STAIRVLA: STAGE-AWARE HIERARCHICAL ACTION GENERATION FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models increasingly rely on diffusion- or flow-matching-based action heads to generate continuous robot actions. These action heads typically process the denoising trajectory in a largely uniform manner. However, we observe that the conditioning focus naturally shifts across denoising stages: early stages combine language instructions and visual observations to establish a coarse action trajectory, whereas later stages place greater emphasis on current visual observations for action alignment. Based on this insight, we introduce StairVLA, a stage-aware hierarchical action generation framework that uses partially denoised actions as a natural interface between coarse long-horizon action generation and local refinement. A high-level VLA performs early denoising to produce a reusable long-horizon partially denoised action trajectory, while a lightweight refiner operates at a higher frequency to refine local action chunks using the latest observations. This design amortizes expensive high-level VLA computation while preserving frequent closed-loop correction. On LIBERO, our GR00T-style instantiation improves average success from 96.5% to 97.8% while reducing amortized inference latency from 115.0 ms to 44.2 ms per action chunk. More broadly, across two VLA backbones, simulation benchmarks, and real-robot tasks, StairVLA consistently reduces inference cost while maintaining strong task performance.

1 INTRODUCTION

Vision-language-action (VLA) models have demonstrated strong performance across a wide range of robot manipulation tasks (Kim et al., 2024), combining vision-language understanding with generative action heads to translate multimodal context into continuous robot actions (Black et al., 2024; Bjorck et al., 2025). Specifically, these diffusion- or flow-matching-based action heads are conditioned on language instructions, visual observations, and robot states to progressively transform noise into executable action trajectories (Chi et al., 2025). Despite this progressive generation process, multimodal conditioning is typically applied throughout the entire denoising trajectory, leaving open whether different types of conditioning information should always be processed together during action generation.

Intuitively, language instructions typically specify task intent that remains stable over many control steps, whereas visual observations and robot states continuously change during execution. Task-level information therefore need not be processed as frequently as the feedback required for precise closed-loop control. Existing hierarchical VLAs exploit this temporal asymmetry by processing high-level information at a lower frequency and communicating it to higher-frequency controllers through subgoals, waypoints, macro-actions, or learned representations (Liu et al., 2024; Li et al., 2025; Wei et al., 2026; Du et al., 2026). Such hierarchies are typically introduced through externally designed representations or model components. This leads to our central question: *can such a hierarchy arise directly from the generative action process itself?*

In practice, we observe that the denoising trajectory itself already exhibits such a structure. As illustrated in Fig. 1, substantial action structure emerges well before denoising is complete, while the interaction between evolving action and multimodal conditioning continues to change throughout

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.

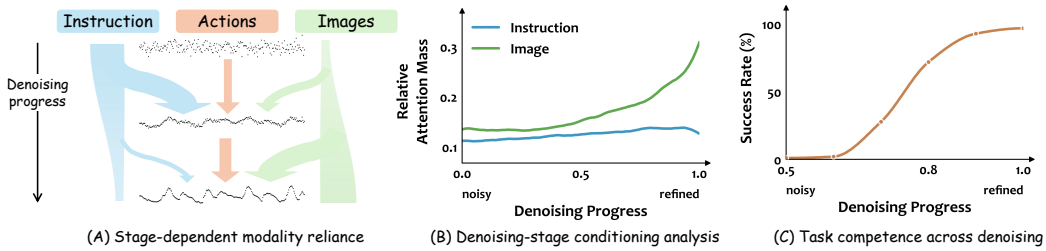


Figure 1: Stage dependence in generative VLA action formation. **(A)** Conceptual illustration of the evolving conditioning focus across denoising stages, from joint instruction–observation conditioning toward stronger visual alignment. **(B)** Relative attention to instruction and image tokens across denoising stages in π_0 on LIBERO, showing an increasing emphasis on visual observations during later refinement. **(C)** Task success of partially denoised actions generated by GR00T on LIBERO, showing that substantial task competence emerges before denoising is complete.

denoising stages. Intermediate actions are therefore not merely noisy versions of the final prediction: they already encode meaningful structure, even as subsequent denoising continues to reshape them toward execution. This reveals a natural stage dependence in generative VLA action formation, with different denoising stages serving distinct functional roles. This behavior is also consistent with the coarse-to-fine progression commonly observed in generative denoising (Balaji et al., 2022; Falck et al., 2025).

This observation suggests a direct way to exploit the hierarchy already present in denoising: expensive task conditioning can be concentrated in less frequent early-stage generation, while later stages perform frequent local refinement using up-to-date observations. However, this separation raises a key challenge: how can the refiner preserve task information without repeatedly processing the full instruction? Our key insight is that partially denoised actions themselves can bridge the two stages. By the end of early denoising, they already encode task-relevant action structure while remaining open to further refinement. The low-level refiner can therefore operate directly on these intermediate actions together with the latest observation, preserving high-level task information without repeatedly processing the full instruction.

Based on this insight, we introduce **StairVLA**, a stage-aware hierarchical action generation framework. A high-level VLA performs the early denoising stages over a long-horizon action trajectory and outputs partially denoised actions that are cached across multiple control steps. For each local chunk, a lightweight refiner takes the corresponding local action segment together with the latest observation and cached high-level context, and completes the remaining refinement before execution. This allows expensive task-conditioned generation to be reused across multiple control steps while keeping local actions responsive to the evolving environment. Importantly, the two stages communicate directly through intermediate states of the generative action trajectory, rather than through an externally designed action abstraction.

We evaluate our approach across two VLA backbones, simulation benchmarks, and real-robot manipulation tasks. On LIBERO, our GR00T-style instantiation improves average success from 96.5% to 97.8% while reducing amortized inference latency from 115.0 ms to 44.2 ms per action chunk, with consistent performance-efficiency gains on a π-style VLA backbone. Although performance gradually degrades as the reuse horizon increases, further analysis shows that StairVLA maintains strong performance over longer high-level reuse horizons than prior hierarchical VLA methods.

In summary, our contributions are threefold:

- We reveal stage dependence in generative VLA action formation, showing that task-relevant action structure emerges progressively while the role of multimodal conditioning evolves throughout denoising.
- We introduce StairVLA, a stage-aware hierarchical action generation framework that uses partially denoised actions as a natural interface between coarse long-horizon action generation and local observation-conditioned refinement.
- We validate the proposed design across two generative VLA backbones, simulation benchmarks, and real-robot tasks, demonstrating substantially reduced inference cost while maintaining strong task performance.

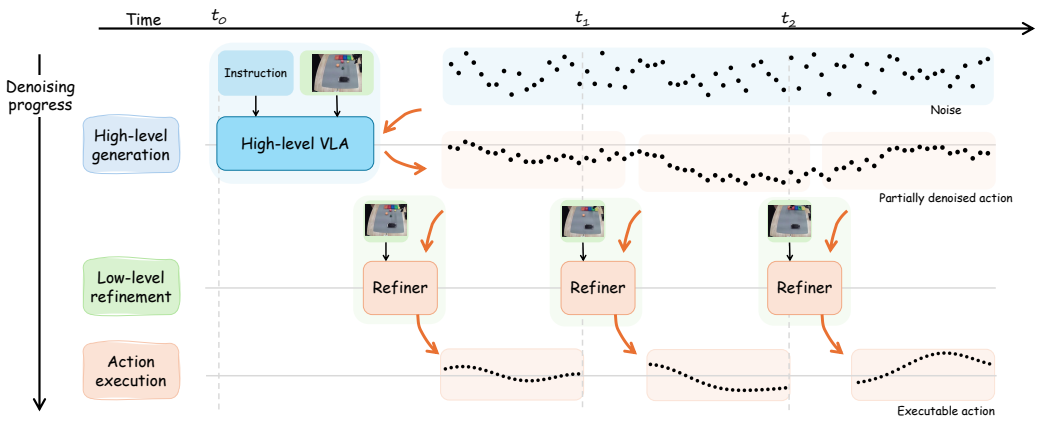


Figure 2: Overall framework of StairVLA. The high-level VLA performs early denoising over a long-horizon trajectory and caches the partially denoised trajectory for reuse across multiple steps. For each local chunk, the refiner uses the corresponding segment and the latest observation to complete refinement. The top, middle, and bottom sequences denote the initial noise, partially denoised trajectory, and executable action chunk, respectively.

2 Related Work

Action Chunking and Unified VLA Models. Unified VLA models typically map language instructions, visual observations, and robot states into action sequences within a single policy. Recent VLA models further adopt generative action heads based on diffusion or flow matching to produce continuous actions (Black et al., 2024; Bjorck et al., 2025; Chi et al., 2025). These methods commonly use action chunking, predicting a sequence of future actions rather than a single-step action to improve temporal consistency and reduce policy invocation frequency (Zhao et al., 2023; Chi et al., 2025). However, action chunking introduces an inherent horizon trade-off: short chunks require frequent invocation of computationally expensive VLMs, while long chunks weaken closed-loop feedback and real-time correction during execution (Zhang et al., 2026). Therefore, processing different information and generating actions at a single temporal scale may be unnecessarily restrictive.

Hierarchical VLA Models. Recent works increasingly adopt hierarchical structures to bridge high-level task understanding and low-level robot control. Earlier methods use explicit intermediate goals to guide low-level policies, such as subgoals, waypoints, keypoints, or language subinstructions (Hu et al., 2023; Liu et al., 2024; Li et al., 2025; Shi et al., 2025). More recent works further integrate this hierarchy into the model architecture and explore different high-to-low-level interfaces, including VLM tokens, learned features, discrete macro-actions, and coarse action representations (Cui et al., 2025; Guo et al., 2026; Wei et al., 2026; Wu et al., 2026). Other dual-system approaches focus more explicitly on temporal organization, using event-driven high-level updates or allowing different levels to operate at different frequencies (Chen et al., 2026; Sun et al., 2026). Despite their different forms, these methods highlight a common design question in hierarchical VLA: how should high-level task intent be transferred to low-level action generation? Rather than using manually designed or learned interfaces, we directly leverage intermediate states formed during action generation as a natural high-to-low-level interface.

Generative Action Models and Intermediate Denoising States. Diffusion- and flow-based generation progressively transforms noise into executable actions. In both generative modeling and robot policies, prior works assign different denoising experts to different sampling stages to better accommodate stage-specific generation requirements (Balaji et al., 2022; Zhang et al., 2024; Reuss et al., 2025; Du et al., 2026; Niu et al., 2026). Another line of work reuses partially denoised trajectories or constructs better warm starts to reduce redundant sampling and control latency (Høeg et al., 2025; Chen et al., 2025; Li et al., 2026). Our method further uses partially denoised actions to transfer task intent across stages, enabling high-level coarse action generation and local refinement to use different conditioning and operate at different frequencies.

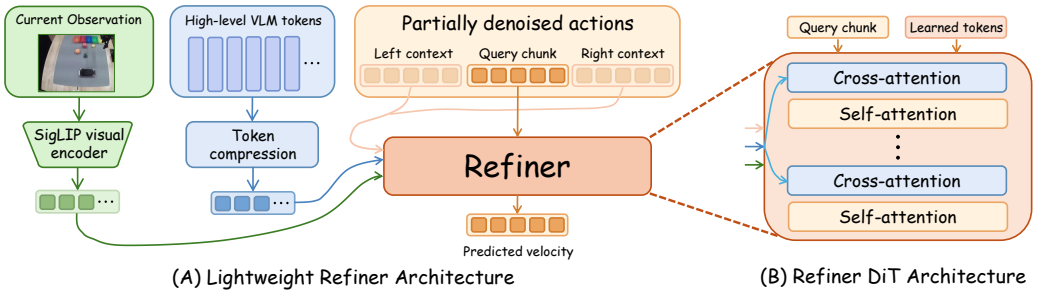


Figure 3: Overview of the low-level refiner. **(A)** The refiner uses the current observation, compressed high-level VLM tokens, and partially denoised action context to refine the query chunk. **(B)** The refiner adopts a GR00T-style DiT with interleaved cross-attention and self-attention for conditional fusion and action modeling.

3 METHOD

3.1 OVERVIEW

As illustrated in Fig. 2, we decompose VLA action generation into two stages that operate at different frequencies. The high-level VLA is invoked at a lower frequency to perform early denoising and produce a long-horizon partially denoised action trajectory, which is reused across multiple local action chunks. The low-level refiner runs at a higher frequency, taking the corresponding local action segment and the latest visual observation to produce an executable action chunk. This design reduces repeated VLM computation while preserving closed-loop correction with updated observations.

Our framework is designed to be applicable to flow-matching-based generative VLAs and is trained in two stages. The high-level VLA retains the original VLA architecture with an extended action horizon. The lightweight refiner is the main additional module and follows the design of the VLA action head. It starts from partially denoised actions produced by the trained high-level VLA and performs local refinement using current observations. The high-level generation, refiner architecture and training, and multi-rate inference procedures are detailed in the following subsections.

3.2 HIGH-LEVEL INTENT TRAJECTORY GENERATION

The high-level VLA retains the original generative architecture and training objective, while extending the action horizon to H. Taking a flow-matching VLA as an example, training constructs a continuous path between Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the ground-truth $\mathbf{a}^{\mathrm{gt}}$:

$$\mathbf{a}^{\gamma} = (1-\gamma)\boldsymbol{\epsilon} + \gamma \mathbf{a}^{\mathrm{gt}}, \qquad \gamma \in [0,1]. \tag{1}$$

The corresponding target velocity is

$$\mathbf{v}^{*} = \mathbf{a}^{\mathrm{gt}} - \boldsymbol{\epsilon}. \tag{2}$$

Given the intermediate action state $\mathbf{a}^{\gamma}$, denoising progress γ, and VLM conditioning feature $\mathbf{z}_t$, the action head v_θ is trained to predict this velocity:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}\left[\left\| v_\theta\left(\mathbf{a}^{\gamma}, \gamma, \mathbf{z}_t\right) - \mathbf{v}^{*} \right\|_2^2\right]. \tag{3}$$

At inference time, action generation starts from random noise, $\mathbf{a}^{\gamma_0} = \boldsymbol{\epsilon}$, and iteratively updates the action state by repeatedly predicting the velocity:

$$\mathbf{a}^{\gamma_{n+1}} = \mathbf{a}^{\gamma_n} + (\gamma_{n+1} - \gamma_n)\, v_\theta\left(\mathbf{a}^{\gamma_n}, \gamma_n, \mathbf{z}_t\right). \tag{4}$$

A standard VLA continues this process until $\gamma_N = 1$ to obtain the final executable action trajectory. In contrast, we terminate the high-level generation at $\gamma_N = \alpha < 1$ and use the resulting partially denoised trajectory $\mathbf{a}^{\alpha}_{t:t+H-1}$ as the high-level output. This long-horizon trajectory is then cached and serves as the starting point for subsequent low-level refinement.

3.3 Observation-Conditioned Low-Level Refinement

As illustrated in Figure 3, we introduce a lightweight refiner to further refine the partially denoised local action segments produced by the high-level VLA. The refiner uses a SigLIP-B/16 visual encoder to process the latest observation, which is jointly trained. Meanwhile, the hidden states from the last layer of the high-level VLM are compressed into a fixed number of tokens through adaptive average pooling, which are cached and reused across multiple refinement cycles.

As illustrated on the right side of Figure 3, the refiner follows the overall architecture of the GR00T action head (Bjorck et al., 2025) and takes two types of inputs: action-side tokens and condition-side tokens. The action side includes the robot state token, learnable future tokens, and the current query action chunk, while the condition side consists of visual tokens, compressed VLM tokens, and temporal action context extracted from the cached high-level trajectory. Self-attention operates only on the action-side tokens to model dependencies within the action representation, whereas cross-attention uses the action-side tokens as queries to incorporate information from the condition-side tokens. These two attention layers are interleaved throughout the DiT, alternating between action modeling and conditional fusion.

The refiner training does not directly follow the original flow-matching path of the high-level VLA. Due to prediction errors, its partially denoised actions may not lie exactly on the ideal generation path defined by the denoising origin and the ground-truth. Therefore, as illustrated in Figure 4, for each local query chunk, we reconstruct a refinement path toward the ground-truth.

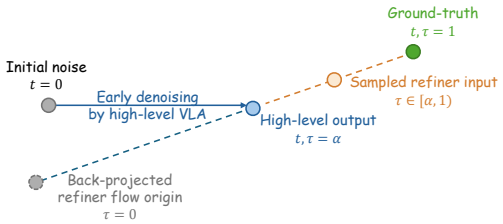


Figure 4: Refiner training path construction. After early denoising by the high-level VLA, the resulting partially denoised action is used to construct the refinement path toward the ground-truth action, from which both refiner inputs and targets are generated.

For clarity, we omit the local chunk indices in this subsection. We treat the high-level output $\mathbf{a}^{\alpha}$ as the state of a reconstructed flow at time α and back-project its virtual origin as

$$\boldsymbol{\epsilon}_{\text{ref}} = \frac{\mathbf{a}^{\alpha} - \alpha \mathbf{a}^{\text{gt}}}{1 - \alpha}. \tag{5}$$

The corresponding target velocity is

$$\mathbf{v}_{\text{ref}} = \mathbf{a}^{\text{gt}} - \boldsymbol{\epsilon}_{\text{ref}} = \frac{\mathbf{a}^{\text{gt}} - \mathbf{a}^{\alpha}}{1 - \alpha}. \tag{6}$$

During training, we sample $\tau \sim \mathcal{U}(\alpha, 1)$ and construct the intermediate refiner input along the reconstructed path:

$$\mathbf{a}^{\tau}_{\text{ref}} = (1 - \tau)\boldsymbol{\epsilon}_{\text{ref}} + \tau \mathbf{a}^{\text{gt}}. \tag{7}$$

The refiner v_{ϕ} is optimized to predict the reconstructed velocity:

$$\mathcal{L}_{\text{ref}} = \mathbb{E}\left[\left\| v_{\phi}\left(\mathbf{a}^{\tau}_{\text{ref}}, \tau, \mathbf{c}_{\text{ref}}\right) - \mathbf{v}_{\text{ref}} \right\|_2^2 \right], \tag{8}$$

where $\mathbf{c}_{\text{ref}}$ denotes the condition-side inputs described above.

3.4 Multi-Rate Inference and Horizon Reuse

During inference, each high-level VLA invocation generates and caches a partially denoised trajectory of horizon H, which is reused across the following M low-level refinement cycles. At each cycle, the low-level refiner extracts the corresponding partially denoised segment of length h from the cached trajectory and further refines it.

Let T_{high} and T_{ref} denote the inference time of one high-level invocation and one low-level refinement, respectively. The average inference cost per local action chunk is

$$\overline{T}(M) = T_{\text{ref}} + \frac{T_{\text{high}}}{M}. \tag{9}$$

As M increases, the high-level VLA cost is amortized over more local action chunks. However, a larger M also reuses the same high-level trajectory for a longer horizon, which may reduce its consistency with the latest environment state. Therefore, M controls the trade-off between computational efficiency and the freshness of high-level information. We evaluate this trade-off under different reuse horizons in Section 4.

4 EXPERIMENTS

We evaluate StairVLA in terms of task performance, inference efficiency and cross-backbone generality, followed by analyses of stage-dependent denoising behavior, key design choices and real-world deployment.

4.1 MODEL AND TRAINING SETUP

Model Configuration. We implement our method based on StarVLA (StarVLA Community, 2026). By default, the high-level policy uses Qwen3-VL-4B-Instruct as the vision-language backbone with a GR00T-style flow-matching action head, and predicts an action horizon of $H = 32$. The low-level refiner outputs action chunks of length $h = 5$ and uses a trainable SigLIP-B/16 visual encoder. In addition, the refiner uses $k = 5$ high-level actions before and after the query chunk as temporal context.

Training Setup. We adopt a two-stage training strategy. On LIBERO, we first train the high-level VLA for 30k steps and then freeze the high-level policy to train the low-level refiner for another 30k steps. By default, the refiner is trained using high-level trajectories at denoising progress $\alpha = 0.8$. We use AdamW with a global batch size of 128, an initial learning rate of 5×10^{-5}, cosine learning-rate decay, 1k warmup steps, and a weight decay of 0.01. All models are trained with bf16 mixed precision on 8 NVIDIA RTX A6000 GPUs.

Inference and Latency Measurement. Unless otherwise specified, we use high-level trajectories at $\alpha = 0.97$ and reuse each trajectory for $M = 4$ refinement cycles, with $h = 5$ actions per cycle. Latency is measured per executable action chunk and includes the amortized high-level inference cost and low-level refinement cost, excluding environment execution. All measurements are conducted on a single NVIDIA A100 GPU. Additional implementation, training, and preprocessing details are provided in Appendix A.

4.2 MAIN RESULTS

LIBERO. We first evaluate task performance and inference efficiency on LIBERO (Liu et al., 2023). We jointly train a single model on the Spatial, Object, Goal, and Long suites, and evaluate each task with 50 rollouts. As shown in Table 1, our method achieves strong task performance across both VLA backbones, while also maintaining a low average inference latency. Compared with existing hierarchical and coarse-to-fine approaches, our method provides a competitive trade-off between task success and inference efficiency. We note that some prior methods use additional subtask-level supervision, while latency definitions, measurement scopes, and hardware settings also vary between works. Therefore, the reported efficiency results are intended to reflect overall trends rather than serve as strictly comparable absolute runtimes.

LIBERO-Plus. We further evaluate our method on LIBERO-Plus (Fei et al., 2026), which introduces seven types of perturbations to the original LIBERO tasks. Following the standard protocol, we consider both zero-shot transfer and supervised fine-tuning. In the zero-shot setting, the model trained on LIBERO is directly evaluated in the perturbed environments, while in the supervised setting, it is further fine-tuned on the LIBERO-Plus training set using the same hyperparameter configuration as in LIBERO. As shown in Table 2, our method achieves competitive overall performance in the zero-shot setting. However, since the low-level refiner mainly relies on visual observations for local correction, performance drops more noticeably under image-related perturbations such as camera changes. After supervised fine-tuning, performance recovers substantially across these perturbation categories, reaching an average success rate of 83.7% and competitive results on multiple

Table 1: Comparison on the LIBERO benchmark. We report success rates (%) on the four LIBERO suites, the average success rate (%), and reported inference latency (ms/chunk) when available. Chunk sizes may vary across methods. Best and second-best results are shown in **bold** and underlined, respectively. * Uses additional subtask-level supervision beyond standard action demonstrations and is excluded from best/second-best ranking. ‡ StreamVLA reports average wall-clock latency under its gated reasoning schedule. § CF-VLA reports action-sampling latency excluding visual-language prefix encoding and KV-cache construction.

Method	Spatial	Object	Goal	Long	Avg.	Latency
Representative VLA Policies						
OpenVLA (Kim et al., 2024)	84.7	88.4	79.2	53.7	76.5	–
π_0 (Black et al., 2024)	96.8	98.8	95.8	85.2	94.1	–
GR00T (Bjorck et al., 2025)	94.4	97.6	93.0	90.6	93.9	–
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	98.8	98.2	98.0	92.4	96.9	–
Hierarchical / Coarse-to-Fine VLA Policies						
HiMoE-VLA (Du et al., 2025)	98.2	99.4	**98.6**	94.8	97.8	–
StreamVLA (Chen et al., 2026)	99.2*	99.4*	98.6*	96.6*	98.5*	128.0‡
CF-VLA (Du et al., 2026)	98.0	99.2	96.6	92.0	96.5	7.81§
Coarse-to-Control (Wu et al., 2026)	98.8	**100.0**	97.8	95.0	97.9	–
Libra-VLA (Wei et al., 2026)	98.6	99.4	98.0	92.8	97.2	122.0
StarVLA Baselines and Ours						
StarVLA-GR00T (StarVLA Community, 2026)	97.8	98.8	97.4	92.0	96.5	115.0
Ours (GR00T-base)	98.0	99.6	97.8	95.8	97.8 (+1.3)	44.2
StarVLA-π (StarVLA Community, 2026)	99.2	99.0	97.2	95.8	97.8	183.3
Ours (π-base)	**99.6**	99.0	96.8	**97.4**	**98.3** (+0.5)	44.5

Table 2: Results on LIBERO-Plus under zero-shot transfer and supervised fine-tuning.

Method	Camera	Robot	Language	Light	Background	Noise	Layout	Avg.
Zero-Shot Transfer								
OpenVLA (Kim et al., 2024)	0.8	3.5	23.0	8.1	34.8	15.2	28.5	15.6
π_0-FAST (Pertsch et al., 2025)	65.1	21.6	61.0	73.2	73.2	74.4	68.8	61.6
OpenVLA-OFT (Kim et al., 2025)	56.4	31.9	79.5	88.7	93.3	75.8	74.2	69.6
GR00T (Bjorck et al., 2025)	34.6	50.8	85.1	86.5	86.5	63.0	73.6	66.8
Libra-VLA (Wei et al., 2026)	**68.9**	48.8	**92.7**	**97.9**	**93.4**	**86.3**	**77.5**	**79.5**
Ours	43.7	**54.5**	82.5	93.4	90.5	67.8	74.0	70.4
Supervised Fine-Tuning								
π_0 (Black et al., 2024)	79.6	21.1	72.5	84.7	86.2	68.3	69.4	67.4
$\pi_{0.5}$ (Physical Intelligence et al., 2025)	70.3	41.7	81.1	**97.3**	94.6	71.8	**84.9**	75.7
OpenVLA-OFT+ (Kim et al., 2025)	92.8	30.3	**85.8**	94.9	93.9	89.3	77.6	79.6
Libra-VLA (Wei et al., 2026)	94.5	41.8	83.2	95.3	94.3	93.7	75.3	82.3
Ours	**95.5**	**48.4**	83.7	96.8	**94.9**	**95.4**	77.4	**83.7**

categories. These results suggest that, although it is sensitive to zero-shot visual shifts, it can effectively adapt to new visual conditions through target-distribution fine-tuning.

Across VLA Backbones. We apply our method to both GR00T-based and π-based StarVLA backbones. As shown in Table 1, hierarchical refinement improves the average success rate on both, demonstrating that the method is not limited to a specific VLA backbone. Complete results across different reuse horizons for both backbones are provided in Appendix B.2.

4.3 Analysis of Stage-Aware Action Generation

Stage-Dependent Denoising Behavior. We analyze a trained StarVLA-π model using 256 randomly sampled LIBERO examples and measure velocity prediction error at different denoising progress levels. As shown in Fig. 5, even without any masking, the error rises sharply near the end of denoising, suggesting that a unified generation mechanism is not equally effective across all stages. Masking the image further causes a larger error increase at later stages, supporting the stronger dependence on visual observations during late-stage refinement. This effect decreases again near completion, possibly because the remaining residuals can be largely corrected through simple local

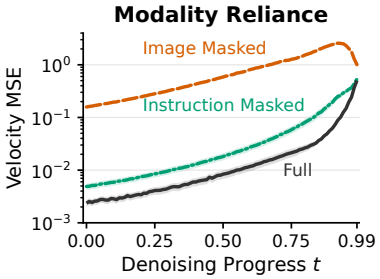


Figure 5: Conditioning sensitivity across denoising stages on π_0.

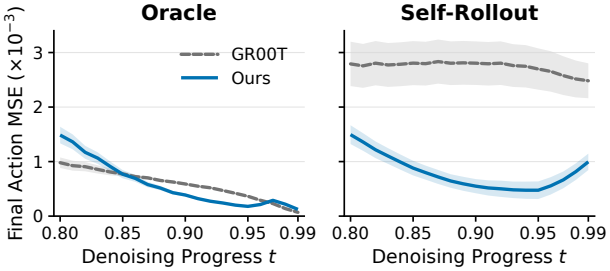


Figure 6: Final action error of late-stage refinement under oracle and self-rollout settings.

smoothing. These results motivate our stage-wise generation design and observation-conditioned refiner. Additional denoising-stage conditioning analyses are provided in Appendix B.1.

Effectiveness of Late-Stage Refinement. We randomly sample 1024 LIBERO examples and evaluate two settings: inputs constructed along the training denoising trajectory, and intermediate actions obtained through iterative inference, where prediction errors accumulate. As shown in Fig. 6, we measure the error between the final generated actions and the ground truth. Our refiner maintains low and stable error in both settings, while GR00T degrades substantially once accumulated errors are introduced.

Effect of Additional Instruction Conditioning. To investigate whether the low-level refiner requires additional instruction information, we augment the baseline refiner with full, unpooled instruction tokens and compare their performance across different inference-time denoising levels. As shown in Fig. 7, the additional instruction tokens help recover task-relevant information at higher noise levels, but do not further improve the refiner's *best achievable performance*. In contrast, using lighter task conditioning allows the refiner to focus more on observation-conditioned action alignment, leading to a higher performance ceiling.

4.4 Ablation Studies and Analysis

Component Ablation. As shown in Table 3, the top-only setting, which only increases the high-level prediction horizon, leads to a clear performance drop. Adding the observation-conditioned refiner substantially recovers performance, while temporal action context provides a further improvement, validating both design choices.

Table 3: Ablations on LIBERO.

Setting	Refiner	Action Context	Spatial	Object	Goal	Long	Avg.	Δ
StarVLA-GR00T (StarVLA Community, 2026)	–	–	97.8	98.8	97.4	92.0	96.5	–
Ours w/o refiner (Top-only)	✗	–	96.8	89.4	96.8	86.4	92.4	-4.1
Ours w/o Action Context	✓	✗	**99.2**	98.6	95.6	95.4	97.2	+0.7
Ours	✓	✓	98.0	**99.6**	**97.8**	**95.8**	**97.8**	**+1.3**

High-Level Trajectory Reuse Horizon. We study the effect of the reuse length M on performance and efficiency, where M denotes the number of low-level refinement cycles executed between two high-level VLA updates. As shown in Fig. 8, our method maintains strong task performance under longer high-level reuse horizons while substantially reducing inference latency, leading to a more favorable efficiency–performance trade-off. Additional experiments are provided in Appendix B.3.

Denoising Progress of the Transferred Trajectory. We further study the denoising progress of the transferred high-level trajectory during both training and inference. As shown in Fig. 9, the refiner is relatively robust to the training-time transfer point α over the tested range. At inference time, the baseline sweep shown in Fig. 7 indicates that performance improves as the transferred trajectory approaches complete denoising and peaks near $\alpha = 0.97$. Notably, we directly use this

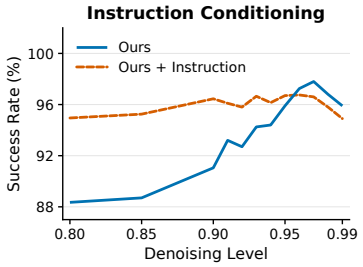


Figure 7: Effect of additional instruction conditioning.

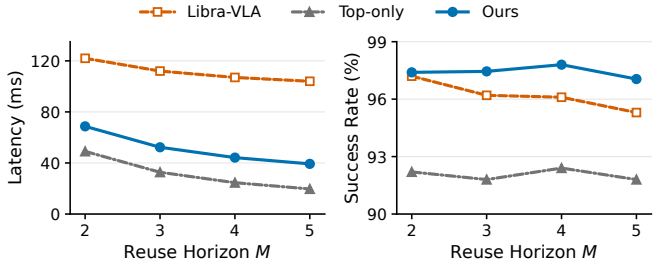


Figure 8: Effect of high-level reuse horizon M on inference latency and task success.

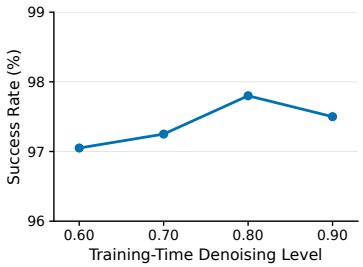


Figure 9: Effect of the training-time denoising level.

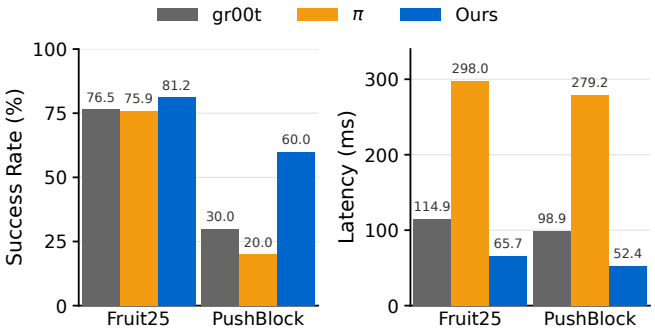


Figure 10: Real-world results on Fruit25 and PushBlock.

inference-time setting across other benchmarks and backbone variants without retuning, while still maintaining strong performance. These results suggest that the transfer point around 0.97 provides a useful balance between task information already encoded in the partially denoised trajectory and the residual space available for low-level refinement.

4.5 Real-World Experiments

We further evaluate our method on an AgileX PiperX robot in two real-world settings. *Fruit25* includes 24 language-conditioned fruit pick-and-place tasks and one long-horizon multi-object sorting task, while *PushBlock* consists of a single continuous pushing task. We compare against StarVLA-GR00T and StarVLA-π (StarVLA Community, 2026), all using Qwen3-VL-2B-Instruct as the vision-language backbone and evaluated on the same NVIDIA RTX A4000 GPU with the same training data and evaluation protocol. As shown in Fig. 10, the real-world results exhibit performance and efficiency gains consistent with those observed in simulation. Additional details on the real-world platform, datasets, training and evaluation protocols, and per-task results are provided in Appendix C.

5 Conclusion

We propose StairVLA, a stage-aware hierarchical action generation framework that combines low-frequency high-level generation with high-frequency observation-conditioned refinement to reduce inference cost while maintaining strong task performance. More importantly, our results suggest that VLA action generation is not a homogeneous denoising process: different stages exhibit different information dependencies, while partially denoised actions themselves already encode task-relevant information for subsequent refinement. Our method exploits this stage difference by explicitly separating early-stage generation from later-stage refinement. Further understanding the functional roles of different denoising stages, better utilizing the information encoded in intermediate actions, and learning such hierarchical structures adaptively remain important directions for future research.

AI USE STATEMENT

In this work, we used generative AI tools to assist with method implementation and translation. We did not use generative AI tools for synthetic data generation, development of theoretical models or conceptual frameworks, formulation or proof of mathematical claims, proposing or refining research hypotheses, research methodology or experimental design, dataset cleaning or reformatting, qualitative or thematic data analysis, or interpretation of experimental results; the remaining tasks requiring disclosure were not applicable to this work. Additionally, we used generative AI tools to search for research information, identify relevant literature, and improve the language and readability of the manuscript.

We reviewed all AI-assisted work. AI-assisted code generation or modification was conducted under explicit guidance from the authors and was subsequently inspected and tested by the authors. Relevant literature and research information identified with AI assistance were independently verified by consulting the original sources. We take full responsibility for the final content of this work, including any text, code, or other research artifacts produced or modified with the assistance of generative AI.

REFERENCES

Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, et al. ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv preprint arXiv:2211.01324*, 2022.

Sivakumar Balasubramanian, Alejandro Melendez-Calderon, Agnes Roby-Brami, and Etienne Burdet. On the analysis of movement smoothness. *Journal of neuroengineering and rehabilitation*, 12(1):112, 2015.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Tongqing Chen, Hang Wu, Jiasen Wang, Xiaotao Li, and Lu Fang. Streamvla: Breaking the reason-act cycle via completion-state gating. *arXiv preprint arXiv:2602.01100*, 2026.

Zhuoqun Chen, Xiu Yuan, Tongzhou Mu, and Hao Su. Responsive noise-relaying diffusion policy: Responsive and efficient visuomotor control. *arXiv preprint arXiv:2502.12724*, 2025.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Can Cui, Pengxiang Ding, Wenxuan Song, Shuanghao Bai, Xinyang Tong, Zirui Ge, Runze Suo, Wanqi Zhou, Yang Liu, Bofang Jia, et al. Openhelix: A short survey, empirical analysis, and open-source dual-system vla model for robotic manipulation. *arXiv preprint arXiv:2505.03912*, 2025.

Fan Du, Feng Yan, Jianxiong Wu, Xinrun Xu, Weiye Zhang, Weinong Wang, Yu Guo, Bin Qian, Zhihai He, Fei Wang, et al. Cf-vla: Efficient coarse-to-fine action generation for vision-language-action policies. *arXiv preprint arXiv:2604.24622*, 2026.

Zhiying Du, Bei Liu, Yaobo Liang, Yichao Shen, Haidong Cao, Xiangyu Zheng, Zhiyuan Feng, Zuxuan Wu, Jiaolong Yang, and Yu-Gang Jiang. Himoe-vla: Hierarchical mixture-of-experts for generalist vision-language-action policies. *arXiv preprint arXiv:2512.05693*, 2025.

Fabian Falck, Teodora Pandeva, Kiarash Zahirnia, Rachel Lawrence, Richard Turner, Edward Meeds, Javier Zazo, and Sushrut Karmalkar. A fourier space perspective on diffusion models. *arXiv preprint arXiv:2505.11278*, 2025.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: A progressive robustness benchmark for visual-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 38574–38583, 2026.

Weiyu Guo, He Zhang, Pengteng Li, Tiefu Cai, Ziyang Chen, Yandong Guo, Xiao He, Yongkui Yang, Ying Sun, and Hui Xiong. A brain-inspired embodied intelligence for fluid and fast reflexive robotics control. *arXiv preprint arXiv:2601.14628*, 2026.

Sigmund H Høeg, Yilun Du, and Olav Egeland. Fast policy synthesis with variable noise diffusion models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4821–4828. IEEE, 2025.

Neville Hogan and Dagmar Sternad. Sensitivity of smoothness measures to movement duration, amplitude, and arrests. *Journal of motor behavior*, 41(6):529–534, 2009.

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, 2023.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag 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, 2024. URL https://arxiv.org/abs/2406.09246.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

Jinhao Li, Yuxuan Cong, Yingqiao Wang, Hao Xia, Shan Huang, Yijia Zhang, Ningyi Xu, and Guohao Dai. STEP: Warm-started visuomotor policies with spatiotemporal consistency prediction. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=nWtjaAmug8.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, et al. Hamster: Hierarchical action models for open-world robot manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 24040–24068, 2025.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. Moka: Open-world robotic manipulation through mark-based visual prompting. *arXiv preprint arXiv:2403.03174*, 2024.

Dantong Niu, Zhuoyang Liu, Zekai Wang, Boning Shao, Zhao-Heng Yin, Anirudh Pai, Yuvan Sharma, Stefano Saravalle, Ruijie Zheng, Jing Wang, et al. T-rex: Tactile-reactive dexterous manipulation. *arXiv preprint arXiv:2606.17055*, 2026.

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

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Moritz Reuss, Jyothish Pari, Pulkit Agrawal, and Rudolf Lioutikov. Efficient diffusion transformer policies with mixture of expert denoisers for multitask learning. In *International Conference on Learning Representations*, volume 2025, pp. 17247–17275, 2025.

Lucy Xiaoyang Shi, Brian Ichter, Michael Robert Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 54919–54933. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/shi25d.html.

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing, 2026. URL https://arxiv.org/abs/2604.05014.

Lin Sun, Zhiwei Guan, Conglin Wang, Zihong Chen, Jianhai Yu, Zongsheng Li, Boyong He, Tao Sun, Jiale Cao, and Lige Liu. Unifs: Unified fast-to-slow hierarchical architecture for vision-language-action models. *arXiv preprint arXiv:2606.22794*, 2026.

Yifei Wei, Linqing Zhong, Yi Liu, Yuxiang Lu, Xindong He, Maoqing Yao, and Guanghui Ren. Libra-vla: Achieving learning equilibrium via asynchronous coarse-to-fine dual-system. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39799–39815, 2026.

Jinhao Wu, Shiduo Zhang, Yicheng Liu, Xiaopeng Yu, Sixian Li, Siyin Wang, Hang Zhao, Jing Huo, Yang Gao, Jingjing Gong, et al. Coarse-to-control: Action-token planning for vision-language-action models. *arXiv preprint arXiv:2606.07107*, 2026.

Huijie Zhang, Yifu Lu, Ismail Alkhouri, Saiprasad Ravishankar, Dogyoon Song, and Qing Qu. Improving training efficiency of diffusion models via multi-stage framework and tailored multi-decoder architecture. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7372–7381. IEEE, 2024.

Thomas T. Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and data augmentation yield exponential improvements in behavior cloning for continuous spaces. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 126689–126742, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/cd96cb9a239c37b39dbf34f3f5a4c56f-Paper-Conference.pdf.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016.

APPENDIX

This appendix is organized as follows:

- **Appendix A** provides additional implementation details.
- **Appendix B** presents additional simulation results and analyses.
- **Appendix C** provides detailed real-world experimental settings, task definitions, evaluation protocols, and additional analyses.

A IMPLEMENTATION DETAILS

Unless otherwise specified, all experiments use the model architecture, training protocol, and evaluation settings described in this section.

A.1 MODEL ARCHITECTURE

High-level policy. The high-level policy uses Qwen3-VL-4B-Instruct as the vision-language backbone, followed by a GR00T-style flow-matching action head. The action prediction horizon is set to $H = 32$.

Low-level refiner. The low-level refiner adopts a DiT-style flow-matching architecture with 16 Transformer blocks, hidden dimension 768, 12 attention heads, and FFN dimension 3072. Cross-attention and self-attention blocks are interleaved. The refiner predicts an action chunk of length $h = 5$.

Visual and high-level conditioning. The refiner uses a trainable SigLIP-B/16 visual encoder initialized from `google/siglip-base-patch16-224`, with RGB observations from both the primary and wrist cameras. The final-layer hidden states of the high-level VLM are first projected to dimension 768 and then compressed along the token dimension using adaptive average pooling, producing eight high-level latent tokens. These features are cached together with the high-level action trajectory and reused across multiple low-level refinement cycles.

Local action context. For each query chunk, the refiner additionally uses five coarse actions before and five coarse actions after the query segment from the same high-level trajectory as local temporal context. The query chunk itself is excluded from the context. A learned padding token is used when the required context extends beyond the trajectory boundary.

Multi-rate inference. The refiner is invoked once every $h = 5$ environment steps and returns five actions that are executed sequentially. The high-level policy is refreshed every M low-level refinement cycles, corresponding to $5M$ environment steps. Refined actions are not written back into the cached high-level trajectory; each local chunk is refined from the original high-level coarse trajectory.

A.2 Training Details

High-level policy training. The high-level policy is trained on the combined LIBERO dataset for 30k steps with an action horizon of $H = 32$. After training, both the high-level VLM and action head are frozen during refiner training.

Refiner training. During refiner training, the frozen high-level policy remains active to generate intermediate action trajectories. The high-level policy is run to a fixed denoising progress $\alpha = 0.8$, producing a partially denoised trajectory C.

For each training sample, we then sample

$$\tau \sim \mathcal{U}(0.8, 1.0),$$

and construct the refiner input by interpolating between the high-level coarse trajectory and the ground-truth action:

$$x_\tau = C + \frac{\tau - \alpha}{1 - \alpha} \left(a_{\mathrm{gt}} - C\right).$$

The refiner predicts the remaining flow velocity toward the ground-truth action and is trained using a validity-masked mean-squared error. The refiner is trained for 30k steps.

Optimization. The refiner is trained with AdamW using a learning rate of 5×10^{-5} for the refiner head and a weight decay of 0.01. We use a cosine learning-rate schedule with 1k warmup steps. Training uses bf16 mixed precision, DeepSpeed ZeRO-2, and gradient clipping, and is conducted on eight NVIDIA RTX A6000 48GB GPUs. The per-GPU batch size is 16 with no gradient accumulation, resulting in a global batch size of 128.

Dataset and preprocessing. We train on the combined LIBERO dataset and evaluate on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. The action dimension is 7 and the high-level action horizon is 32. Continuous action dimensions are min–max normalized to $[-1, 1]$. Action sequences extending beyond episode boundaries are zero-padded, and invalid padded steps are excluded from the training loss using a validity mask. Both primary and wrist-camera observations are used during training and evaluation, and video frames are decoded using `torchvision_av`.

B Additional Experimental Results and Analysis

B.1 Denoising-Stage Conditioning Analysis

We further analyze how generative VLA action heads use different conditioning information throughout the denoising process. Specifically, we examine how action queries allocate cross-attention to different conditioning sources across denoising stages in both GR00T and π_0. This analysis is intended as a diagnostic of stage-dependent conditioning behavior rather than a strict causal attribution of individual token groups to the final action prediction.

B.1.1 Experimental Setup

Here, we use two trained StarVLA checkpoints based on GR00T and π_0, and randomly sample 16 observation–instruction pairs from the LIBERO LeRobot dataset. All analyses are performed at inference time without additional training or parameter updates.

To examine how attention evolves throughout denoising, we densify the inference trajectory into 100 analysis stages and record the cross-attention probability from action queries to conditioning-token keys at each stage. This 100-stage trajectory is used only for analysis and visualization and does not correspond to the native inference setting of the models, whose default inference trajectory uses four denoising steps. For each model, we analyze both the first and last action-head layers, resulting in four cases: GR00T first layer, GR00T last layer, π_0 first layer, and π_0 last layer.

At each denoising stage, we first average the attention probabilities over attention heads and action-query tokens to obtain a key-side attention distribution. We then sum the attention mass over tokens belonging to the same group and normalize it by the total attention mass over valid tokens. The reported curves represent the mean attention mass over the 16 sampled observation–instruction pairs.

We divide the conditioning tokens into five groups: *task instruction*, *wrapper prompt*, *grounding prompt*, *image tokens*, and *special tokens*. The task instruction corresponds to the actual LIBERO command, such as "pick up the cream cheese and place it in the basket." The wrapper prompt refers to template text preceding the task instruction, such as "Your task is." The grounding prompt refers to localization-oriented template text following the task instruction, such as a request to identify key objects using bounding-box coordinates. Importantly, the grounding prompt specifies only the output format and does not contain ground-truth bounding boxes. Image tokens correspond to visual inputs, while special tokens include chat-template tokens, image-boundary tokens, role-boundary tokens, padding tokens, and other special tokens.

For compact visualization, wrapper-prompt, grounding-prompt, and special tokens can be grouped as "other prompt tokens." In the analysis below, we keep the five groups separate to better examine the source of attention shifts across models.

B.1.2 Cross-Attention Allocation across Denoising Stages

The overall results are shown in Fig. 11. Both GR00T and π_0 exhibit changes in cross-attention allocation throughout denoising, indicating that the action head does not read conditioning information in a completely static manner during generation. Compared with the last layer, these changes are more pronounced in the first action-head layer.

The first-layer result of π_0 provides the clearest example. As denoising approaches later stages, the attention mass allocated to image tokens increases substantially, while the proportion assigned to prompt-related tokens decreases. This suggests that late-stage action queries in π_0 place greater explicit emphasis on visual conditioning, consistent with the interpretation that later refinement increasingly focuses on alignment with the current observation.

GR00T exhibits a different surface trend. In its first action-head layer, image-token attention decreases toward later stages, while attention to grounding-prompt and other prompt-related positions increases. We attribute this difference primarily to the different conditioning architectures of the two models. π_0 uses layer-wise VLM conditioning, such that its first action-head layer receives comparatively shallow VLM-contextualized representations. In contrast, GR00T conditions its action head on the final-layer hidden states of Qwen2.5-VL. Therefore, even at the first action-head layer,

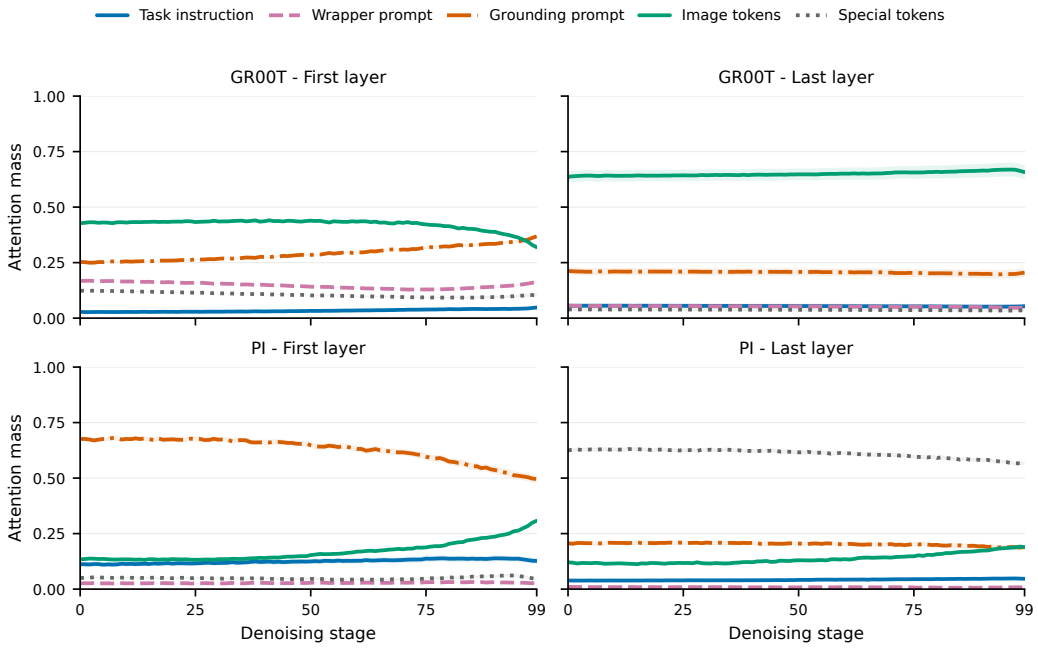


Figure 11: Denoising-stage attention analysis for GR00T and π_0. We record action-query cross-attention to different conditioning-token groups over 100 dense denoising stages. Both models exhibit stage-dependent attention patterns, with the changes being more pronounced in the first action-head layer.

token positions corresponding to grounding or prompt text have already undergone multiple layers of visual-language contextualization.

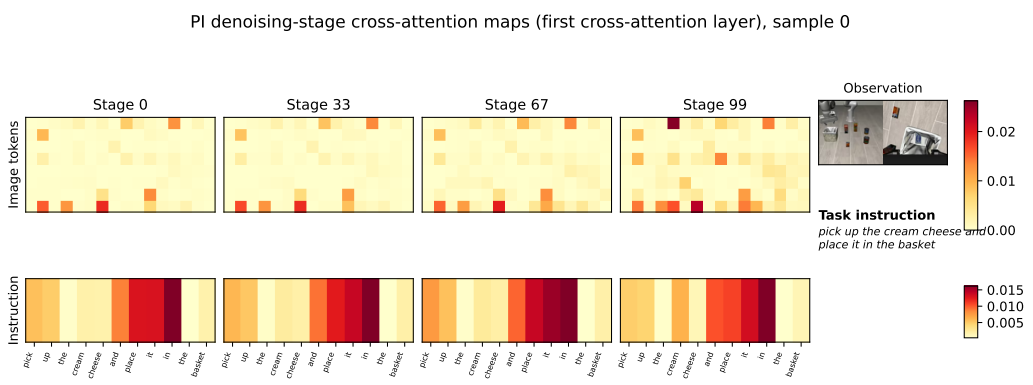
Consequently, the increase in attention to grounding-related token positions in GR00T should not be interpreted simply as an increased reliance on template text. Importantly, these grounding representations themselves encode image-derived spatial information, such as object bounding boxes. Their increased attention therefore more likely reflects a reallocation of cross-attention toward more explicit visual-spatial grounding representations during later denoising stages. Overall, although GR00T and π_0 exhibit different attention patterns, both suggest a shift toward more visually grounded information in later denoising stages.

The last-layer trends are weaker and should be interpreted more cautiously. One possible explanation is that the first action-head layer is the earliest point at which action queries explicitly read conditioning tokens, making it more sensitive to stage-dependent changes in conditioning usage. After multiple action-head transformations, conditioning information has progressively been integrated into the evolving action representation, making explicit attention over the original token groups less sensitive to such changes. The last layer of π_0 still exhibits a weaker but visible increase in image-token attention, whereas GR00T remains comparatively stable.

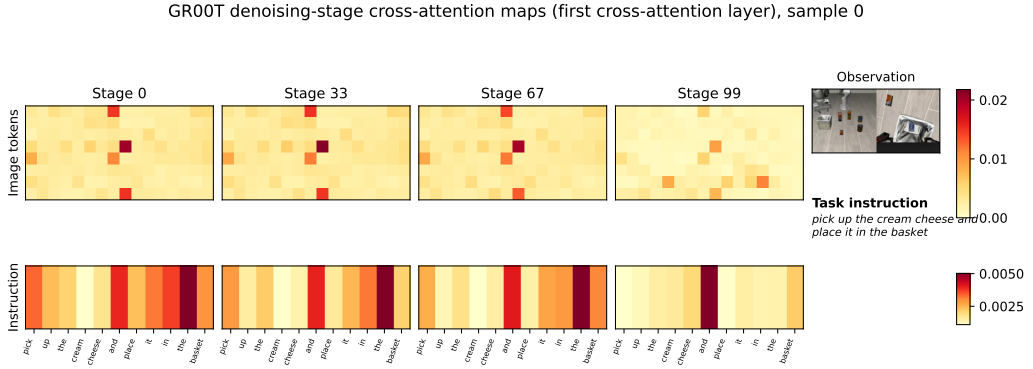
B.1.3 Qualitative Attention Maps and Self-Attention

To complement the averaged attention curves, we further visualize representative first-layer attention maps. As shown in Fig. 12(a), π_0 assigns stronger attention to image tokens at later denoising stages, particularly near the end of the trajectory, while task-instruction attention does not show a clear monotonic trend. This qualitative observation is consistent with the averaged curves and further illustrates the increased explicit emphasis on visual information during late-stage denoising.

In contrast, Fig. 12(b) shows that GR00T does not simply shift its late-stage cross-attention toward raw image or task-instruction tokens. In this representative example, both image-token and task-instruction attention become weaker near the final stage. Together with the averaged results in Fig. 11, this suggests that GR00T reallocates late-stage attention toward other contextualized token



(a) π_0



(b) GR00T

Figure 12: Representative first-layer cross-attention maps of π_0 and GR00T at different denoising stages. For π_0, attention to image tokens becomes stronger toward later stages, while task-instruction attention does not exhibit a clear monotonic trend. For GR00T, late-stage cross-attention does not simply shift toward raw image or task-instruction tokens, indicating that different VLA action heads may realize stage-dependent conditioning through different token pathways.

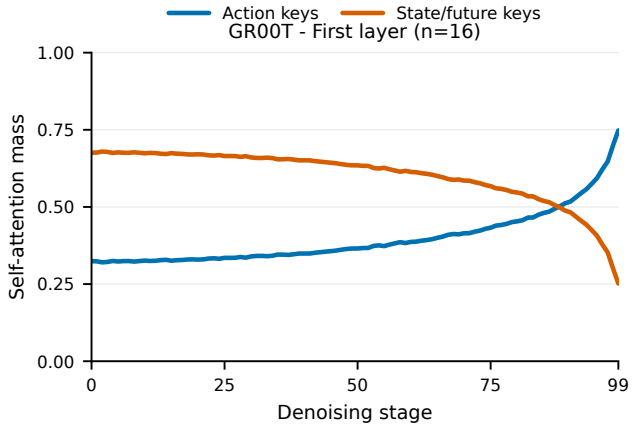


Figure 13: First-layer self-attention allocation in GR00T across denoising stages. Attention to action keys increases toward later stages, while attention to state/future keys decreases.

positions, such as grounding- or template-related representations. These results further indicate that different VLA action heads can exhibit stage-dependent conditioning behavior through different token pathways.

Table 4: Complete LIBERO results across GR00T-style and π-style VLA backbones. M denotes the number of 5-action low-level chunks executed between two high-level policy updates. The hierarchical GR00T- and π-based models use high-level action horizons of $H = 32$ and $H = 20$, respectively. Latency is reported per native returned action chunk. The non-hierarchical StarVLA baselines use 8-action chunks, whereas our low-level refiner outputs 5-action chunks.

Method	M	**Chunk Size**	**Spatial**	**Object**	**Goal**	**Long**	**Avg.**	**Δ Avg.**	**Latency**
GR00T-style backbone									
StarVLA-GR00T	–	8	97.8	98.8	97.4	92.0	96.5	–	115.0
Ours (GR00T-base)	2	5	98.4	98.8	97.6	94.8	97.40	+0.90	68.7
Ours (GR00T-base)	3	5	**99.2**	98.8	**97.8**	94.0	97.45	+0.95	52.3
Ours (GR00T-base)	4	5	98.0	**99.6**	**97.8**	**95.8**	**97.80**	**+1.30**	44.2
Ours (GR00T-base)	5	5	98.4	98.8	97.4	93.6	97.05	+0.55	**39.3**
π-style backbone									
StarVLA-π	–	8	99.2	99.0	**97.2**	95.8	97.8	–	183.3
Ours (π-base)	2	5	99.2	99.4	96.2	95.8	97.7	-0.1	70.7
Ours (π-base)	3	5	99.4	**99.6**	96.8	96.2	98.0	+0.2	53.1
Ours (π-base)	4	5	**99.6**	99.0	96.8	**97.4**	**98.3**	**+0.5**	**44.5**

In addition to cross-attention, we analyze self-attention within the GR00T action head. As shown in Fig. 13, the attention mass assigned to action keys increases throughout denoising, while the attention to state/future keys decreases. This suggests that later denoising increasingly operates on the evolving action representation itself. One possible interpretation is that, once task-level action structure has formed, subsequent denoising increasingly refines the existing action representation, while external conditioning information may already have been selectively read and integrated at earlier stages or shallower action-head layers.

Finally, we emphasize that attention mass does not provide a strict causal attribution of conditioning information. It measures only the explicit attention allocated to different key-token groups at a particular layer and denoising stage, rather than their causal contribution to the final action prediction. Stronger causal evidence requires intervention-based analyses, such as masking image or instruction tokens and measuring the resulting change in action prediction. Nevertheless, the attention analysis consistently shows that both GR00T and π_0 exhibit denoising-stage-dependent conditioning patterns, providing additional empirical evidence for stage-dependent conditioning behavior in generative VLA action formation.

B.2 Generalization Across VLA Backbones

The main experiments report the overall results of our method on both GR00T-style and π-style generative VLA backbones. Here, we provide the complete LIBERO results under different high-level reuse horizons, including per-suite success rates and native inference latency.

As shown in Table 4, our method maintains strong task performance across different reuse horizons on both backbones while substantially reducing inference cost. For the GR00T-style backbone, the 5×4 setting achieves the highest average success rate of 97.8%, while the 5×5 setting still maintains 97.05% success with lower latency. For the π-style backbone, performance improves from 97.8% for the StarVLA-π baseline to 98.3% at 5×4, with a native latency of 44.5 ms per 5-action chunk.

These results further support the cross-backbone generality of the proposed hierarchical refinement framework.

B.3 Long-Horizon Action Generation and Reuse

We further investigate how the proposed hierarchical architecture behaves when the high-level policy is extended to substantially longer action horizons. In addition to the default setting, we train high-level GR00T policies with action horizons of $H = 48$ and $H = 64$, together with their cor-

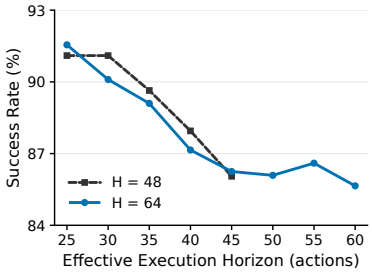


Figure 14: Effect of long-horizon high-level trajectory reuse. We report the average LIBERO success rate as the cached high-level trajectory is reused for increasingly long execution horizons. Both curves are shown from 25 executed actions onward, corresponding to the shared evaluation range of the $H = 48$ and $H = 64$ models.

responding low-level refiners. The refiner predicts 5-step action chunks, and the cached high-level trajectory is refreshed after executing M consecutive refinement chunks. Therefore, one high-level policy invocation corresponds to an effective execution horizon of $5M$ actions. For $H = 48$ and $H = 64$, the maximum refinement horizons used during training are $5{\times}9 = 45$ and $5{\times}12 = 60$ actions, respectively.

As shown in Fig. 14, task performance generally decreases as the same high-level trajectory is reused for a longer period. For $H = 48$, the average success rate remains at 91.10% for execution horizons of 25 and 30 actions, but gradually decreases to 86.05% when the trajectory is reused for 45 actions. A similar trend is observed for $H = 64$: the success rate decreases from 91.55% at 25 actions and remains around 85%–87% when the reuse horizon is extended to 40–60 actions. Notably, increasing the high-level prediction horizon from $H = 48$ to $H = 64$ does not provide a clear performance advantage under long reuse. This suggests that simply training the upper-level policy to predict farther into the future does not directly translate into a proportionally longer effective reuse horizon.

These results reveal an important limitation of the current hierarchical action-generation architecture: although a high-level trajectory can be reused across multiple low-level refinement cycles, its effective reuse range is not unlimited. When the high-level policy generates and caches an increasingly long future plan, the subsequent execution must continue to operate under an evolving closed-loop state. As execution proceeds, uncertainty arising from action errors, observation changes, and environment interactions may accumulate, causing the cached high-level trajectory to become progressively less consistent with the current state. Although the low-level refiner can use the latest visual observation to locally adjust the executable actions, its refinement remains conditioned on the cached high-level trajectory. Consequently, when the upper-level plan spans an excessively long timescale and is reused without refresh, local refinement may no longer be sufficient to compensate for the growing mismatch between the cached plan and the actual closed-loop state.

From this perspective, the efficiency benefit of hierarchical VLA inference cannot be obtained simply by continuously increasing the high-level prediction horizon or reuse length. A longer cached trajectory further amortizes the expensive high-level VLA invocation, but simultaneously reduces the frequency at which the high-level decision process can respond to newly observed states. Our results therefore suggest a practical trade-off between high-level reuse and closed-loop responsiveness: within a moderate reuse range, low-level refinement can effectively provide local closed-loop correction while reducing high-level inference frequency; beyond this range, accumulated uncertainty and plan–state mismatch may exceed what the local refiner can compensate for, leading to substantial performance degradation.

This observation also motivates a natural direction for future work. Rather than refreshing the high-level trajectory after a fixed number of refinement cycles, an adaptive mechanism could trigger a new high-level VLA invocation according to the observed state deviation, action uncertainty, or consistency between the cached trajectory and the current observation. Such an event-driven or uncertainty-aware high-level refresh strategy may provide a more flexible balance between inference efficiency and closed-loop robustness.

C Real-World Experimental Details

C.1 Real-World Platform and System Setup

We conduct real-world experiments on an AgileX PiperX 6-DoF robotic arm. The system uses a fixed top camera and a wrist-mounted Intel RealSense camera to provide dual-view RGB observations at a resolution of 640×480. The policy takes as input the language instruction, dual-view images, and an 8-dimensional robot state

$$s_t = [x, y, z, q_x, q_y, q_z, q_w, g],$$

where (x, y, z) denotes the end-effector position, (q_x, q_y, q_z, q_w) represents the end-effector orientation as a quaternion, and g denotes the gripper opening.

The policy outputs a 7-dimensional end-effector action

$$a_t = [\Delta x, \Delta y, \Delta z, \Delta r_x, \Delta r_y, \Delta r_z, g^*],$$

where the translational delta is defined in the robot world frame, the rotational delta is represented as an axis-angle increment in the local end-effector frame, and g^* denotes the absolute gripper target. During execution, the predicted end-effector target pose is converted to joint targets through local inverse kinematics.

For safe and consistent real-world evaluation, all methods use the same control constraints: the per-step translation and rotation are limited to $0.05\,\mathrm{m}$ and $0.20\,\mathrm{rad}$, respectively, and the maximum joint velocity is limited to $25^\circ/\mathrm{s}$. Before each trial, the robot returns to a fixed initial configuration, the internal action cache is cleared, and the cameras are warmed up.

Fruit25 and PushBlock are executed at control frequencies of 8 Hz and 20 Hz, respectively. Inference is synchronous: when the action buffer is empty, the control loop waits for a new action chunk and then executes the predicted actions sequentially at the corresponding control frequency. Independently of the policy control frequency, robot feedback is recorded at approximately 50 Hz for subsequent trajectory-quality and motion-smoothness analysis. All real-world experiments are conducted on the same NVIDIA RTX A4000 GPU.

C.2 Datasets and Data Processing

The two real-world environments are illustrated in Figs. 15 and 16. Fruit25 is collected through VR teleoperation and contains 25 tasks. Among them, 24 are single-fruit pick-and-place tasks formed by four fruit categories—apple, orange, lemon, and lime—and six target locations, including five colored baskets and one metal tray. The final task is a long-horizon multi-object fruit-sorting task. We perform quality control on 1,220 collected episodes through trajectory inspection, video inspection, and manual replay review, removing episodes with clear execution errors, corrupted recordings, or abnormal trajectory segments. This results in 1,186 valid demonstrations with 109,684 frames. PushBlock contains a single continuous pushing task in which the robot pushes a black block into a blue target region. The dataset contains 100 demonstrations with 32,776 frames. Both Fruit25 and PushBlock include top and wrist RGB observations, end-effector poses, gripper states, and language instructions.

The two datasets use different sampling and control frequencies. Fruit25 mainly consists of relatively discrete and quasi-static manipulation behaviors such as grasping, transporting, and placing objects, for which 8 Hz provides sufficient temporal resolution. In contrast, PushBlock requires sustained contact with the object and more frequent closed-loop corrections during execution, so we use a higher frequency of 20 Hz to provide finer-grained motion control.

All demonstrations are recorded only after robot initialization is completed, so initialization motion is excluded from the training trajectories. We do not apply a unified automatic trimming procedure to remove stationary segments at the beginning or end of trajectories; instead, low-quality episodes are removed as complete trajectories. The recorded end-effector poses are converted into temporal delta-EE actions. The translational increment is defined as

$$\Delta p_t = p_{t+1} - p_t,$$

and the rotational increment is defined as

$$\Delta r_t = \log\left(R_t^{-1} R_{t+1}\right).$$

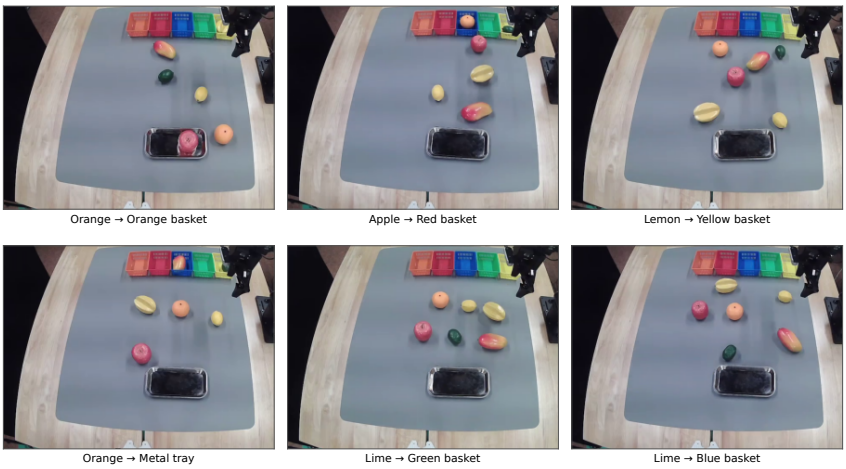


Figure 15: Example initial scenes from Fruit25.

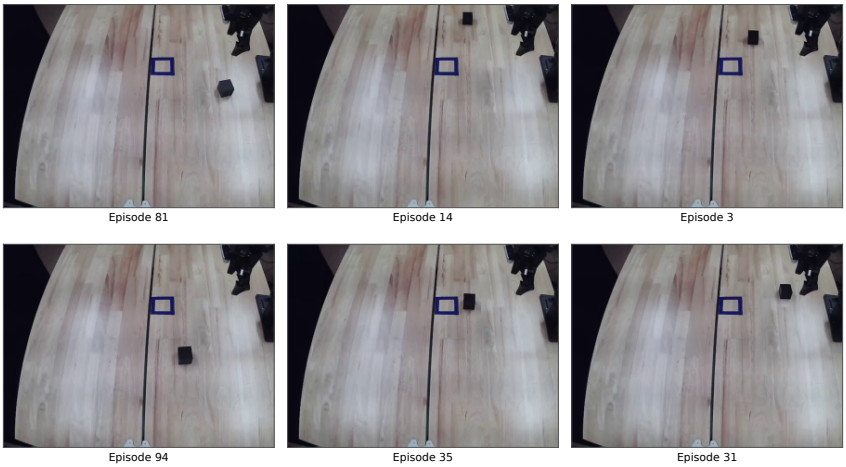


Figure 16: Example initial scenes from PushBlock.

The gripper command uses the absolute opening at the next timestep, and the motion delta of the final frame is set to zero.

C.3 Model Configuration and Training

We compare StarVLA-GR00T, StarVLA-π, and our hierarchical model. All three methods use Qwen3-VL-2B-Instruct as the vision-language backbone and are trained on the same data, language instructions, and state/action representations for each benchmark. StarVLA-GR00T and StarVLA-π are fully fine-tuned end-to-end and use an action horizon of 8.

For our method, we set the high-level action horizon to $H = 32$, the low-level chunk size to $h = 8$, and the reuse length to $M = 3$, corresponding to 24 executed actions between two high-level policy updates. During refiner training, the high-level partial-denoising progress is sampled from $[0.6, 1.0]$, the low-level refinement scale from $[0.8, 1.0]$, and the temporal-context trajectory is fully denoised. At inference time, we use a high-level denoising progress of 0.97, two high-level inference steps, and one low-level refinement step, with the refiner operating from an assumed denoising progress of 0.97.

For Fruit25, all models are trained for 30k steps, while for PushBlock they are trained for 10k steps. On Fruit25, StarVLA-GR00T and both stages of our method use a global batch size of 64, while StarVLA-π uses a global batch size of 32. On PushBlock, all three methods use an effective batch

Table 5: Task definitions and retained demonstration counts for the real-world datasets.

Task ID	Benchmark	Task Description	Retained Demonstrations
1	Fruit25	Pick up the apple and place it into the red basket.	100
2	Fruit25	Pick up the apple and place it onto the metal tray.	100
3	Fruit25	Pick up the orange and place it onto the metal tray.	100
4	Fruit25	Pick up the orange and place it into the orange basket.	96
5	Fruit25	Pick up the lemon and place it onto the metal tray.	98
6	Fruit25	Pick up the lemon and place it into the yellow basket.	99
7	Fruit25	Pick up the lime and place it into the green basket.	98
8	Fruit25	Pick up the apple and place it into the orange basket.	20
9	Fruit25	Pick up the apple and place it into the blue basket.	20
10	Fruit25	Pick up the apple and place it into the green basket.	19
11	Fruit25	Pick up the apple and place it into the yellow basket.	20
12	Fruit25	Pick up the orange and place it into the red basket.	19
13	Fruit25	Pick up the orange and place it into the blue basket.	20
14	Fruit25	Pick up the orange and place it into the green basket.	20
15	Fruit25	Pick up the orange and place it into the yellow basket.	20
16	Fruit25	Pick up the lemon and place it into the orange basket.	20
17	Fruit25	Pick up the lemon and place it into the red basket.	20
18	Fruit25	Pick up the lemon and place it into the blue basket.	20
19	Fruit25	Pick up the lemon and place it into the green basket.	20
20	Fruit25	Pick up the lime and place it into the orange basket.	20
21	Fruit25	Pick up the lime and place it into the red basket.	20
22	Fruit25	Pick up the lime and place it into the blue basket.	20
23	Fruit25	Pick up the lime and place it into the yellow basket.	13
24	Fruit25	Pick up the lime and place it onto the metal tray.	84
25	Fruit25	Sort the fruits into their corresponding baskets.	100
1	PushBlock	Push the black block into the blue target region.	100

size of 8. We use a cosine learning-rate schedule with 1,000 warm-up steps, a main learning rate of 10^{-4} for the action-related modules, and 2.5×10^{-5} for the vision-language backbone.

C.4 Evaluation Protocol

All methods are evaluated in the same real-world environment with fixed camera poses, lighting conditions, background, and robot initial configuration. Since object placement is performed manually, identical object poses cannot be guaranteed across methods; instead, we maintain similar initial layouts across trials.

For Fruit25, Tasks 1–7, 24, and 25 are evaluated with 10 trials per method, while Tasks 8–23 are evaluated with 5 trials per method, resulting in 170 trials for each method. For PushBlock, each method is evaluated with 10 trials, which may be terminated earlier once the task is clearly completed or has failed.

All tasks use manually assigned binary success/failure labels. For the atomic Fruit25 tasks, a trial is considered successful if the specified fruit is grasped and placed into the instructed basket or onto the metal tray. For the Sort Fruits task, four fruits are present in the scene, and the trial is considered successful if at least three are correctly sorted into their corresponding destinations. For PushBlock, success is defined as pushing the black block into the blue target region.

C.5 Detailed Main Results

We further report detailed success rates and latency results for the two benchmarks. Fruit25 is primarily composed of quasi-static pick-and-place tasks and is executed at 8 Hz; therefore, an 8-step action chunk corresponds to approximately 1 s of execution. As a result, inference is invoked relatively sparsely and the latency difference is less visually apparent during execution. In contrast, PushBlock runs at 20 Hz, where the same 8-step chunk covers only about 0.4 s, making infer-

Table 6: Per-task success results on the real-world benchmarks. Fruit25 tasks are evaluated with either 5 or 10 trials, while PushBlock is evaluated with 10 trials per method.

ID	Task	StarVLA-GR00T	StarVLA-π	Ours
	Fruit25			
1	Apple → Red basket	7/10 (70%)	6/10 (60%)	**10/10 (100%)**
2	Apple → Metal tray	**10/10 (100%)**	9/10 (90%)	9/10 (90%)
3	Orange → Metal tray	9/10 (90%)	9/10 (90%)	9/10 (90%)
4	Orange → Orange basket	5/10 (50%)	**8/10 (80%)**	7/10 (70%)
5	Lemon → Metal tray	10/10 (100%)	10/10 (100%)	10/10 (100%)
6	Lemon → Yellow basket	8/10 (80%)	8/10 (80%)	8/10 (80%)
7	Lime → Green basket	9/10 (90%)	9/10 (90%)	**10/10 (100%)**
8	Apple → Orange basket	**4/5 (80%)**	3/5 (60%)	3/5 (60%)
9	Apple → Blue basket	4/5 (80%)	4/5 (80%)	**5/5 (100%)**
10	Apple → Green basket	3/5 (60%)	3/5 (60%)	3/5 (60%)
11	Apple → Yellow basket	2/5 (40%)	**4/5 (80%)**	3/5 (60%)
12	Orange → Red basket	4/5 (80%)	4/5 (80%)	**5/5 (100%)**
13	Orange → Blue basket	3/5 (60%)	**5/5 (100%)**	**5/5 (100%)**
14	Orange → Green basket	**5/5 (100%)**	4/5 (80%)	3/5 (60%)
15	Orange → Yellow basket	3/5 (60%)	**4/5 (80%)**	3/5 (60%)
16	Lemon → Orange basket	**4/5 (80%)**	3/5 (60%)	1/5 (20%)
17	Lemon → Red basket	4/5 (80%)	4/5 (80%)	**5/5 (100%)**
18	Lemon → Blue basket	4/5 (80%)	**5/5 (100%)**	**5/5 (100%)**
19	Lemon → Green basket	4/5 (80%)	4/5 (80%)	**5/5 (100%)**
20	Lime → Orange basket	**3/5 (60%)**	**3/5 (60%)**	2/5 (40%)
21	Lime → Red basket	**5/5 (100%)**	4/5 (80%)	**5/5 (100%)**
22	Lime → Blue basket	5/5 (100%)	5/5 (100%)	5/5 (100%)
23	Lime → Yellow basket	3/5 (60%)	4/5 (80%)	**5/5 (100%)**
24	Lime → Metal tray	**10/10 (100%)**	6/10 (60%)	**10/10 (100%)**
25	Sort Fruits	**2/10 (20%)**	1/10 (10%)	**2/10 (20%)**
Overall	**170 trials**	**130/170 (76.5%)**	**129/170 (75.9%)**	**138/170 (81.2%)**
	PushBlock			
1	Push black block into blue target region	3/10 (30%)	2/10 (20%)	**6/10 (60%)**



Figure 17: Examples of relatively easy PushBlock initial configurations, where the block is approximately aligned with the target region and can be pushed along a near-straight trajectory.

ence latency more critical to continuous closed-loop control. In addition, real-world deployment introduces fixed overhead from image transmission, inverse kinematics, and robot control, which partially reduces the relative latency gain compared with simulation.

The advantage of our method is more pronounced on PushBlock. StarVLA-GR00T and StarVLA-π mainly succeed in simple cases where the block is approximately aligned below the target region and can be pushed along a near-straight trajectory, as illustrated in Fig. 17. With a longer high-level horizon, our method can form a more extended motion plan and succeeds repeatedly under more challenging initial layouts requiring larger directional corrections. Detailed per-task success rates for both Fruit25 and PushBlock are reported in Table 6.

Table 7: Motion smoothness and execution continuity on Fruit25 trials. Values are reported as median [Q1, Q3]. Higher SPARC and lower jerk and stop ratio indicate smoother and more continuous execution.

Method	T-SPARC ↑	R-SPARC ↑	T-Jerk ↓	R-Jerk ↓	Stop (%) ↓
StarVLA-GR00T	-5.480 [-5.817, -5.127]	-10.499 [-11.932, -9.440]	9.599 [9.408, 9.788]	10.026 [9.833, 10.216]	12.537 [10.575, 15.102]
StarVLA-π	-6.799 [-7.266, -6.233]	-12.134 [-13.370, -11.135]	9.939 [9.681, 10.212]	10.355 [10.091, 10.599]	26.066 [23.566, 28.767]
Ours	**-5.151 [-5.398, -4.887]**	**-9.992 [-11.099, -8.959]**	**9.484 [9.300, 9.644]**	**9.909 [9.733, 10.091]**	**11.274 [8.974, 13.607]**

C.6 Motion Smoothness and Execution Continuity

We further analyze the motion smoothness and execution continuity of the real robot using independently recorded high-rate feedback. Since PushBlock contains only 10 evaluation trials per method, we restrict this analysis to Fruit25. Both successful and failed trials are included. Table 7 reports the median and interquartile range (IQR) over these trials. We use median-based statistics to reduce sensitivity to occasional outlier trajectories and long execution pauses.

We use three complementary measures of motion quality. **Translational SPARC** and **Rotational SPARC** are spectral arc-length measures computed from the magnitudes of the end-effector linear and angular velocities (Balasubramanian et al., 2015), respectively; values closer to zero indicate smoother motion. **Translational** and **Rotational dimensionless jerk** measure the temporal variation of translational and rotational motion after normalization by trajectory duration and motion magnitude (Hogan & Sternad, 2009); we report the same $\log_{10}$ dimensionless jerk cost used throughout our analysis, where lower values indicate smoother motion. **Stop ratio** is the fraction of the active-motion interval for which the end-effector linear velocity falls below $0.01\,\mathrm{m/s}$, with lower values indicating fewer execution pauses.

As shown in Table 7, our hierarchical policy produces smoother and more continuous real-world execution, with fewer pauses.