets are reported in Table 5; the Base policy is evaluated without online refinement. AHR follows Algorithm 1 and samples equally from $\mathcal{D}_{\text{off}}^{(\text{AHR})}$ and $\mathcal{D}_{\text{on}}^{(\text{AHR})}$ (Ball et al., 2023).

Table 5: Task-specific online interaction budgets used in simulation.

Task	Online interaction budget
SQUARE	500k
TRANSPORT	1M
TOOL HANG	1M
COFFEE	500k
PIECE ASSEMBLY	1M
THREADING	1M
DRAWER CLEANUP	1M

B.5 EVALUATION PROTOCOL

For each task and method, we train five independent seeds. After the prescribed online budget, we evaluate the resulting checkpoint with 100 evaluation rollouts, with exploration and parameter updates disabled. All methods use the same evaluation initialization, termination, and success criteria for a given task.

For each seed, success rate is the fraction of successful evaluation rollouts. We report the mean and standard deviation across the five training seeds. Benchmark-level averages are first computed across the evaluated tasks within each seed and then aggregated across seeds.

B.6 BASELINE IMPLEMENTATIONS

We compare AHR with prior policy-improvement methods and FCR+AAC. For each baseline, we use the same task-specific frozen GR00T N1.6 checkpoint, pixel observations, sparse reward, reset distribution, and primitive-step budget. The base-adaptation demonstrations come from the shared source $\mathcal{D}_{\text{demo}}$; each method constructs its own offline replay representation $\mathcal{D}_{\text{off}}^{(m)}$ and online buffer $\mathcal{D}_{\text{on}}^{(m)}$ according to its transition schema. Prior methods retain their published training and execution procedures.

We distinguish common experimental inputs from method-specific training procedures. All methods use the same raw demonstration episodes and frozen base-policy checkpoint, while each prior method constructs and samples replay according to its original procedure. For AHR and its controlled variants, the duration-specific offline replay pools and sampling rules are specified in Appendix A.2. Fixed-k samples uniformly from its corresponding pool $\mathcal{D}_k$, whereas the adaptive variants allocate equal shares of each offline minibatch to their candidate horizons and sample uniformly within each pool. These controlled variants use the same offline minibatch size, offline-to-online sampling ratio, and update budget per primitive environment step.

GR00T N1.6. The base-policy baseline directly executes the task-adapted frozen GR00T N1.6 checkpoint without online refinement or any additional trainable module. This baseline measures the performance of the task-adapted VLA before online policy improvement.

DSRL. DSRL (Wagenmaker et al., 2025) improves a frozen generative policy by steering the latent noise used for action generation. We use the DSRL-NA variant and apply its steering policy to the diffusion action head of the same frozen GR00T N1.6 checkpoint. The steering actor receives the task-specific RGB observations and proprioception, modifies the generative latent, and the frozen GR00T action head decodes the resulting latent into a 16-step action proposal. Training follows the original DSRL-NA procedure: the replay buffer is initialized with demonstration transitions and augmented with online transitions during interaction. The action-space critic is learned from replay, while the latent-noise critic distills action-space values through the frozen generator to train the steering actor. No action-space residual is applied after decoding. DSRL therefore provides a latent-space policy-improvement baseline, contrasting with AHR's refinement of the decoded action proposal.

ResFiT. ResFiT (Ankile et al., 2026) freezes the action-chunked base policy and learns a lightweight per-step residual controller with off-policy actor–critic learning. GR00T produces a

16-step base proposal at each scheduled policy refresh. At every primitive control step, ResFiT uses the current RGB observations and proprioception to predict a residual for the corresponding command in the retained proposal. The corrected command is executed immediately, while the remaining base commands are retained until the next scheduled refresh. We preserve the original TD3-style residual-learning procedure, including replay from $\mathcal{D}_{\text{off}}^{(\text{ResFiT})}$ and online transitions in $\mathcal{D}_{\text{on}}^{(\text{ResFiT})}$, critic warm-starting, bounded residual actions, and per-step exploration. A demonstration record contains the frozen base action, recorded expert action, successor base proposal, reward, and terminal flag; the residual is learned off-policy and is not fitted to an expert residual target. ResFiT therefore provides closed-loop per-step action correction while retaining a fixed base-proposal refresh schedule, separating stepwise residual feedback from AHR's adaptive segment-level correction.

DICE-RL. We adapt the released DICE-RL implementation (Sun & Song, 2026) to the same frozen GR00T N1.6 base, retaining its residual actor–critic, selective behavior regularization, adaptive demonstration/online replay, multi-sample value learning, and value-guided action selection.

GR00T supplies an $H = 16$-step base proposal, while DICE-RL uses a task-specific fixed execution horizon k_{DICE}. We use $k_{\text{DICE}} = 4$ for SQUARE and COFFEE, and $k_{\text{DICE}} = 8$ for TRANSPORT, TOOL HANG, PIECE ASSEMBLY, THREADING, and DRAWER CLEANUP. RoboMimic-Pixel follows the released DICE-RL setup; for our DexMimicGen extension, the fixed horizon is selected on validation rollouts and then held fixed.

After executing k_{DICE} commands, the method obtains a fresh observation and GR00T proposal and discards the unused suffix. The same fixed horizon is used during training and evaluation. All runs follow the primitive-step budgets and evaluation protocol in Appendices B.4 and B.5.

Full-Chunk Residual + AAC. FCR+AAC combines fixed-horizon residual learning with inference-time AAC (Liang et al., 2026). The refiner follows the Fixed-k configuration in Appendix C.1.1 with $k = H = 16$, learning a horizon-independent residual over each complete GR00T proposal. Replay construction and optimization follow the same Fixed-k protocol.

At evaluation, the trained refiner is frozen and AAC determines the execution horizon. For each observation, GR00T generates $M = 20$ proposal samples, each containing $H = 16$ commands. The refiner corrects each sample independently in normalized action coordinates, followed by elementwise clipping to $[-1, 1]$. AAC selects $k_t \leq H$ from the largest increase in prefix-averaged action entropy, subject to its motion-magnitude constraint (`move_th`=3). Both statistics use the corrected samples, with normalized and decoded action fields updated accordingly. Following the released implementation, we execute the k_t-step prefix of the first corrected sample, then obtain a fresh observation and repeat.

B.7 GENERALIZATION ACROSS BASE VLA BACKBONES

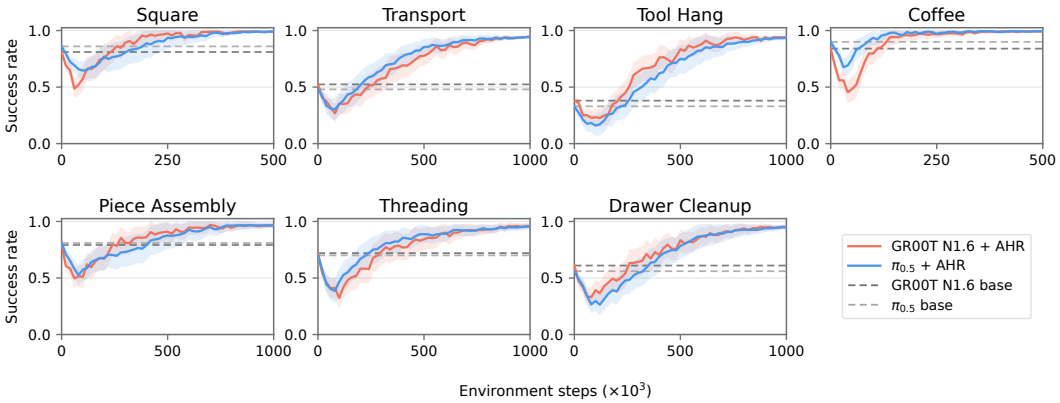


Figure 6: AHR learning curves with GR00T N1.6 and $\pi_{0.5}$ across seven tasks. Solid lines and shaded regions show the mean and one standard deviation across five seeds; dashed lines show the corresponding frozen-base performance.

C Controlled Analyses and Additional Ablations

C.1 Controlled Analyses of the Correction Unit

C.1.1 Variant Configurations and Fixed-Horizon Selection

We construct the four variants by changing the residual generator and horizon selector within the AHR actor–critic framework. The comparison in Table 2 uses THREADING. All variants receive the same task-specific frozen GR00T N1.6 checkpoint, RGB observations, proprioception, and complete base proposal. The proposal length is $H = 16$. The fixed-horizon sweep uses $\mathcal{K}_{\text{fixed}} = \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$, whereas the adaptive selectors use $\mathcal{K}_{\text{adapt}} = \{2, 4, 8, 16\}$. Table 6 summarizes the changes to the actor.

Table 6: Actor configurations for the controlled comparison. Fixed-k denotes a separately trained policy for each $k \in \mathcal{K}_{\text{fixed}}$. Horizon-only, Shared Residual, and AHR use the four-way selector over $\mathcal{K}_{\text{adapt}}$. All variants use the common critic and execution interface described below.

Variant	Residual generation	Horizon selection	Actor update
Fixed-k	One fixed-length residual actor outputting $k \times d_a$; no horizon embedding	Constant k; selector disabled	Residual branch at k
Horizon-only	Residual output fixed to zero	Learned four-way categorical selector	Selector branch
Shared Residual	One 16-step residual, independent of k; candidates take prefixes	Same selector as AHR	Residual and selector branches jointly
AHR	One shared decoder conditioned on the candidate horizon	Learned four-way categorical selector	Residual and selector branches jointly

Fixed horizons and Best Fixed. We train nine independent policies, Fixed-1, Fixed-2, Fixed-4, Fixed-6, Fixed-8, Fixed-10, Fixed-12, Fixed-14, and Fixed-16. For Fixed-k, the residual actor outputs exactly k commands and has no horizon-selection head or horizon embedding. The actor corrects the first k commands of each new 16-step proposal. Its actor objective contains the single candidate k, and its target actor also uses k at the successor state. Thus, the residual policy is trained for its prescribed execution length, rather than obtained by fixing the selector of a previously trained adaptive policy. Best Fixed is the validation-selected member of these nine independently trained policies. The fixed-horizon sweep uses the broader set $\mathcal{K}_{\text{fixed}} = \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$, whereas the adaptive variants use $\mathcal{K}_{\text{adapt}} = \{2, 4, 8, 16\}$.

Horizon-only. We retain AHR's four-way selector and critic and bypass the residual decoder. For every candidate horizon, the command sequence is the unmodified base prefix. The selector is optimized with the same probability-weighted candidate-value objective as AHR, with the residual branch fixed at zero. Initial collection and subsequent exploration vary the selected horizon; neither adds residual action noise. Target construction likewise uses unmodified base prefixes without residual smoothing. During collection, the selector samples a horizon; during evaluation, it chooses the highest-probability horizon.

Shared Residual. We remove the candidate-horizon input from the residual decoder and generate one 16-step residual sequence per observation and base proposal. For the Threading comparison, the four candidates take its 2-, 4-, 8-, and 16-step prefixes. The observation encoder, actor trunk, categorical selector, and joint actor objective follow AHR. Both the residual generator and selector receive gradients from that objective: the selector weights candidate values, while residual gradients from overlapping prefixes accumulate in the shared sequence. For target smoothing, the target actor perturbs the single residual sequence before extracting candidate prefixes. Horizon selection therefore uses the same learned mechanism as AHR; the changed component is whether residual generation receives the candidate horizon.

AHR. The shared observation representation is passed to one residual decoder together with each candidate's learned horizon embedding. For the Threading comparison, the decoder produces a correction for each of the four horizons, and the categorical selector assigns probabilities to the resulting correction units. The residual decoder is shared across horizons; AHR does not train four independent residual networks. The residual and selector branches are jointly optimized using Eq. 5.

Execution and value learning. Each variant commits to its selected k-step segment, unless the episode terminates earlier. On completing the segment, it obtains a fresh observation and base proposal and discards the old suffix. This refresh rule applies to Fixed-k, Horizon-only, Shared Residual, and AHR. Residual-bearing variants use the same residual bounds, exploration-noise rule, and normalized action clipping as AHR.

All variants use the same critic architecture and input convention: the selected command sequence is zero-padded to 16 steps and accompanied by the horizon encoding. The command sequence, accumulated reward, terminal indicator, and realized duration are stored using Eq. 3. The critic loss, ensemble-head sampling, and duration-dependent discount are unchanged. For the Threading comparison, target candidates are generated by the corresponding variant: one corrected k-step candidate for Fixed-k, four unmodified prefixes for Horizon-only, four prefixes of one corrected sequence for Shared Residual, and four horizon-conditioned corrections for AHR. The target-policy expectation is evaluated over that candidate set.

Replay and optimization budget. For the Threading comparison, all variants use the same raw demonstration episodes and follow the offline replay construction and sampling protocol in Appendix A.2. Fixed-k uses the pool $\mathcal{D}_k$ associated with its fixed execution horizon. Horizon-only, Shared Residual, and AHR share the same candidate-horizon pools and use the same duration-balanced offline sampling rule. Demonstration commands remain expert commands for all variants, including Horizon-only. Each training run collects its own online transitions and uses the same minibatch size, with equal numbers of offline and online transitions. The visual/state encoder and actor/critic trunk widths are held fixed; only the actor heads change as listed in Table 6.

The critic warm-start and subsequent update schedule follow Algorithm 1. After a segment of realized duration n_t, each run receives gn_t critic-update credit, with an actor update every d_π critic updates. Matching updates per primitive environment step keeps short-horizon policies from receiving more updates merely because they make more decisions. All runs use the same task budget, training seeds, and evaluation protocol in Appendix B.

Fixed-horizon selection and budget accounting. Every Fixed-k candidate receives the full per-run training budget and the same validation-rollout budget. We aggregate validation success across the five training seeds, select one task-level horizon $k^\star$, and freeze it before evaluating the selected policy on disjoint final rollouts. Final-rollout scores are never used for horizon selection. For the Threading comparison, the complete fixed-horizon sweep comprises nine training runs per seed; each adaptive variant comprises one. This is 9×1M primitive training steps for the sweep versus 1M for an individual policy. The adaptive selector remains four-way over $\mathcal{K}_{\text{adapt}}$. Best Fixed reports the selected policy's performance at its own 1M-step budget; the additional sweep cost is the cost of selecting the constant horizon. This Fixed-k sweep is a controlled variant of the AHR learning framework and is separate from the DICE-RL baseline configuration in Appendix B.6.

Table 7: Fixed-horizon sweep on THREADING. Each column is one independently trained Fixed-k policy; entries report final success as mean $\pm$ standard deviation across five seeds. The selected $k^\star = 4$ column is shaded and boldfaced, and defines Best Fixed for the controlled comparison.

Fixed horizon k	1	2	4	6	8	10	12	14	16
Success (%)	84.6 ± 2.7	87.4 ± 2.7	**90.2 ± 2.8**	88.6 ± 2.1	87.0 ± 2.5	85.4 ± 2.1	84.4 ± 2.4	82.8 ± 2.2	80.8 ± 2.4

Threading fixed-horizon result. Aggregating the validation rollouts across the five training seeds selects $k^\star = 4$. Its disjoint final evaluation is $90.2 \pm 2.8\%$, compared with $84.6 \pm 2.7\%$ for Fixed-1, the per-step proposal-refresh setting. *Best Fixed* in Table 2 therefore denotes the independently trained Fixed-4 policy evaluated at the same final rollout budget as the other variants.

C.1.2 TASK-SPECIFIC CONTROLLED CORRECTION-UNIT ANALYSES

We extend the controlled correction-unit analysis to the three ROBOMIMIC-PIXEL tasks: SQUARE, TRANSPORT, and TOOL HANG. The common construction of Fixed horizon, Horizon-only, Shared Residual, and AHR is given in Appendix C.1.1. All task-specific comparisons use the same task-adapted frozen GR00T N1.6 checkpoint and proposal length $H = 16$. The fixed-horizon sweep uses $\mathcal{K}_{\text{fixed}} = \{1, 2, 4, 6, 8, 10, 12, 14, 16\}$, whereas the adaptive variants retain the four-way selector $\mathcal{K}_{\text{adapt}} = \{2, 4, 8, 16\}$. All configurations use the same transition schema, replay mixture, primitive-step budget, and evaluation protocol, with five training seeds per setting. Fixed-k policies are trained independently for every $k \in \mathcal{K}_{\text{fixed}}$; Horizon-only, Shared Residual, and AHR select only from $\mathcal{K}_{\text{adapt}}$.

For each task, the fixed-horizon table reports final success for all nine independently trained Fixed-k policies. We aggregate validation success over the five training seeds, select one task-level $k^\star$, freeze that choice, and evaluate it once on disjoint final rollouts; final scores never enter the selection rule. The variant table uses the selected policy as Best Fixed and compares it with Horizon-only, Shared Residual, and AHR under the common four-way adaptive candidate set. Best Fixed is selected independently for each task from the nine-horizon sweep.

SQUARE. The Square sweep selects $k^\star = 4$. The table has a broad local maximum around $k = 2$–6, while the shorter $k = 1$ and longer $k \geq 8$ settings are lower.

Table 8: Fixed-horizon sweep on SQUARE. Entries are final success (mean $\pm$ standard deviation) across five seeds. The selected $k^\star = 4$ column is shaded and boldfaced.

k	1	2	**4**	6	8	10	12	14	16
Success (%)	94.4 ± 1.1	96.4 ± 1.1	$\mathbf{97.2 \pm 0.8}$	96.6 ± 0.9	95.4 ± 1.1	94.4 ± 1.1	93.4 ± 1.5	92.2 ± 1.5	91.4 ± 1.5

Table 9: Controlled correction-unit variants on SQUARE. Best Fixed is the independently trained Fixed-4 policy selected in Table 8.

Variant	Residual	Adaptive horizon	Horizon-conditioned residual	Success (%)
Frozen Base	×	×	×	81.0 ± 2.6
Best Fixed ($k^\star$)	✓	×	×	97.2 ± 0.8
Horizon-only	×	✓	×	90.6 ± 1.8
Shared Residual	✓	✓	×	98.0 ± 0.7
AHR	✓	✓	✓	$\mathbf{99.2 \pm 0.4}$

Task-specific interpretation. Square's Best Fixed result is $97.2 \pm 0.8\%$ at $k^\star = 4$. AHR reaches $99.2 \pm 0.4\%$, an improvement of $+2.0$ percentage points. The small gap is consistent with Square being close to the success ceiling; frequent replanning at $k = 1$ and delayed feedback at $k \geq 8$ both reduce the fixed-horizon result.

TRANSPORT. Transport selects $k^\star = 8$. The maximum is broad: $k = 6$, $k = 8$, and $k = 10$ remain close, which is consistent with the coordinated transport phase benefiting from several consecutive corrected commands.

Table 10: Fixed-horizon sweep on TRANSPORT. Entries are final success (mean $\pm$ standard deviation) across five seeds. The selected $k^\star = 8$ column is shaded and boldfaced.

k	1	2	4	6	**8**	10	12	14	16
Success (%)	80.6 ± 1.9	84.0 ± 2.5	87.2 ± 2.3	89.4 ± 2.1	$\mathbf{90.8 \pm 1.9}$	90.0 ± 1.6	88.6 ± 1.9	87.0 ± 2.3	85.8 ± 2.6

Table 11: Controlled correction-unit variants on TRANSPORT. Best Fixed is the independently trained Fixed-8 policy selected in Table 10.

Variant	Residual	Adaptive horizon	Horizon-conditioned residual	Success (%)
Frozen Base	×	×	×	52.4 ± 3.4
Best Fixed ($k^\star$)	✓	×	×	90.8 ± 1.9
Horizon-only	×	✓	×	66.8 ± 3.1
Shared Residual	✓	✓	×	92.6 ± 1.5
AHR	✓	✓	✓	$\mathbf{95.0 \pm 0.7}$

Task-specific interpretation. Transport's Best Fixed result is $90.8 \pm 1.9\%$ at $k^\star = 8$. AHR reaches $95.0 \pm 0.7\%$, an improvement of $+4.2$ percentage points. The broad optimum over $k = 6$–10 indicates that the result is associated with the duration of the coordinated transfer phase rather than a single sharply tuned horizon.

TOOL HANG. Tool Hang selects $k^\star = 4$. Its fixed results peak in the short-to-medium horizon range and decline for $k \geq 10$, matching the need for frequent feedback during fine interaction.

Table 12: Fixed-horizon sweep on TOOL HANG. Entries are final success (mean ± standard deviation) across five seeds. The selected $k^\star = 4$ column is shaded and boldfaced.

k	1	2	**4**	6	8	10	12	14	16
Success (%)	84.8 ± 2.6	88.0 ± 2.5	$\mathbf{89.6 \pm 1.9}$	88.8 ± 1.9	87.6 ± 1.9	86.0 ± 2.4	84.6 ± 1.9	83.0 ± 2.4	81.6 ± 2.1

Table 13: Controlled correction-unit variants on TOOL HANG. Best Fixed is the independently trained Fixed-4 policy selected in Table 12.

Variant	Residual	Adaptive horizon	Horizon-conditioned residual	Success (%)
Frozen Base	×	×	×	38.0 ± 4.0
Best Fixed ($k^\star$)	✓	×	×	89.6 ± 1.9
Horizon-only	×	✓	×	50.8 ± 3.7
Shared Residual	✓	✓	×	91.4 ± 1.8
AHR	✓	✓	✓	$\mathbf{94.0 \pm 1.2}$

Task-specific interpretation. Tool Hang's Best Fixed result is $89.6 \pm 1.9\%$ at $k^\star = 4$. AHR reaches $94.0 \pm 1.2\%$, an improvement of $+4.4$ percentage points. This task therefore benefits from horizon-conditioned corrections while retaining frequent refinement during contact-sensitive interaction.

Cross-task summary. Table 14 collects the selected fixed horizon and the corresponding AHR gain for all four controlled tasks. The fixed reference is selected separately from each task's nine-policy sweep; the AHR column is the main benchmark result for the same task.

Table 14: Cross-task fixed-horizon selection and correction-unit results. Success values are mean ± standard deviation; the last column is the AHR–Best Fixed difference in percentage points.

Task	Best k	Frozen Base (%)	Horizon-only (%)	Best Fixed (%)	Shared Residual (%)	AHR (%)	AHR–Fixed (pp)
SQUARE	**4**	81.0 ± 2.6	90.6 ± 1.8	97.2 ± 0.8	98.0 ± 0.7	$\mathbf{99.2 \pm 0.4}$	$+2.0$
TRANSPORT	**8**	52.4 ± 3.4	66.8 ± 3.1	90.8 ± 1.9	92.6 ± 1.5	$\mathbf{95.0 \pm 0.7}$	$+4.2$
TOOL HANG	**4**	38.0 ± 4.0	50.8 ± 3.7	89.6 ± 1.9	91.4 ± 1.8	$\mathbf{94.0 \pm 1.2}$	$+4.4$
THREADING	**4**	72.0 ± 3.8	83.8 ± 3.3	90.2 ± 2.8	91.8 ± 1.9	$\mathbf{96.0 \pm 2.4}$	$+5.8$

C.1.3 SHARED-PREFIX COMPATIBILITY IN JOINT AMPLITUDE–HORIZON LEARNING

Shared Residual and AHR differ in whether candidate horizons are required to use identical corrections on overlapping commands. We characterize this distinction for the policy-weighted candidate objective in Eq. 5. The analysis is per refinement input and compares the residual outputs admissible under the two representations; AHR may still share network parameters across horizons, as in our implementation.

Setup. Fix a refinement input $x = (s, A^b)$ and the current parameters of Q_{avg}. Let $\mathcal{K} \subseteq \{1, \dots, H\}$ be a nonempty candidate-horizon set. Let $\mathcal{U} \subset \mathbb{R}^{d_a}$ be a nonempty compact per-command residual domain and define

$$\mathcal{R}_k = \mathcal{U}^k.$$

For $r \in \mathcal{R}_H$, let

$$P_k r = r_{1:k}$$

denote prefix extraction.

For a horizon-selection distribution

$$p = (p_k)_{k\in\mathcal{K}}, \qquad p_k \geq 0, \qquad \sum_{k\in\mathcal{K}} p_k = 1,$$

define

$$q_k(z) = Q_{\text{avg}}\big(s, \text{clip}(A^b_{1:k} + z), k\big), \qquad z \in \mathcal{R}_k.$$

Assume that each q_k is continuous on $\mathcal{R}_k$. Define

$$J_{\text{sh}}(r; p) = \sum_{k\in\mathcal{K}} p_k q_k(P_k r), \qquad r \in \mathcal{R}_H,$$

and

$$J_{\text{hc}}(z; p) = \sum_{k\in\mathcal{K}} p_k q_k(z_k), \qquad z = (z_k)_{k\in\mathcal{K}} \in \prod_{k\in\mathcal{K}} \mathcal{R}_k. \tag{6}$$

Shared Residual imposes the equalities

$$z_k = P_k r, \qquad \forall k \in \mathcal{K},$$

whereas the horizon-conditioned output space removes these shared-prefix equalities. This comparison isolates the output constraint introduced by a horizon-independent residual; it does not require independently parameterized networks for different horizons.

Proposition C.1 (Shared-prefix compatibility and optimality gap). For each $k \in \mathcal{K}$, define

$$M_k = \max_{z\in\mathcal{R}_k} q_k(z), \qquad \mathcal{M}_k = \arg\max_{z\in\mathcal{R}_k} q_k(z).$$

Let

$$V_{\text{sh}}(p) = \max_{r\in\mathcal{R}_H} J_{\text{sh}}(r; p),$$

and

$$V_{\text{hc}}(p) = \max_{z\in\prod_{k\in\mathcal{K}}\mathcal{R}_k} J_{\text{hc}}(z; p).$$

Then, for every horizon-selection distribution p,

$$V_{\text{hc}}(p) = \sum_{k\in\mathcal{K}} p_k M_k,$$

and

$$\boxed{G(p) := V_{\text{hc}}(p) - V_{\text{sh}}(p) = \min_{r\in\mathcal{R}_H} \sum_{k\in\mathcal{K}} p_k \left[M_k - q_k(P_k r)\right] \geq 0.} \tag{7}$$

Let

$$\mathcal{K}_+(p) = \{k \in \mathcal{K} : p_k > 0\}.$$

Then

$$\boxed{G(p) = 0 \iff \exists r \in \mathcal{R}_H \text{ such that } P_k r \in \mathcal{M}_k, \quad \forall k \in \mathcal{K}_+(p).} \tag{8}$$

Hence the shared-prefix constraint is lossless exactly when the active horizon-specific optima admit a common prefix-compatible realization. Otherwise,

$$G(p) > 0.$$

Proof. Because $\mathcal{U}$ is nonempty and compact, every finite product $\mathcal{R}_k$, $\mathcal{R}_H$, and $\prod_{k\in\mathcal{K}} \mathcal{R}_k$ is nonempty and compact. The prefix maps P_k are continuous, and each q_k is continuous by assumption. Hence all extrema below are attained.

In the horizon-conditioned output space, the variables z_k may be selected independently. Choosing

$$z_k \in \mathcal{M}_k$$

for every k gives

$$V_{\mathrm{hc}}(p) = \sum_{k\in\mathcal{K}} p_k M_k.$$

Therefore,

$$\begin{aligned} G(p) &= \sum_{k\in\mathcal{K}} p_k M_k - \max_{r\in\mathcal{R}_H} \sum_{k\in\mathcal{K}} p_k q_k(P_k r) \\ &= \min_{r\in\mathcal{R}_H} \sum_{k\in\mathcal{K}} p_k \left[M_k - q_k(P_k r)\right]. \end{aligned}$$

Since

$$q_k(P_k r) \le M_k$$

for every k and r, every summand is nonnegative, proving $G(p) \ge 0$.

If there exists an $r \in \mathcal{R}_H$ such that

$$P_k r \in \mathcal{M}_k, \qquad \forall k \in \mathcal{K}_+(p),$$

then every positively weighted term in Eq. 7 is zero and therefore $G(p) = 0$.

Conversely, suppose $G(p) = 0$, and let $r^\star$ attain the minimum in Eq. 7. Since Eq. 7 is a finite sum of nonnegative terms, the sum can equal zero only if

$$M_k - q_k(P_k r^\star) = 0, \qquad \forall k \in \mathcal{K}_+(p).$$

Equivalently,

$$P_k r^\star \in \mathcal{M}_k, \qquad \forall k \in \mathcal{K}_+(p).$$

This proves Eq. 8. If no such shared residual exists, the attained nonnegative minimum cannot equal zero, and hence $G(p) > 0$. □

The condition is defined on the maximizing sets $\mathcal{M}_k$, rather than on an arbitrary choice of maximizer. Thus, unique maximizers that disagree on any overlapping command are sufficient for a strict gap, whereas nonunique maximizers—including value-equivalent residuals induced by clipping—may still admit a prefix-compatible optimum.

In AHR, $p_\theta(\cdot \mid x)$ is produced by a softmax selector. At finite logits, every candidate therefore has positive probability, so Proposition C.1 characterizes whether the candidate-specific residual objectives at the current actor update can be optimized simultaneously under a shared-prefix representation.

Joint actor update. The same shared-prefix structure appears in the actor gradient. Let

$$p_{\theta,k} = p_\theta(k \mid x).$$

For Shared Residual, write

$$r_\theta = r_\theta(x),$$

whereas for the horizon-conditioned actor write

$$z_{\theta,k} = f_\theta(x; k).$$

Substituting these parameterized outputs into Eq. 6 gives, at differentiability points,

$$\nabla_\theta J_{\mathrm{sh}} = \sum_{k\in\mathcal{K}} q_k(P_k r_\theta)\nabla_\theta p_{\theta,k} + (D_\theta r_\theta)^\top \sum_{k\in\mathcal{K}} p_{\theta,k} P_k^\top \nabla q_k(P_k r_\theta),$$

and

$$\nabla_\theta J_{\mathrm{hc}} = \sum_{k\in\mathcal{K}} q_k(z_{\theta,k})\nabla_\theta p_{\theta,k} + \sum_{k\in\mathcal{K}} p_{\theta,k}(D_\theta z_{\theta,k})^\top \nabla q_k(z_{\theta,k}). \tag{9}$$

Here D_θ denotes the corresponding output Jacobian, and $P_k^\top$ embeds a k-step residual gradient into the H-step residual space by zero-padding the unused suffix. Equation 9 follows directly from the product and chain rules using

$$D_\theta(P_k r_\theta) = P_k D_\theta r_\theta.$$

For Shared Residual, horizon-specific residual gradients on overlapping commands are first accumulated in the same residual-output coordinates:

$$\sum_{k\in\mathcal{K}} p_{\theta,k} P_k^\top \nabla q_k(P_k r_\theta).$$

Consequently, disagreeing horizon-specific gradients can partially or completely cancel before propagation through $D_\theta r_\theta$. In the horizon-conditioned actor, the corresponding gradients propagate through candidate-dependent outputs $z_{\theta,k}$. The candidates may still share network parameters, but they are not constrained to use identical residual variables on overlapping commands.

The selector-gradient term remains present in both architectures. The minibatch actor gradient is the negative average of these per-input gradients over the replay batch.

Corollary C.2 (Single-horizon limit). Let

$$\Delta_{\mathcal{K}} = \left\{ p : p_k \geq 0,\ \sum_{k\in\mathcal{K}} p_k = 1 \right\}.$$

If both the selector distribution and residual outputs are optimized freely for the fixed input, then

$$\boxed{\max_{p\in\Delta_{\mathcal{K}}} V_{\mathrm{sh}}(p) = \max_{p\in\Delta_{\mathcal{K}}} V_{\mathrm{hc}}(p) = \max_{k\in\mathcal{K}} M_k.} \tag{10}$$

Proof. For every $p \in \Delta_{\mathcal{K}}$,

$$V_{\mathrm{hc}}(p) = \sum_{k\in\mathcal{K}} p_k M_k \leq \max_{k\in\mathcal{K}} M_k,$$

and

$$V_{\mathrm{sh}}(p) \leq V_{\mathrm{hc}}(p).$$

Let

$$k^\star \in \arg\max_{k\in\mathcal{K}} M_k$$

and choose

$$z^\star \in \mathcal{M}_{k^\star}.$$

Because $\mathcal{U}$ is nonempty, $z^\star$ can be extended to some $r^\star \in \mathcal{R}_H$ satisfying

$$P_{k^\star} r^\star = z^\star.$$

Taking

$$p_{k^\star} = 1$$

therefore yields

$$V_{\mathrm{sh}}(p) = V_{\mathrm{hc}}(p) = M_{k^\star}.$$

Hence Eq. 10 follows. □

Corollary C.2 is the single-candidate limit of Proposition C.1: once all selection mass is placed on one horizon, there is no longer a multi-horizon prefix-compatibility requirement. During the joint update in Eq. 5, by contrast, the current softmax distribution assigns positive weight to all candidate units, and Proposition C.1 characterizes whether their residual objectives admit a common optimum.

Relation to the controlled comparison. Shared Residual and AHR use the same candidate horizons, selector, critic, execution interface, and joint actor objective; the controlled change is whether the residual output is constrained to share prefixes across horizons. Proposition C.1 and Eq. 9 characterize the resulting difference in the candidate objective and residual-gradient structure, while Tables 2 and 8–12 evaluate its effect under finite-network actor–critic learning.

C.2 Additional Ablation Studies

Figure 7 presents the seven-task comparison for frozen base checkpoints.

Base-Policy Training Steps. We consider frozen base policies obtained after 5k, 10k, 20k, and 30k training steps, with the 30k checkpoint as the default. The horizontal axis reports environment interactions during subsequent refinement.

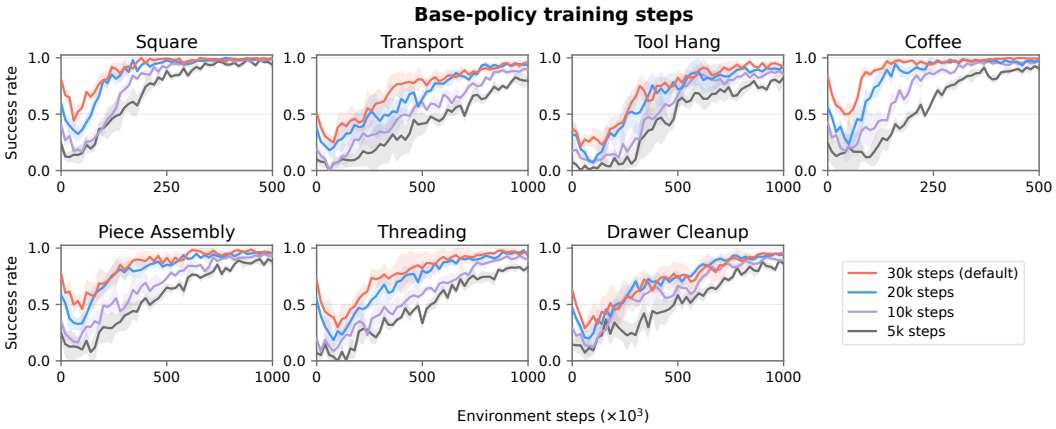


Figure 7: Frozen base-policy training steps across seven simulation tasks. The legend identifies the checkpoint used by each curve.

We study training design choices across all seven simulation tasks using the protocols in Appendix B. Figures 8 and 9 show means and one sample standard deviation over five seeds.

Critic Ensemble Size. We compare $N \in \{2, 5, 10\}$ critic heads, with $N = 10$ as the default, holding the shared trunk and update budget fixed. Targets use the minimum of two distinct randomly sampled heads, and the actor uses all heads' mean; both heads are selected when $N = 2$. Figure 8 shows that increasing N from two to five mainly benefits Transport, Tool Hang, and Threading. With five heads, final success rates are within 1.3 percentage points of ten heads across all tasks, while Square and Coffee converge to similar performance. Gains from five to ten heads are smaller.

Table 15: Rollout-level execution and inference analysis on SQUARE. Best Fixed uses the validation-selected fixed correction horizon $k = 4$, while AHR adaptively selects the correction horizon from $\{2, 4, 8, 16\}$. Simulation runs at 20 Hz. Total rollout time is computed as primitive-action execution time plus total policy inference time under synchronous inference.

Method	Success (%)	Primitive steps	Refinement decisions	Execution time	Policy inference	Total rollout time
Best Fixed correction ($k = 4$)	97.2 ± 0.8	124	31	6.200 s	1.973 s	8.173 s
AHR	$\mathbf{99.2 \pm 0.4}$	112	13	5.600 s	0.861 s	**6.461** s

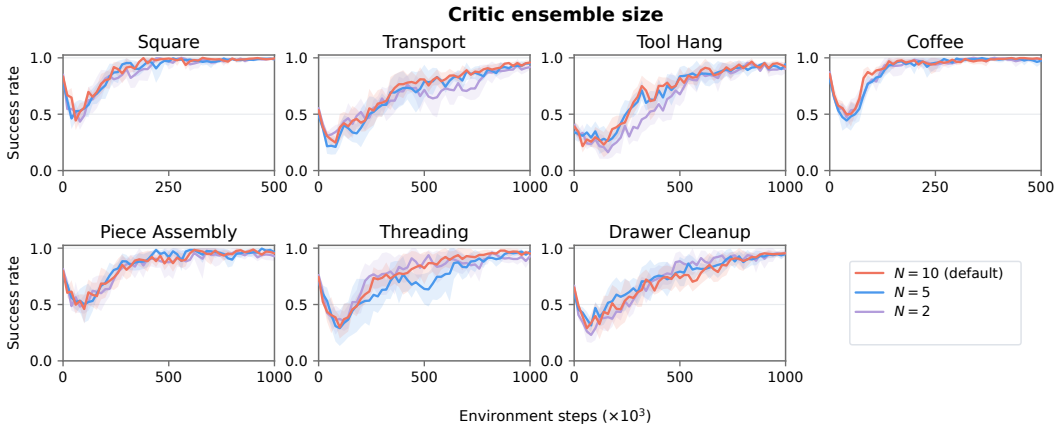


Figure 8: Critic ensemble size across seven simulation tasks.

Number of Gradient Updates. We compare $g \in \{0.25, 0.5, 1\}$ critic updates per primitive environment step, with $g = 1$ as the default, holding the critic ensemble, actor-update interval, and interaction budget fixed. Lower budgets generally delay recovery after the initial drop in Figure 9. Reducing g from 1 to 0.25 lowers final success from 95.0% to 79.4% on Transport and from 94.0% to 82.0% on Tool Hang; Square and Coffee remain above 97%. The intermediate budget stays close to the default on Square, Coffee, and Piece Assembly, with larger gaps on the remaining tasks.

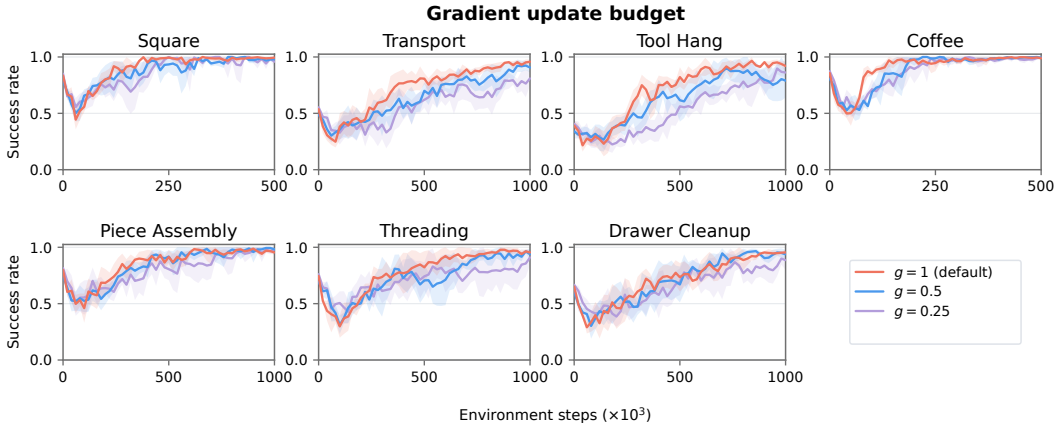


Figure 9: Critic updates per primitive environment step across seven tasks.

C.3 INFERENCE COST ANALYSIS

With the final trained weights on an RTX 5090, the mean inference latency is 4.7 ms per AHR decision and 61.5 ms per GR00T N1.6 proposal. The AHR measurement covers the refinement module, including horizon selection; the critic is not used during deployment.

Adaptive correction horizons reduce end-to-end execution cost. Best Fixed applies residual corrections at a fixed correction horizon of $k = 4$, executing four corrected commands before ev-

ery subsequent refinement decision. On SQUARE, this results in 31 refinement decisions over 124 primitive control steps. AHR instead adapts the temporal extent of each correction according to the current execution state and completes the rollout with 13 refinement decisions over 112 primitive steps.

At the 20 Hz control rate, the corresponding action-execution times are 6.200 s for Best Fixed and 5.600 s for AHR. Including frozen-base and refinement inference, the total policy-inference costs are 1.973 s and 0.861 s, respectively. Under synchronous inference, the resulting end-to-end rollout times are therefore 8.173 s for Best Fixed and 6.461 s for AHR (Table 15).

AHR thus reduces refinement decisions by 58.1%, policy-inference time by 56.4%, and total rollout time by 20.9%, while increasing task success from 97.2% to 99.2%. Importantly, this gain does not arise from uniformly increasing the correction horizon. AHR retains short-horizon corrective feedback around precision-sensitive interactions while using longer corrected segments when extended execution is reliable. The results therefore indicate that repeatedly applying corrections at a fixed temporal resolution introduces unnecessary refinement and inference overhead, whereas state-dependent correction horizons improve both execution efficiency and task performance.

D Real-World Experimental Details

D.1 Hardware, Tasks, and Reset Procedure

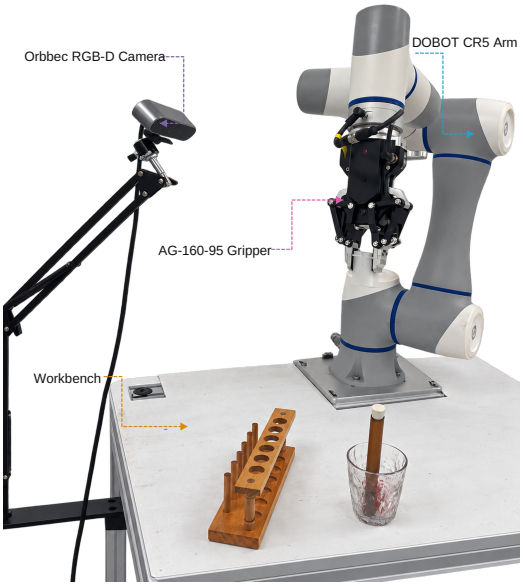


Figure 10: Real-world experimental setup.

Experiments use a Dobot CR5 arm equipped with an AG-160-95 parallel-jaw gripper and a fixed Orbbec Astra RGB-D camera. Policies receive the RGB stream and robot proprioception; depth is not used. Inference runs on an NVIDIA GeForce RTX 5090. Cartesian end-effector commands are executed through the CR5 interface at a nominal rate of 20 Hz. One primitive environment step corresponds to one command executed at this task-control interface.

In cup stacking, the robot picks up a movable cup and places it on a target cup. In test-tube insertion, it grasps a tube and inserts it into a hole in a rack. Object poses are sampled within task-specific planar regions at the start of each attempt. Success requires the cup to remain stably stacked or the tube to remain properly seated after release.

An operator records the outcome and resets the robot and objects between attempts. Rewards are sparse and terminal: one for success and zero for failure or timeout. No corrective actions or additional demonstrations are provided during online refinement. All methods use the same initialization distribution, success and failure criteria, and task-specific deadline.

D.2 Base-Policy Adaptation and Online Refinement

Base policy. We adapt GR00T N1.6 on 110 demonstrations for cup stacking and 170 for test-tube insertion. The resulting task-specific checkpoints are frozen and shared by Base, PLD, and AHR. Base executes the policy without online refinement. No additional expert trajectories are collected; the original base-adaptation demonstrations are reprocessed into each method's offline replay representation.

Refinement. We independently train AHR and PLD five times per task using the shared frozen VLA; every run collects its own online experience. AHR uses separate actor and learner processes following Ankile et al. (2026). The actor executes corrected segments and transfers completed rollouts to online replay. The learner updates AHR from mixed demonstration and online segment transitions using Algorithm 1.

PLD is trained and evaluated at its residual-specialist stage before distillation (Xiao et al., 2026). Its single-step specialist predicts an additive correction from RGB observations, proprioception, and the corresponding frozen-VLA action. Each method collects its own online experience and retains its replay, exploration, optimization, and proposal-refresh procedures. No distillation is performed in the PLD comparison.

Online interaction. Each independent training run receives a nominal budget of 60,000 primitive environment steps for stacking and 150,000 for insertion. Interaction is counted separately for each run as

$$N_{\text{env}} = \sum_{t} n_t,$$

where n_t is the number of commands actually executed in segment t. Initial online collection and all executed commands from successful, failed, and timed-out attempts contribute to this count.

Table 16: Real-world online interaction. Nominal budgets apply to each of the five independent training runs. Actual steps and rollout counts are from one run per method-task setting. Actual steps include the small overrun from completing the current collection unit; rollout counts denote completed training attempts.

Task	Method	Budget	Actual steps	Rollouts
Cup stacking	AHR	60,000	60,008	131
	PLD	60,000	60,132	122
Test-tube insertion	AHR	150,000	150,011	293
	PLD	150,000	150,074	254

Table 17: Real-world task success (%). Values are mean $\pm$ sample standard deviation across five independent training runs for AHR and PLD, with 50 matched evaluation trials per trained model. Base is evaluated on five matched batches without online refinement.

Task	Base	PLD specialist	AHR
Cup stacking	72.0 ± 3.2	91.2 ± 2.3	98.0 ± 1.4
Test-tube insertion	58.0 ± 3.2	79.2 ± 3.0	93.2 ± 2.3

Unexecuted proposal suffixes and replay-only updates do not add environment steps. Scene resets are outside the training attempts and are not used as learning experience. Rollout counts are recorded separately: matching primitive interaction need not yield the same number of task attempts.

Training endpoint. Each run is controlled by its task-specific nominal primitive-step budget. Because collection is asynchronous, the actor completes the current collection unit before stopping once the nominal budget has been reached. Table 16 reports the per-run nominal budget and actual interaction counts from one of the five runs in each method-task setting. Completed transitions from the final partial rollout are retained in replay; reaching the budget does not change their terminal flags. Rollout counts refer to completed attempts, with any final partial attempt recorded separately.

The learner completes the updates allocated by its specified training schedule, after which we freeze the final actor for evaluation. This endpoint rule is used for every run of both methods; final-test performance does not determine checkpoint selection.

D.3 Evaluation Protocol and Success Metric

AHR and PLD each produce five independently trained checkpoints per task, each evaluated on 50 trials. The frozen Base is evaluated on five matched batches of 50 trials. Within each run, all methods use the same pre-generated sequence of initial configurations. Evaluation disables exploration and parameter updates; its trajectories are excluded from training replay. Task conditions follow Appendix D.1. Failures and timeouts count as unsuccessful trials.

Let x_r be successes in 50 trials for trained model r, or matched evaluation batch r for Base. We report

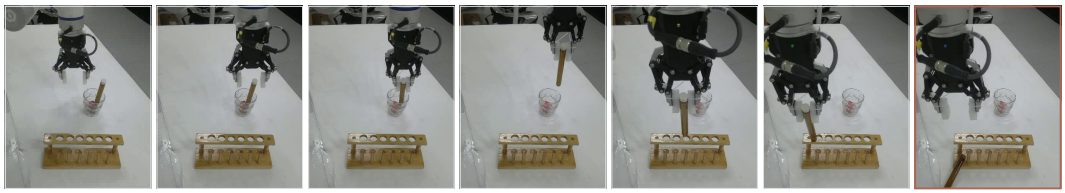
$$s_r = 100\frac{x_r}{50}, \qquad \bar{s} = \frac{1}{5}\sum_{r=1}^{5} s_r, \qquad s_{\mathrm{run}} = \sqrt{\frac{1}{4}\sum_{r=1}^{5}(s_r - \bar{s})^2}.$$

Table 17 reports $\bar{s} \pm s_{\mathrm{run}}$. The standard deviation summarizes variation in evaluation success across five independently trained models for AHR and PLD, and across five evaluation batches for Base.

D.4 Real-World Task Sequences

Figure 11 shows representative test-tube-insertion executions for Base and AHR; Figure 12 shows cup-stacking executions.

Base



AHR

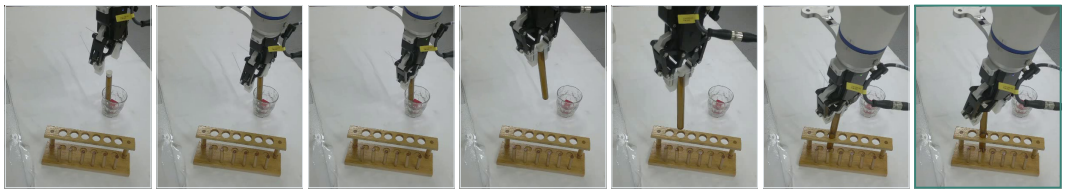
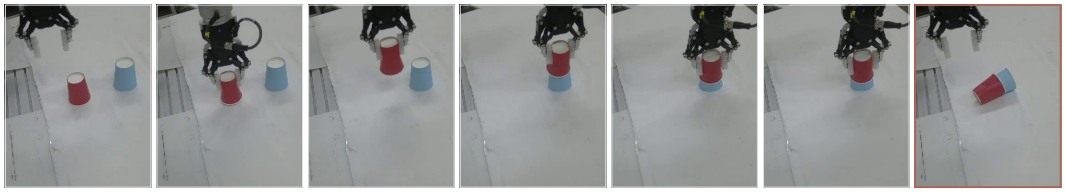


Figure 11: Real-world test-tube insertion sequences for Base (top) and AHR (bottom).

Base



AHR

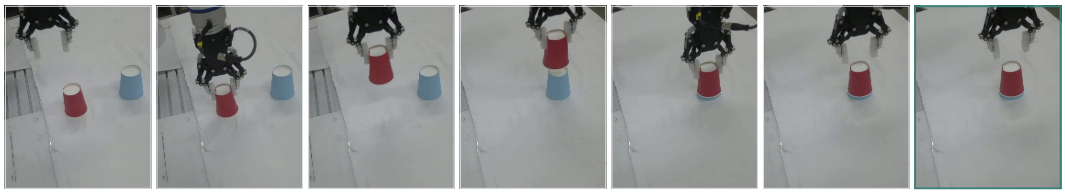


Figure 12: Real-world cup-stacking sequences for Base (top) and AHR (bottom).

E Reproducibility Details

E.1 Training Losses

The critic and actor objectives are given in Eqs. 4 and 5, respectively. For each critic update, one pair of target heads is sampled without replacement and shared across the minibatch and candidate horizons. All online critic heads are fitted to the same stop-gradient target, with squared TD errors averaged over heads and transitions. The actor uses the mean value of all critic heads and optimizes the policy-weighted value over all candidate correction units.

E.2 Network Architecture

AHR consists of a lightweight visual encoder, a horizon-conditioned residual actor, and an ensemble critic. At each refinement decision, the module receives the task-specific RGB observations, robot proprioception, and the complete frozen-VLA proposal $A_t^b \in \mathbb{R}^{H \times d_a}$.

Visual encoding. Each RGB view is resized to 256×256, converted to floating point, and normalized to $[-1, 1]$. We use a lightweight Vision Transformer (ViT) whose parameters are shared across

all camera views. Each image is partitioned into non-overlapping 16×16 patches, resulting in a 16×16 grid of 256 patch tokens. Patch embeddings have dimension 256 and are augmented with learned absolute positional embeddings.

The visual encoder contains four pre-LayerNorm Transformer blocks. Each block uses four self-attention heads and a two-layer feed-forward network with hidden dimension 1024 and GELU activations. We use neither attention dropout nor MLP dropout. After the final LayerNorm, patch tokens are mean-pooled to obtain a 256-dimensional representation $z_t^{(v)}$ for camera view v.

For V camera views, the per-view features are concatenated in a fixed task-specific camera order,

$$z_t^{\text{img}} = \left[z_t^{(1)}; \ldots; z_t^{(V)} \right] \in \mathbb{R}^{256V}. \tag{11}$$

We do not use an additional cross-view attention module. The same ViT weights are used for every camera stream. In our experiments, $V = 2$ for the RoboMimic-Pixel tasks and $V = 3$ for the DexMimicGen tasks.

The visual encoder is initialized from scratch. Linear layers, patch embeddings, and positional embeddings are initialized from a truncated normal distribution with standard deviation 0.02; linear biases are initialized to zero. LayerNorm scale and bias are initialized to one and zero, respectively.

Actor representation. The multi-view visual feature is concatenated with proprioception and the flattened complete base proposal,

$$x_t^{\text{act}} = \left[z_t^{\text{img}}; s_t^{\text{prop}}; \text{vec}(A_t^b) \right]. \tag{12}$$

A two-layer MLP with 1024 hidden units per layer, LayerNorm, and ReLU activations maps this input to a shared actor representation $h_t \in \mathbb{R}^{1024}$.

Horizon-conditioned residual decoder. Each candidate horizon $k \in \mathcal{K}$ is represented by a learned embedding $e_k \in \mathbb{R}^{32}$. A single residual decoder is shared across all candidate horizons. For each k, the decoder takes $[h_t; e_k] \in \mathbb{R}^{1056}$ as input and applies

$$\mathbb{R}^{1056} \to \mathbb{R}^{1024} \to \mathbb{R}^{Hd_a}. \tag{13}$$

The output is reshaped to an $H \times d_a$ residual tensor and bounded elementwise as

$$\widetilde{\Delta A}_t^{[k]} = \alpha \tanh\left(g_\theta(h_t, e_k) \right), \qquad \alpha = 0.1. \tag{14}$$

Only the first k residual commands are retained for candidate k,

$$\Delta A_t^{[k]} = \widetilde{\Delta A}_{t,1:k}^{[k]} \in \mathbb{R}^{k \times d_a}. \tag{15}$$

Thus, all candidate horizons share the decoder parameters, while their residual sequences are generated separately through the corresponding horizon embeddings. In particular, residuals on overlapping commands are not constrained to be identical across different horizons.

The corrected candidate is

$$C_t^{[k]} = \text{clip}\left(A_{t,1:k}^b + \Delta A_t^{[k]}, -1, 1 \right). \tag{16}$$

Positions beyond k are not part of the correction unit. Zero-padding to length H is applied only when a selected corrected segment is provided to the critic.

Horizon selection. A categorical prediction head operates on the shared actor representation h_t and produces

$$p_\theta(k \mid s_t, A_t^b) = \text{softmax}\left(g_{\text{hor}}(h_t) \right), \qquad k \in \mathcal{K}. \tag{17}$$

The selected horizon and its corresponding residual sequence jointly define the correction unit used for execution.

Table 18: AHR actor, visual encoder, and critic architecture. The visual encoder is shared across camera views, and the residual decoder is shared across candidate horizons while being conditioned on a learned horizon embedding.

Component	Configuration
Visual encoder	
RGB preprocessing	Resize each view to 256×256; convert to float and normalize to $[-1, 1]$
Patch embedding	16×16 non-overlapping patches; 256 patch tokens per view
Token dimension	256
Transformer	4 pre-LayerNorm blocks; 4 attention heads per block
Feed-forward network	$256 \to 1024 \to 256$, GELU
Dropout	None
Pooling	Mean pooling over patch tokens after final LayerNorm
Per-view feature	256 dimensions
Multi-view fusion	Concatenation in fixed camera order; shared ViT parameters across views
Initialization	From scratch; truncated-normal initialization with std. 0.02
Actor	
Actor input	Multi-view RGB feature, proprioception, and complete base proposal A_t^b
Shared actor trunk	2-layer MLP with 1024 hidden units per layer
Normalization / activation	LayerNorm / ReLU
Horizon embedding	Learned embedding, dimension 32
Residual decoder	One decoder shared across all $k \in \mathcal{K}$; $1056 \to 1024 \to Hd_a$
Residual output	Reshape to $H \times d_a$, apply $\alpha \tanh(\cdot)$, and retain the first k positions for candidate k
Residual scale	$\alpha = 0.1$
Horizon-selection head	Categorical head over $\lvert\mathcal{K}\rvert$ candidate horizons
Selection distribution	$p_\theta(k \mid s_t, A_t^b)$ via softmax
Critic	
Critic input	Multi-view RGB feature, proprioception, padded corrected segment C_t, and one-hot horizon encoding
Command representation	C_t zero-padded beyond k_t to proposal length H
Critic trunk	Shared representation across the ensemble
Critic ensemble	$N = 10$ separately parameterized scalar value heads
Per-head network	2-layer MLP with 1024 hidden units per layer
Normalization / activation	LayerNorm / ReLU
Critic output	$Q_{\phi_i}(s_t, C_t, k_t)$

Critic. The critic receives the same visual representation z_t^{img}, robot proprioception, the selected corrected segment C_t zero-padded beyond k_t to length H, and a one-hot encoding of k_t. A shared critic trunk feeds an ensemble of $N = 10$ independently parameterized scalar value heads. Each head uses a two-layer MLP with 1024 hidden units per layer, LayerNorm, and ReLU activations and predicts

$$Q_{\phi_i}(s_t, C_t, k_t). \tag{18}$$

The visual encoder is optimized through the critic objective. During actor optimization, visual features are detached, while gradients through the critic with respect to the candidate command inputs are retained. During target construction, the target actor and target critics reuse the online visual encoder without gradient tracking; no separate target visual encoder is maintained.

E.3 AHR Hyperparameters

Unless otherwise specified, the AHR settings in Table 19 are shared across simulation experiments. Task-specific environment-interaction budgets are reported in Appendix B.4, and base-VLA adaptation settings are reported separately in Appendix B.3.

Table 19: Default AHR training hyperparameters. Environment interaction and update budgets are measured in primitive environment steps.

Category	Hyperparameter	Value
Optimization	Optimizer	AdamW
	Actor learning rate	1×10^{-6}
	Critic learning rate	1×10^{-4}
	AHR visual-encoder learning rate	1×10^{-4}
	Gradient-norm clipping	1.0
Value learning	Primitive-step discount γ	0.99
	Critic ensemble size N	10
	Target heads per update	2
	Target-head sampling	Uniform, without replacement
	Target aggregation	Minimum of sampled heads
	Actor value aggregation	Mean over all N heads
	Target retention coefficient ρ	0.995
Update schedule	Critic updates per primitive step g	1
	Actor-update interval d_π	4 critic updates
	Critic-target update	Every critic update
	Actor-target update	Every actor update
Replay	Minibatch size	256 segment transitions
	Demonstration / online ratio	1:1
	Online replay capacity	200,000 segment transitions
	Initial online collection T_{warm}	10,000 primitive steps
	Critic-only warm-start	10,000 updates
Residual / exploration	Residual scale α	0.1
	Initial exploration bound η	0.2
	Collection-noise std. σ	0.05
	Target-smoothing std.	0.05 (0 during critic warm-start)
	Target-noise clipping c	0.3
	Corrected-command clipping	$[-1, 1]$
Candidate horizons	GR00T N1.6	$H = 16$, $\mathcal{K} = \{2, 4, 8, 16\}$
	$\pi_{0.5}$	$H = 10$, $\mathcal{K} = \{2, 3, 5, 10\}$

Update schedule. After critic warm-starting, we initialize the update credit $b = 0$ and global critic-update counter $m = 0$. Following a collected segment with realized duration n_t, the update budget is accumulated as

$$b \leftarrow b + gn_t, \qquad U_t = \lfloor b \rfloor, \qquad b \leftarrow b - U_t. \tag{19}$$

We perform U_t critic updates with fresh minibatches sampled equally from demonstration and online replay. The offline sampling rule in Appendix A.2 applies during both warm-starting and subsequent training. Each critic update increments m and updates the critic targets. The actor and its target are updated whenever m is divisible by d_π. The remaining update credit and global update counter are carried across segment and episode boundaries.

Candidate horizons. Candidate horizons are defined relative to the base-policy proposal length:

$$\mathcal{K}(H) = \left\{ \left\lceil \frac{H}{8} \right\rceil, \left\lceil \frac{H}{4} \right\rceil, \left\lceil \frac{H}{2} \right\rceil, H \right\}. \tag{20}$$

Duplicate values after rounding are retained only once. This gives $\mathcal{K} = \{2, 4, 8, 16\}$ for GR00T N1.6 with $H = 16$ and $\mathcal{K} = \{2, 3, 5, 10\}$ for $\pi_{0.5}$ with $H = 10$.