TRAJSCALE-VLA:TRAJECTORY ALIGNED SCALE AUTOREGRESSIVE ACTION GENERATION FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive vision-language-action models (VLAs) rely on action tokenization to represent continuous trajectories as discrete sequences for prediction. A key challenge is that fine-grained action tokens preserve control precision but lead to long autoregressive sequences, while hierarchical tokenization can shorten decoding but does not inherently determine how trajectory information should be organized across scales. Existing tokenizers impose an ordering or multi-scale grouping on action tokens, but do not explicitly align each token group with the trajectory structure it is expected to represent. To address this challenge, we propose **TrajScale-VLA**, which uses trajectory supervision to guide the learning of an action token hierarchy, with early predictions capturing coarse motion and subsequent predictions recovering local detail. We first train a multi-scale residual tokenizer with a multi-scale trajectory refinement (MTR) objective that aligns cumulative reconstructions with targets at progressively finer temporal resolutions. This supervision assigns each scale a distinct reconstruction task within the action hierarchy. We then train a policy to predict the resulting token groups autoregressively across scales and in parallel within each scale, reducing the number of sequential decoding steps. TrajScale-VLA outperforms all compared StarVLA baselines, achieving success rates of 98.4% on LIBERO and 59.8% on RoboCasa-GR1. On real-world *USB Pick-and-Insert*, which requires coordinated transport and fine alignment, it surpasses the best baseline in each setting by 13.3 and 26.7 percentage points under in-distribution and out-of-distribution conditions. These results highlight the importance of organizing action representations through trajectory supervision for autoregressive robot control.

1 INTRODUCTION

Vision-language-action models (VLAs) provide a unified framework for multi-task robotic manipulation by mapping visual observations and language instructions to robot actions (Brohan et al., 2023; Kim et al., 2024). Their action generation mechanisms commonly follow two approaches, modeling actions in continuous space, as in π_0 and $\pi_{0.5}$ (Black et al., 2024; Intelligence et al., 2025), or predicting discrete action token sequences (Pertsch et al., 2025; Wang et al., 2025; Liu et al., 2025). Autoregressive VLAs follow the latter approach by using an action tokenizer to encode continuous trajectories into discrete sequences and training the policy to predict these sequences conditioned on visual observations and language instructions. The predicted tokens are then decoded into continuous actions for robot execution.

Tokenizer training determines how information about an entire action chunk is distributed across tokens and organized for prediction. While language tokens follow textual order, action tokens need not follow the temporal order of execution. Recent tokenizers organize action information from coarse motion structure to finer detail, using ordered tokens or multi-scale token groups to progressively refine the reconstructed trajectory. OAT learns ordered tokens that support decoding from prefixes, allowing additional tokens to refine action reconstructions (Liu et al., 2026). MINT uses frequency-domain reconstruction supervision to encourage coarse tokens to capture low-frequency motion structure and finer tokens to encode high-frequency details (Huang et al., 2026).

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.

These token hierarchies support progressive refinement, but the temporal structures that each intermediate reconstruction should recover remain implicit. When different token prefixes or scales share the complete trajectory as their reconstruction target, they are trained toward the same output without distinct temporal-resolution objectives. This supervision does not ensure that early reconstructions establish a coarse trajectory for subsequent stages to refine.

We propose **TrajScale-VLA**, which combines a trajectory-aligned action tokenizer with a scale-autoregressive policy. The tokenizer uses multi-scale residual quantization, with multi-scale trajectory refinement (MTR) supervising intermediate reconstructions against targets at corresponding temporal resolutions. To construct these targets, we average expert actions within temporal windows and linearly interpolate the pooled values to the original sequence length. Larger windows provide coarse motion summaries, whereas smaller windows retain more local variation. MTR compares the shared decoder's output from each intermediate cumulative latent representation with the corresponding target. After tokenizer training, the policy learns to autoregressively generate successive token groups conditioned on images, language instructions, and robot state, predicting all tokens within each scale in parallel. The frozen decoder converts the accumulated latent representation into continuous actions. Across LIBERO and RoboCasa-GR1, our model outperforms every baseline provided by StarVLA in task success rate. These results motivate closer examination of how intermediate supervision shapes the temporal structure of action representations and their use in robot control.

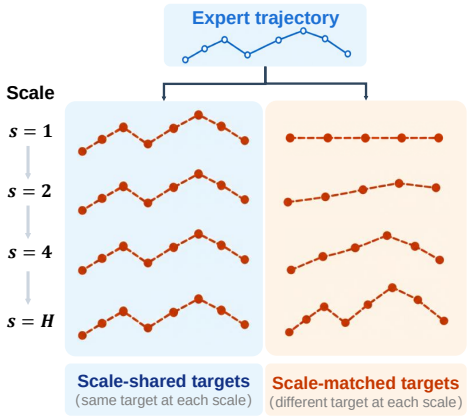


Figure 1: **Supervision targets across scales.** Shared full-trajectory targets versus scale-matched targets.

Our contributions are threefold:

- We introduce multi-scale trajectory refinement (MTR), a trajectory-aligned supervision objective for hierarchical action tokenization. MTR supervises cumulative token representations against scale-matched trajectory targets, explicitly specifying the temporal structure each scale should encode and organizing action representations from coarse motion structure to finer trajectory details.
- We propose **TrajScale-VLA**, combining a trajectory-aligned tokenizer with a scale-autoregressive policy. Its MTR objective aligns shared-decoder reconstructions of cumulative latent representations with targets constructed through average pooling and linear interpolation.
- TrajScale-VLA achieves higher task success rates than all baselines provided by StarVLA on both LIBERO and RoboCasa-GR1. Supervision ablations and intermediate reconstruction analyses further support the effectiveness of scale-matched trajectory supervision.

2 PRELIMINARIES

Autoregressive VLAs with VQ action tokenizers. Given visual observations, a language instruction, and robot state, collectively denoted by c, a VLA policy predicts an action chunk $A \in \mathbb{R}^{H \times d_a}$ with horizon H and action dimension d_a. In a basic single-scale vector-quantized action tokenizer, an encoder E produces N latent vectors $F = E(A) = (f_1, \ldots, f_N)$ (Wang et al., 2025). Each vector f_i is matched to its nearest entry in a codebook of V learned vectors, $\mathcal{C} = \{e_v\}_{v=1}^{V}$, yielding the discrete index $z_i = \arg\min_{1 \le v \le V} \|f_i - e_v\|_2^2$. The selected codebook vectors form the quantized representation $\hat{F} = (e_{z_1}, \ldots, e_{z_N})$, which a decoder D reconstructs into actions as $\hat{A} = D(\hat{F})$. The resulting indices constitute the token sequence $z = (z_1, \ldots, z_N)$, whose conditional distribution is

modeled autoregressively,

$$p_\theta(z \mid c) = \prod_{i=1}^{N} p_\theta(z_i \mid z_{<i}, c), \tag{1}$$

where θ denotes policy parameters and $z_{<i} = (z_1, \ldots, z_{i-1})$ is the preceding token prefix. The policy is trained by minimizing token-level cross-entropy using ground-truth token prefixes. At inference, the policy autoregressively generates an action token sequence conditioned on c, and the corresponding codebook vectors are retrieved and passed to D to produce continuous actions.

Multi-scale residual quantization. Multi-scale residual quantization extends the codebook matching and feature reconstruction steps between the encoder E and decoder D through residual accumulation across scales (Tian et al., 2024). We describe a simplified formulation for temporal features. Let $s_1 < \cdots < s_K$ denote the temporal resolutions of K scales, and let $\hat{F}_k \in \mathbb{R}^{N \times d_f}$ be the accumulated approximation of F after scale k, initialized with $\hat{F}_0 = 0$. At each scale, the remaining residual $F - \hat{F}_{k-1}$ is downsampled to s_k temporal positions while preserving the feature dimension d_f. Each of the resulting s_k feature vectors is then quantized, yielding a token group r_k. Its codebook vectors are then upsampled and added to the accumulated representation,

$$\begin{aligned} r_k &= \mathcal{Q}_k\Big(\mathrm{Down}_{s_k}(F - \hat{F}_{k-1})\Big), \\ \hat{F}_k &= \hat{F}_{k-1} + \mathrm{Up}(\mathrm{Lookup}_k(r_k)). \end{aligned} \tag{2}$$

Here, $\mathcal{Q}_k$ assigns nearest-codeword indices at scale k, and Lookup_k retrieves the corresponding codebook vectors. Down_{s_k} reduces the temporal resolution to s_k, while Up restores it to that of F. The token groups $(r_1, \ldots, r_K)$ are serialized into z, and the final accumulated representation is decoded into actions as $\hat{A} = D(\hat{F}_K)$.

3 METHOD

As shown in Figure 2, TrajScale-VLA combines a trajectory-aligned action tokenizer with a scale-autoregressive policy. We first introduce the motivation and design of MTR, followed by two-stage training and coarse-to-fine inference.

3.1 MULTI-SCALE TRAJECTORY REFINEMENT

Multi-scale residual quantization builds a hierarchy of latent representations, but leaves the temporal structure of intermediate action reconstructions implicit. This hierarchy alone does not ensure that early predictions establish a coarse trajectory for later scales to refine. We argue that supervision should explicitly guide this progression, with early reconstructions capturing coarse motion and later ones recovering finer temporal details. We therefore supervise intermediate reconstructions with trajectory targets at corresponding temporal resolutions.

Let $\hat{A}_k = D(\hat{F}_k)$ denote the full-length action reconstruction from the first k scales, where D is shared across scales. To construct its target, we average-pool the demonstrated action chunk A to s_k time steps and linearly interpolate the pooled sequence back to horizon H:

$$T_{s_k}(A) = \mathrm{Up}_H^{\mathrm{lin}}\left(\mathrm{AvgPool}_{s_k}(A)\right). \tag{3}$$

Each chosen s_k divides H, so each pooled value summarizes H/s_k consecutive actions. Larger windows provide coarse motion summaries, while smaller windows retain more local variation. All targets have the same length H: at $s_k = 1$, the target repeats the mean action over the chunk; at $s_k = H$, it recovers the complete action sequence, with $T_H(A) = A$.

Average pooling provides the best constant approximation within each window under squared error, making the window mean a representative summary of local actions. Linear interpolation then connects these summaries with gradual transitions. The formal properties of this construction and their proof are provided in Appendix A.

Based on these targets, we define the multi-scale trajectory refinement (MTR) objective:

$$\mathcal{L}_{\mathrm{mtr}} = \frac{1}{K-1} \sum_{k=1}^{K-1} \ell_w\Big(\hat{A}_k, T_{s_k}(A)\Big), \tag{4}$$

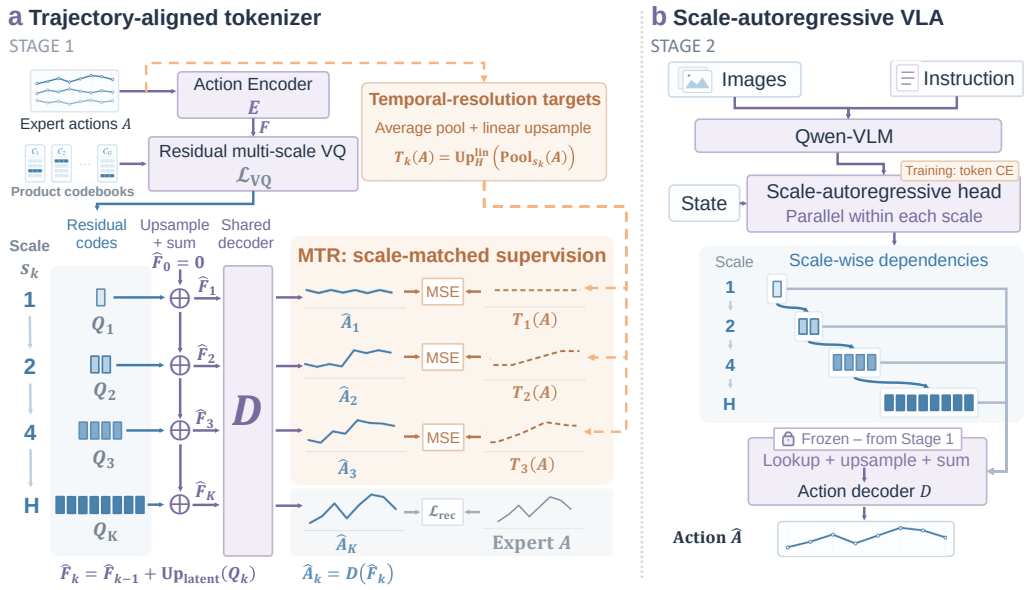


Figure 2: **Overview of TrajScale.** (a) A multi-scale residual tokenizer with a shared decoder. MTR aligns intermediate reconstructions with progressively finer temporal-resolution targets constructed by average pooling and linear interpolation. (b) Conditioned on images, instructions, and robot state, the VLA predicts tokens autoregressively across scales and in parallel within each scale. The accumulated latent representation is decoded into actions using the frozen Stage 1 decoder.

where ℓ_w denotes weighted mean squared error. Each term aligns the trajectory decoded from the first k scales with its corresponding target. MTR supervises the non-final cumulative reconstructions, while the final reconstruction $\hat{A}_K$ is supervised directly against the complete action chunk A.

3.2 Two-Stage Training

Stage 1: Trajectory-Aligned Action Tokenization. Following Equation 2, we apply Product-VQ (Jegou et al., 2010) to the downsampled residual features at each scale. Each of the s_k feature vectors is divided into G channel groups of dimension d_f/G, with each group quantized using its own codebook of V entries. The retrieved group embeddings are concatenated, upsampled, and added to the cumulative representation. The shared decoder reconstructs actions from these cumulative representations.

Alongside MTR, we supervise the final reconstruction against the complete demonstrated action sequence. A temporal-difference loss additionally matches the changes between consecutive actions, encouraging the reconstruction to follow the temporal variations in the demonstrations. For Product-VQ, the codebook loss updates the selected embeddings toward the residual features, while the commitment loss encourages these features to remain close to their assigned codewords. The complete Stage 1 objective is

$$\mathcal{L}_1 = \underbrace{\|\hat{A}_K - A\|^2_{w^{\text{rec}}}}_{\text{Final reconstruction}} + \underbrace{\lambda_{\text{vel}}\|\Delta\hat{A}_K - \Delta A\|^2_{w^{\text{vel}}}}_{\text{Temporal difference matching}}$$
$$+ \underbrace{\lambda_{\text{vq}} \sum_{k=1}^{K}\sum_{g=1}^{G} \frac{\|\,\text{sg}(u_{k,g}) - q_{k,g}\|_F^2 + \beta\|u_{k,g} - \text{sg}(q_{k,g})\|_F^2}{n_{k,g}}}_{\text{Codebook and commitment}} + \underbrace{\frac{\lambda_{\text{mtr}}}{K-1}\sum_{k=1}^{K-1}\|\hat{A}_k - T_{s_k}(A)\|^2_{w_k^{\text{mtr}}}}_{\text{MTR}}. \tag{5}$$

Here, $\Delta X_t = X_{t+1} - X_t$ denotes the first temporal difference, and $\|X\|_w^2 = \sum_i w_i X_i^2 / \sum_i w_i$ denotes weighted squared error, where i indexes scalar entries and w includes loss-specific weights

and action-dimension masks. For quantization, $u_{k,g}$ and $q_{k,g}$ are the downsampled residual features and their selected codebook embeddings for group g at scale k, with $n_{k,g}$ denoting their number of scalar entries; sg stops gradients, and β weights the commitment term. The coefficients λ_{vel}, λ_{vq}, and λ_{mtr} weight temporal-difference matching, quantization, and MTR, respectively. This objective jointly updates the encoder, codebooks, and decoder.

Stage 2: Scale-Autoregressive Policy Learning. After Stage 1, we freeze the action tokenizer and use its encoder E to map each demonstrated action chunk A to latent features $F = E(A)$. Multi-scale residual Product-VQ then converts F into K token groups $(r_1, \ldots, r_K)$ at resolutions $s_1, \ldots, s_K$. At scale k, each of the s_k downsampled residual vectors is split into G channel groups, each assigned the index of its nearest codebook entry. Thus, $r_k \in \{1, \ldots, V\}^{s_k \times G}$ contains $s_k G$ discrete tokens. We serialize these groups from coarse to fine into a sequence z of length $N_{\text{tok}} = G \sum_{k=1}^{K} s_k$ and store it in a token cache. During Stage 2 training, each cached sequence is paired with its corresponding conditioning input c and serves as the policy's prediction target.

The policy learns to predict these token groups from coarse to fine, conditioning each scale on c and the preceding groups $r_{<k} = (r_1, \ldots, r_{k-1})$. Within each scale, all temporal positions and Product-VQ groups are predicted in parallel. The resulting distribution factorizes as

$$p_\theta(z \mid c) = \prod_{k=1}^{K} \prod_{t=1}^{s_k} \prod_{g=1}^{G} p_\theta(r_{k,t,g} \mid r_{<k}, c)\,, \tag{6}$$

where $r_{k,t,g}$ denotes the token at temporal position t and channel group g of scale k. The first scale is predicted from c alone. During training, predictions at scale k are conditioned on the ground-truth token groups $r_{<k}$ retrieved from the token cache.

We train the policy by minimizing label-smoothed cross-entropy averaged over all tokens in each action chunk,

$$\mathcal{L}_2 = -\frac{1}{N_{\text{tok}}} \sum_{k=1}^{K} \sum_{t=1}^{s_k} \sum_{g=1}^{G} \sum_{v=1}^{V} \tilde{y}_{k,t,g,v} \log p_\theta(r_{k,t,g} = v \mid r_{<k}, c)\,, \tag{7}$$

where $\tilde{y}_{k,t,g,v} = (1-\varepsilon)\mathbb{I}[v = r_{k,t,g}] + \varepsilon/V$ is the smoothed target probability for codebook index v, $\mathbb{I}$ denotes the indicator function, and $\varepsilon = 0.02$. Normalization by N_{tok} gives equal weight to each token across scales and Product-VQ groups. This objective updates only the policy parameters θ, while the action tokenizer remains frozen.

3.3 Coarse-to-Fine Action Generation

At inference, the trained policy generates token groups using its own predictions as context. We encode the conditioning input c once and reuse it throughout generation. At scale k, the policy conditions on c and the previously generated groups $\hat{r}_{<k} = (\hat{r}_1, \ldots, \hat{r}_{k-1})$, selecting the most probable codebook index for each token,

$$\hat{r}_{k,t,g} = \arg \max_{v \in \{1,\ldots,V\}} p_\theta(r_{k,t,g} = v \mid \hat{r}_{<k}, c)\,. \tag{8}$$

The first scale uses only c. All $s_k G$ tokens within each scale are predicted in parallel, and the resulting group provides additional context for subsequent scales. Generating the complete token sequence therefore requires K sequential prediction rounds.

Once all scales have been generated, we reconstruct the action chunk using the frozen codebooks and shared decoder. At each scale, codebook lookup retrieves and concatenates the G group embeddings along the channel dimension. These features are upsampled to the common latent resolution and summed,

$$\hat{F}_K = \sum_{k=1}^{K} \text{Up}(\text{Lookup}_k(\hat{r}_k))\,, \qquad \hat{A} = D(\hat{F}_K). \tag{9}$$

The decoder is applied once to the final accumulated representation, producing the complete action chunk $\hat{A} \in \mathbb{R}^{H \times d_a}$.

Algorithm 1 TrajScale-VLA Training and Inference

Require: Demonstrations $\mathcal{D} = \{(c, A)\}$; encoder E, Product-VQ codebooks, decoder D; policy p_θ; scales $\{s_k\}_{k=1}^{K}$.

I. Training

for each Stage 1 training step **do**
  Sample action chunks A from $\mathcal{D}$; encode $F \leftarrow E(A)$.
  Obtain cumulative features $\{\hat{F}_k\}_{k=1}^{K}$ by residual Product-VQ using Eq. 2.
  Decode $\hat{A}_k \leftarrow D(\hat{F}_k)$ for $k = 1, \ldots, K$.
  Construct targets $T_{s_k}(A)$ for $k < K$ using Eq. 3.
  Update E, the codebooks, and D by minimizing $\mathcal{L}_1$ in Eq. 5.
end for
Freeze the tokenizer and cache token labels z for each demonstration.
for each Stage 2 training step **do**
  Sample paired conditioning inputs c and cached token labels z.
  Predict token groups conditioned on ground-truth $r_{<k}$ using Eq. 6.
  Update θ by minimizing $\mathcal{L}_2$ in Eq. 7.
end for

II. Inference

Encode current input c once; initialize an empty generated token prefix.
for $k = 1, \ldots, K$ **do**
  Predict all $s_k G$ tokens in $\hat{r}_k$ in parallel using Eq. 8.
  Append $\hat{r}_k$ to the generated prefix.
end for
Accumulate the retrieved features and decode $\hat{A}$ using Eq. 9.
return $\hat{A}$.

4 Experiments

We evaluate TrajScale-VLA on LIBERO and RoboCasa-GR1, followed by real-world USB grasping and insertion experiments with an xArm robot. We examine the contribution of scale-matched supervision through ablations and trajectory representation analysis.

4.1 Simulation Experiments

Experimental Setup and Evaluation Protocol. We evaluate TrajScale-VLA on LIBERO (Liu et al., 2023) and RoboCasa-GR1 (Nasiriany et al., 2024; Bjorck et al., 2025) using the StarVLA evaluation infrastructure (Community, 2026). Our LIBERO evaluation covers the Spatial, Object, Goal, and Long suites, each containing 10 tasks with 50 demonstrations per task. These suites cover spatial relations, object variation, task goals, and long-horizon manipulation. RoboCasa-GR1 contains 24 tabletop manipulation tasks for the GR1 humanoid robot, with 1,000 demonstrations per task (Ye et al., 2026). We report success rates for each LIBERO suite and their overall average, and the mean success rate across all 24 RoboCasa-GR1 tasks. We compare against representative autoregressive and continuous-action VLA baselines, with their robot-data pretraining settings indicated in Tables 1 and 2. Further details on the experimental setup are provided in Appendix B.1.

Base Model and Implementation. We implement TrajScale-VLA within StarVLA (Community, 2026) using a Qwen3-VL-4B Bai et al. (2025) backbone. We use action horizons of 8 and 16, with scales $\{1, 2, 4, 8\}$ and $\{1, 2, 4, 8, 16\}$, for LIBERO and RoboCasa-GR1, respectively. The tokenizer uses 16 product codebooks, each containing 512 entries. In Stage 1, we initialize the encoder and decoder through autoencoder training before training the residual quantization model with MTR. In Stage 2, we freeze the tokenizer and fine-tune the VLM and scale-autoregressive prediction head. We use AdamW with a VLM learning rate of 10^{-5}; the remaining trainable modules use 2×10^{-5} on LIBERO and 10^{-4} on RoboCasa-GR1. Further details on the base model (Qwen3-VL-4B) and training hyperparameters are provided in Appendix B.2.

Main Results. Tables 1 and 2 summarize the simulation results. The sources of the baseline success rates in Tables 1 and 2 are detailed in Appendix B.3. TrajScale-VLA achieves the highest average success rates among the compared methods on both benchmarks without prior robot-data policy pretraining. On LIBERO, it reaches 98.4%, exceeding StarVLA-α by 0.5 percentage points.

Table 1: **Success rates on LIBERO (%).** Best results are bold with green shading; second-best results use pale green, including ties. Robo. P.T. denotes prior robot-data policy pretraining, excluding generic VLM and standalone tokenizer pretraining.

Method	Robo. P.T.	Spatial	Object	Goal	Long	Avg. ↑
OpenVLA-OFT (Kim et al., 2025)	✓	97.6	98.4	97.9	94.5	97.1
π_0 (Black et al., 2024)	✓	96.8	98.8	95.8	85.2	94.2
π_0 + FAST (Pertsch et al., 2025)	✓	96.4	96.8	88.6	60.2	85.5
$\pi_{0.5}$ (Intelligence et al., 2025)	✓	98.8	98.2	98.0	92.4	96.9
GR00T-N1.6 (Bjorck et al., 2025)	✓	97.5	98.5	97.5	94.4	97.0
VLA-JEPA (Sun et al., 2026)	✓	96.2	99.6	97.2	95.8	97.2
StarVLA-FAST (Community, 2026)	✗	97.3	97.4	96.3	90.6	95.4
StarVLA-OFT (Community, 2026)	✗	97.8	98.6	96.2	93.8	96.6
StarVLA-π (Community, 2026)	✗	98.8	99.6	95.8	88.4	95.7
StarVLA-GR00T (Community, 2026)	✗	97.8	98.8	97.4	92.0	96.5
StarVLA-α (Ye et al., 2026)	✗	**99.0**	**99.8**	98.5	94.1	97.9
TrajScale-VLA (Ours)	✗	**99.0**	**99.8**	**98.8**	**96.0**	**98.4**

Table 2: **Average success on RoboCasa-GR1 (%).** Pretraining and highlighting follow Table 1. StarVLA averages are as reported.

Method	Robo. P.T.	Avg. ↑
GR00T-N1.6 (Bjorck et al., 2025)	✓	47.6
LangForce (Lian et al., 2026)	✗	52.6
VP-VLA (Wang et al., 2026b)	✗	53.8
DualCoT-VLA (Zhong et al., 2026)	✗	55.1
$\pi_{0.5}$ (Intelligence et al., 2025)	✓	37.0
FoMoVLA (Li et al., 2026)	✗	56.9
Qwen-VLA-Instruct (Wang et al., 2026a)	✓	56.7
ABot-M0 (Yang et al., 2026)	✓	58.3
StarVLA-FAST (Community, 2026)	✗	39.0
StarVLA-OFT (Community, 2026)	✗	48.8
StarVLA-π (Community, 2026)	✗	43.9
StarVLA-GR00T (Community, 2026)	✗	47.8
StarVLA-α (Ye et al., 2026)	✗	53.8
TrajScale-VLA (Ours)	✗	**59.8**

It matches the strongest baseline on Spatial and Object and achieves the highest success rates on Goal and Long, demonstrating consistent performance across all four suites. On RoboCasa-GR1, TrajScale-VLA achieves 59.8%, surpassing ABot-M0 by 1.5 percentage points and StarVLA-α by 6.0 percentage points. Compared with the autoregressive StarVLA-FAST baseline, the improvements are 3.0 and 20.8 percentage points on LIBERO and RoboCasa-GR1, respectively. Together, these results support the effectiveness of trajectory-aligned tokenization and scale-autoregressive generation across both manipulation benchmarks.

Effect of Intermediate Supervision. Table 3 compares intermediate supervision strategies within the same tokenizer backbone. MTR improves success over w/o MTR by 2.2 and 9.0 percentage points on LIBERO and RoboCasa-GR1, respectively, over OAT-style supervision by 1.4 and 8.2 points, and over MINT-style supervision by 1.2 and 6.5 points. MTR also achieves the lowest final reconstruction MSE on both benchmarks. On RoboCasa-GR1, its 9.0-point success gain over w/o MTR accompanies only a 1.35% reduction in final MSE, suggesting that reconstruction accuracy alone does not fully characterize downstream policy performance. Lower intermediate MSE against common scale-matched targets provides a diagnostic of better temporal-resolution alignment. Cumulative token-prefix reconstructions further show that MTR follows progressively refined targets, while OAT- and MINT-style variants approximate the complete trajectory at early scales (Figure 5).

The spectra of successive reconstruction increments reveal a clearer progression from low- to high-frequency corrections under MTR, consistent with coarse-to-fine organization (Figure 6). Diagnostic details are provided in Appendix C.

Table 3: **Ablation of intermediate supervision.** All variants except StarVLA-FAST share the tokenizer backbone. OAT- and MINT-style variants use full-trajectory and full DCT-spectrum targets, respectively. Inter. MSE denotes intermediate trajectory MSE against common scale-matched targets.

Method	Stage 1				Stage 2	
	Final MSE ↓		Inter. MSE ↓		Success (%) ↑	
	LIBERO	RoboCasa-GR1	LIBERO	RoboCasa-GR1	LIBERO	RoboCasa-GR1
StarVLA-FAST (Community, 2026)	4.056×10^{-4}	2.155×10^{-4}	—	—	95.4	39.0
w/o MTR	6.165×10^{-5}	3.632×10^{-6}	3.446×10^{-2}	6.754×10^{-3}	96.2	50.8
OAT-style (Liu et al., 2026)	5.969×10^{-5}	5.178×10^{-6}	1.140×10^{-3}	1.001×10^{-2}	97.0	51.6
MINT-style (Huang et al., 2026)	5.997×10^{-5}	5.127×10^{-6}	1.029×10^{-3}	1.000×10^{-2}	97.2	53.3
TrajScale-VLA (Ours)	$\mathbf{5.731\times10^{-5}}$	$\mathbf{3.583\times10^{-6}}$	$\mathbf{8.104\times10^{-4}}$	$\mathbf{6.777\times10^{-5}}$	**98.4**	**59.8**

4.2 Real-World Evaluation

Robot Platform and Tasks. We use an xArm manipulator equipped with a gripper, as shown in Figure 3. Two wrist-mounted cameras observe the tabletop and the target USB port, respectively. In the *USB Pick-and-Insert* task, the robot must grasp a USB connector from the tabletop and insert it into the port. This precision manipulation task is challenging because small positional or angular errors can prevent successful insertion, requiring coordinated reaching and grasping followed by accurate local alignment. It provides a practical setting to examine whether MTR's supervision across temporal resolutions helps the policy capture overall motion structure while preserving the fine action variations needed for insertion. Further details on the robot platform and tasks are provided in Appendix D.1.



Figure 3: **Real-world setup.** The xArm platform for USB grasping and insertion.

Data Collection and Training. We collect 720 demonstrations of *USB Pick-and-Insert* through VR teleoperation, comprising 116,298 recorded time steps and totaling 3.24 hours of demonstration data. Each demonstration includes RGB observations from both wrist-mounted cameras, robot states, and seven-dimensional actions consisting of end-effector translation and rotation increments and a gripper command. We use all demonstrations for training, following the two-stage procedure in Section 3.2. First, we train the action tokenizer on demonstrated action chunks with the Stage 1 objective. We then freeze the tokenizer, precompute the multi-scale token labels, and train the policy to predict these labels from the corresponding conditioning inputs. Detailed data collection and training configurations are provided in Appendix D.2.

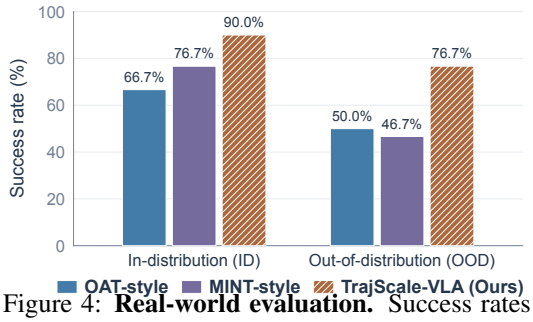


Figure 4: **Real-world evaluation.** Success rates under ID and OOD settings.

Evaluation and Results. We compare TrajScale-VLA with OAT-style and MINT-style supervision using the same tokenizer and policy backbones. Each method is evaluated over 30 trials under

each of two conditions. In-distribution (ID) trials use initial configurations within the demonstration distribution, whereas out-of-distribution (OOD) trials introduce unseen positions for both the USB connector and the socket. Detailed evaluation settings and success criteria are provided in Appendix D.3. All methods are tested on the same initial layouts. A trial is successful if the robot grasps the connector and fully inserts it into the socket without human intervention. Figure 4 summarizes the comparison. OAT-style achieves success rates of 66.7% and 50.0% under ID and OOD conditions, respectively, while MINT-style achieves 76.7% and 46.7%. The corresponding success rates for TrajScale-VLA are 90.0% and 76.7%. Representative rollout analyses are provided in Appendix D.4.

5 Related Work

Action Generation in VLA Models. VLA models commonly generate actions in continuous space or predict discrete action tokens. Continuous approaches include flow matching in π_0 (Black et al., 2024) and the action expert of $\pi_{0.5}$ (Intelligence et al., 2025), and direct regression in OpenVLA-OFT (Kim et al., 2025). For discrete prediction, the central design choice is how to tokenize continuous actions. Early methods use scalar discretization, with RT-2 (Brohan et al., 2023) and OpenVLA (Kim et al., 2024) assigning each action dimension to a discrete bin. To exploit temporal redundancy beyond individual action values, FAST (Pertsch et al., 2025) introduces analytical compression through the discrete cosine transform, coefficient quantization, and byte-pair encoding. Moving from predefined signal transformations to learned latent representations, neural tokenizers train encoders and codebooks for action reconstruction. VQ-BeT (Lee et al., 2024) uses residual vector quantization, while VQ-VLA (Wang et al., 2025) integrates a convolutional residual VQ-VAE trained on real and simulated trajectories into VLA policies. Beyond learning codebooks, a further question is which token structures are best suited to action prediction in VLA models.

Structured Action Tokenization. To design token structures suited to VLA prediction, FASTer (Liu et al., 2025) organizes action chunks along temporal and action dimensions and adopts block-wise autoregressive decoding. ActionCodec (Dong et al., 2026) further examines how token stability, semantic alignment, and dependencies affect VLA learning. A related direction explores coarse-to-fine token representations, implementing this hierarchy through ordered prefixes or token groups at different temporal resolutions. OAT (Liu et al., 2026) randomly retains prefixes of different lengths and trains them to reconstruct complete action chunks. MINT (Huang et al., 2026) adopts multi-scale residual quantization and supervises cumulative reconstructions in the frequency domain. Both methods provide intermediate supervision, but use the complete trajectory as the reference in the time and frequency domains, respectively. TrajScale-VLA extends this direction by assigning different temporal-resolution targets to different scales. Constructed through average pooling and linear interpolation, these targets explicitly specify the trajectory structure each cumulative representation should recover.

6 Conclusion

We presented TrajScale-VLA, which learns a trajectory-aligned action token hierarchy for coarse-to-fine generation. Its MTR objective assigns intermediate reconstructions targets at progressively finer temporal resolutions, giving each scale an explicit reconstruction task. The policy follows this hierarchy through autoregressive prediction across scales and parallel prediction within each scale. Experiments on LIBERO and RoboCasa-GR1, together with supervision ablations, support the effectiveness of scale-matched targets for downstream control. These findings highlight the importance of specifying what intermediate representations should recover when designing action tokenizers for autoregressive policies.

Limitations and Future Work. Our tokenizer is trained independently of the policy and remains frozen during policy learning, so its reconstruction objectives do not directly account for token prediction errors at inference. Future work could explore joint optimization or training with policy-generated tokens to better align representation learning with downstream prediction and improve robustness to token errors.

AI USE STATEMENT

We used generative AI tools solely for language polishing and improving the clarity of the manuscript. These tools were not used for research ideation, method development, code generation, experiments, or data analysis. We reviewed all AI-assisted edits and take full responsibility for the final content of this work, including its text, claims, and results.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

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.

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

StarVLA Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

Zibin Dong, Yicheng Liu, Shiduo Zhang, Baijun Ye, Yifu Yuan, Fei Ni, Jingjing Gong, Xipeng Qiu, Hang Zhao, Yinchuan Li, et al. Actioncodec: What makes for good action tokenizers. *arXiv preprint arXiv:2602.15397*, 2026.

Renming Huang, Chendong Zeng, Wenjing Tang, Jintian Cai, Cewu Lu, and Panpan Cai. Mimic intent, not just trajectories. *arXiv preprint arXiv:2602.08602*, 2026.

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.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*, 2024.

Wei Li, Peijin Jia, Yuan Ma, Xuefeng Jiang, Titong Jiang, Sheng Sun, Yujian Li, Xin Wen, Han Hong, Zhikang Liu, et al. Fomovla: Bridging visual foresight and motion guidance for vision-language-action models. *arXiv preprint arXiv:2607.14739*, 2026.

Shijie Lian, Bin Yu, Xiaopeng Lin, Laurence T Yang, Zhaolong Shen, Changti Wu, Yuzhuo Miao, Cong Huang, and Kai Chen. Langforce: Bayesian decomposition of vision language action models via latent action queries. *arXiv preprint arXiv:2601.15197*, 2026.

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.

Chaoqi Liu, Xiaoshen Han, Jiawei Gao, Yue Zhao, Haonan Chen, and Yilun Du. Oat: Ordered action tokenization. *arXiv preprint arXiv:2602.04215*, 2026.

Yicheng Liu, Shiduo Zhang, Zibin Dong, Baijun Ye, Tianyuan Yuan, Xiaopeng Yu, Linqi Yin, Chenhao Lu, Junhao Shi, Luca Jiang-Tao Yu, et al. Faster: Toward efficient autoregressive vision language action modeling via neural action tokenization. *arXiv preprint arXiv:2512.04952*, 2025.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

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.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. Vla-jepa: Enhancing vision-language-action model with latent world model. In *European Conference on Computer Vision*, pp. 478–497. Springer, 2026.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Advances in neural information processing systems*, 37:84839–84865, 2024.

Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, et al. Qwen-vla: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026a.

Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. Vq-vla: Improving vision-language-action models via scaling vector-quantized action tokenizers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11089–11099. IEEE, 2025.

Zixuan Wang, Yuxin Chen, Yuqi Liu, Jinhui Ye, Pengguang Chen, Changsheng Lu, Shu Liu, Bei Yu, and Jiaya Jia. Vp-vla: Visual prompting as an interface for vision-language-action models. *arXiv preprint arXiv:2603.22003*, 2026b.

Yandan Yang, Shuang Zeng, Tong Lin, Xinyuan Chang, Dekang Qi, Junjin Xiao, Haoyun Liu, Ronghan Chen, Yuzhi Chen, Dongjie Huo, et al. Abot-m0: Vla foundation model for robotic manipulation with action manifold learning. *arXiv preprint arXiv:2602.11236*, 2026.

Jinhui Ye, Ning Gao, Senqiao Yang, Jinliang Zheng, Zixuan Wang, Yuxin Chen, Pengguang Chen, Yilun Chen, Shu Liu, and Jiaya Jia. StarVLA-α: Reducing complexity in vision-language-action systems. *arXiv preprint arXiv:2604.11757*, 2026.

Zhide Zhong, Junfeng Li, Junjie He, Haodong Yan, Xin Gong, Guanyi Zhao, Yingjie Cai, Jiantao Gao, Xu Yan, Bingbing Liu, et al. Dualcot-vla: Visual-linguistic chain of thought via parallel reasoning for vision-language-action models. *arXiv preprint arXiv:2603.22280*, 2026.

A Properties of Scale-Matched Trajectory Targets

Proposition 1 (Optimality of window averaging)**.** *Let $a_1, \dots, a_m \in \mathbb{R}^{d_a}$ be the actions within a temporal window, with $m \geq 1$, and let $\mu = \frac{1}{m}\sum_{i=1}^{m} a_i$. Then μ uniquely minimizes the squared approximation error $\sum_{i=1}^{m} \|a_i - v\|_2^2$ over all constant vectors $v \in \mathbb{R}^{d_a}$.*

Proof. Since $\sum_{i=1}^{m}(a_i - \mu) = 0$, expanding the squared error gives

$$\sum_{i=1}^{m} \|a_i - v\|_2^2 = \sum_{i=1}^{m} \|a_i - \mu\|_2^2 + m\|\mu - v\|_2^2. \tag{10}$$

The first term is independent of v, and the second is nonnegative, with equality if and only if $v = \mu$. Thus, the window mean is the unique minimizer. □

Proposition 2 (Optimality of linear interpolation)**.** *Let $\tau_1 < \cdots < \tau_s$ be window centers in a time interval $[a, b]$, and let $\mu_1, \dots, \mu_s \in \mathbb{R}^{d_a}$ be their corresponding mean actions. Let $f^\star$ be the piecewise linear interpolant through (τ_j, μ_j), extended constantly on $[a, \tau_1]$ and $[\tau_s, b]$.*

Among all absolutely continuous curves $f : [a, b] \to \mathbb{R}^{d_a}$ with square-integrable derivatives satisfying $f(\tau_j) = \mu_j$, the curve $f^\star$ minimizes

$$\mathcal{E}(f) = \int_a^b \|f'(t)\|_2^2 \, dt. \tag{11}$$

Proof. For each pair of adjacent window centers, the interpolation constraints and the Cauchy–Schwarz inequality give

$$\int_{\tau_j}^{\tau_{j+1}} \|f'(t)\|_2^2 \, dt \geq \frac{\left\| \int_{\tau_j}^{\tau_{j+1}} f'(t)\, dt \right\|_2^2}{\tau_{j+1} - \tau_j} = \frac{\|\mu_{j+1} - \mu_j\|_2^2}{\tau_{j+1} - \tau_j}. \tag{12}$$

The linear interpolant attains equality because its derivative is constant on each such interval. On the two boundary intervals, constant extension attains the minimum derivative energy of zero. Summing these bounds proves that $f^\star$ minimizes $\mathcal{E}(f)$.

For $s = 1$, the constant curve $f^\star(t) = \mu_1$ satisfies the interpolation constraint and attains the minimum energy of zero. □

The discrete trajectory target $T_s(A)$ is obtained by evaluating this continuous interpolant at the original action time points.

B Implementation and Training Details

B.1 Detailed Experimental Setup

LIBERO. We follow the StarVLA data and evaluation pipeline (Community, 2026) on the Spatial, Object, Goal, and Long suites of LIBERO (Liu et al., 2023). Each suite contains ten tasks with 50 demonstrations per task, totaling 2,000 demonstrations across the four suites. The policy receives a fixed external view and a wrist view, each resized to 224×224, together with a language instruction and an eight-dimensional proprioceptive state. Actions are seven-dimensional, comprising end-effector translation and rotation commands and a gripper command. We normalize proprioceptive states and the six arm-action dimensions using the 1st and 99th percentiles of the dataset statistics, while leaving the gripper action unnormalized. Each training sample pairs the current observations with an eight-step action chunk. At evaluation, the policy executes all eight predicted actions before replanning. We conduct 50 trials per task using the benchmark- provided initial states, totaling 2,000 trials, and report the success rate for each suite and their overall average.

RoboCasa-GR1. We follow the StarVLA setup on RoboCasa- GR1 (Nasiriany et al., 2024; Bjorck et al., 2025; Ye et al., 2026), which contains 24 tabletop manipulation tasks with 1,000 demonstrations per task. The policy receives a single egocentric RGB view resized to 224×224, a language instruction, and the robot's proprioceptive state. Actions are 29-dimensional joint commands covering both arms, both hands, and the waist, and are normalized using dataset min–max statistics. The corresponding 29-dimensional joint state is encoded using sine and cosine functions, yielding a 58-dimensional proprioceptive input. Each training sample pairs the current observations with a 16-step action chunk. At evaluation, the policy executes the first 12 predicted actions before replanning. We conduct 50 trials per task, totaling 1,200 trials, and report the mean success rate across all 24 tasks.

B.2 Base Model and Training Hyperparameters

Base Model. We use Qwen3-VL-4B (Bai et al., 2025), the 4B dense variant of Qwen3-VL, as our pretrained vision-language backbone. It combines a SigLIP2-Large vision encoder, an MLP-based vision-language merger, and a Qwen3 language model. Its DeepStack mechanism injects visual features from multiple encoder layers into the language model. In TrajScale-VLA, the backbone processes camera observations and language instructions, and its output features condition the scale-autoregressive action head together with encoded robot states. During Stage 2, we fine-tune the VLM and the action prediction modules while keeping the action tokenizer frozen.

Stage 1: Action Tokenizer Training. We train the action tokenizer in two phases: continuous autoencoder initialization followed by residual Product-VQ training with MTR. Both phases use normalized action chunks from the training demonstrations. During autoencoder initialization, quantization is bypassed, and the encoder and decoder are optimized using reconstruction and temporal-difference losses. This provides an initial continuous representation before introducing the discrete bottleneck. Table 4 summarizes the tokenizer configuration and optimization settings. We describe the complete Stage 1 training procedure below, using RoboCasa-GR1 as an example.

For RoboCasa-GR1, we use complete 16-step windows of 29-dimensional absolute joint actions with per-dimension min–max normalization. We train the continuous autoencoder for 50 epochs using AdamW, a batch size of 512, and a learning rate of 2×10^{-4}. The initialization checkpoint minimizes the worst-group mean absolute reconstruction error accumulated during training, where groups are defined by dataset identity and task index. In the verified run, this selects the checkpoint after 50 epochs. We transfer its encoder and decoder weights to the Product-VQ tokenizer and initialize the codebooks from features encoded from 128 minibatches. The features are resampled to the configured temporal scales and pooled into an initialization set. Each product codebook is initialized by randomly sampling the corresponding channel-group features and adding Gaussian noise with standard deviation 10^{-4}.

We then train the quantized tokenizer for 100 epochs with a batch size of 512 and a constant learning rate of 4×10^{-5}. The encoder and decoder remain frozen for the first five epochs while the codebooks adapt; all three components are jointly optimized thereafter. MTR is active from the start of quantized training. Its coefficient is fixed at 0.1, while the VQ loss coefficient increases linearly to 0.05 over the first 20 epochs. The temporal-difference coefficient is 0.25, and the commitment coefficient is 0.2. Reconstruction, temporal-difference matching, and MTR cover all 29 action dimensions. The tokenizer checkpoint after 100 epochs is frozen and used to generate the offline token labels for Stage 2, producing 496 tokens per action chunk.

Both phases apply task-, trajectory-phase-, and timestep-dependent reconstruction weights. Samples from closing tasks receive a factor of 4, windows starting in the latter half of their valid trajectory-start range receive a factor of 2, and samples meeting both conditions receive an additional factor of 1.25. The resulting base sample weight is capped at 10. The final eight time steps in each action chunk receive a factor of 2. Static task-group weights are proportional to the inverse square root of their sample counts, clipped to $[0.5, 2]$ and normalized to unit mean. Adaptive weights are updated from the preceding epoch's task-group MAE relative to the mean group MAE, clipped to $[0.5, 3]$ and normalized to unit mean; updates require at least 1,024 samples per group. These weights apply to reconstruction, temporal-difference, and MTR losses, with normalization by the sum of scalar-entry weights. The codebook and commitment losses use their standard unweighted form.

Table 4: Stage 1 configuration and training hyperparameters. AE denotes continuous autoencoder initialization; tokenizer training denotes the subsequent Product-VQ training with MTR.

Hyperparameter	LIBERO	RoboCasa-GR1
Tokenizer configuration		
Action horizon H	8	16
Action dimension d_a	7	29
Temporal scales	$\{1, 2, 4, 8\}$	$\{1, 2, 4, 8, 16\}$
Latent dimension d_f	32	256
Product-VQ groups G	16	16
Codebook size V	512	512
Autoencoder initialization		
Training epochs	30	50
Batch size	256	512
Learning rate	1×10^{-4}	2×10^{-4}
Product-VQ tokenizer training with MTR		
Training epochs	100	100
Batch size	256	512
Learning rate	5×10^{-5}	4×10^{-5}
Initial encoder/decoder freeze (epochs)	3	5
Temporal-difference weight λ_{vel}	0.5	0.25
Target VQ weight λ_{vq}	0.08	0.05
VQ weight warm-up (epochs)	10	20
MTR weight λ_{mtr}	0.1	0.1
Commitment coefficient β	0.25	0.2
Optimization settings for both phases		
Optimizer	AdamW	AdamW
Weight decay	1×10^{-5}	1×10^{-5}
Gradient clipping norm	1.0	1.0

Table 5: Stage 2 policy training hyperparameters. The action tokenizer remains frozen throughout policy training.

Hyperparameter	LIBERO	RoboCasa-GR1
Optimizer	AdamW	AdamW
VLM learning rate	1×10^{-5}	1×10^{-5}
Learning rate of other modules	2×10^{-5}	1×10^{-4}
AdamW (β_1, β_2)	$(0.9, 0.95)$	$(0.9, 0.95)$
AdamW ϵ	1×10^{-8}	1×10^{-8}
Weight decay	1×10^{-8}	1×10^{-8}
Maximum training steps	50,000	100,000
Learning-rate warm-up steps	5,000	5,000
Learning-rate schedule	Cosine decay	Cosine decay
Minimum learning rate	1×10^{-6}	1×10^{-6}
Batch size per GPU	16	32
Number of GPUs	8	8
Gradient accumulation steps	1	2
Effective global batch size	128	512
Label smoothing ε	0.02	0.02
Gradient clipping norm	1.0	1.0

Stage 2: Policy Training. After freezing the tokenizer, we train the policy to predict the cached multi-scale action tokens using label-smoothed cross-entropy. We use separate learning rates for the pretrained VLM and the remaining trainable modules, with linear warm-up followed by cosine decay. Table 5 lists the policy training settings. The effective global batch size is the per-GPU batch size multiplied by the number of GPUs and the gradient accumulation factor.

B.3 Sources of Baseline Results

LIBERO. The suite-level results for OpenVLA-OFT, π_0, π_0+FAST, StarVLA-FAST, StarVLA-OFT, StarVLA-π, and StarVLA-GR00T are taken from Table 2 of the StarVLA report (Community, 2026). The results for $\pi_{0.5}$, GR00T-N1.6, and StarVLA-α follow Table 1 of the StarVLA-α report (Ye et al., 2026). VLA-JEPA results are taken from Table 1 of its original paper (Sun et al., 2026). We compute each LIBERO average as the arithmetic mean of the four reported suite-level success rates, rounded to one decimal place. In particular, this gives 94.2% for π_0 and 97.9% for StarVLA-α.

RoboCasa-GR1. The results for GR00T-N1.6, $\pi_{0.5}$, StarVLA-FAST, StarVLA-OFT, StarVLA-π, and StarVLA-GR00T are taken from Table 5 of the StarVLA report (Community, 2026). The StarVLA-α result is taken from the specialist setting in Table 1 of its report (Ye et al., 2026). Results for LangForce (Lian et al., 2026), VP-VLA (Wang et al., 2026b), DualCoT- VLA (Zhong et al., 2026), FoMoVLA (Li et al., 2026), Qwen-VLA- Instruct (Wang et al., 2026a), and ABot-M0 (Yang et al., 2026) are taken from their respective original papers or technical reports. We retain the reported average success rates for this benchmark, including those of the StarVLA variants.

The baseline results follow the training and evaluation settings of their respective sources. Model backbones, pretraining data, and training budgets may therefore differ across methods.

C Scale-wise Tokenizer Diagnostics

Diagnostic Setup. We conduct a supplementary controlled Stage-1 study on RoboCasa-GR1 to examine how intermediate supervision shapes scale-wise action representations. The OAT-style, MINT-style, and MTR variants share the tokenizer architecture, training data, initialization, and training budget, and differ in their intermediate supervision. The analysis covers 508 complete action windows from 127 training demonstrations, each containing 16 timesteps and 29 normalized action dimensions. We encode demonstration actions with the frozen tokenizers and decode cumulative token prefixes.

Intermediate Targets and Reconstructions. Figure 5 separates each method's target or target reference from its cumulative reconstruction. MTR constructs a scale-matched target by averaging the original trajectory to the corresponding temporal resolution and linearly interpolating it back to the full action horizon. OAT-style supervision uses the complete trajectory at every intermediate scale. MINT-style supervision compares the full raw DCT-II spectra of the reconstruction and ground truth, with $2/H$ normalization; its displayed time-domain reference is therefore the complete ground-truth trajectory.

All panels use the same action window and action dimension. The window is selected using ground truth alone: its mean temporal variance is closest to the 80th percentile among the sampled windows. We then display the action dimension with the largest temporal variance within that window. This selection does not depend on reconstruction quality.

The visualization illustrates that MTR follows progressively refined trajectory targets, whereas the OAT- and MINT-style variants already approximate the complete trajectory at early scales. This comparison characterizes differences in intermediate representations; proximity to MTR's scale-matched targets is an alignment diagnostic rather than a method-independent measure of reconstruction quality.

Frequency Content of Successive Corrections. To examine what each additional scale contributes, we analyze the difference between consecutive cumulative reconstructions:

$$\Delta\hat{\mathbf{A}}_s = \hat{\mathbf{A}}_s - \hat{\mathbf{A}}_{s/2}, \qquad s \in \{2, 4, 8, 16\}. \tag{13}$$

Here, $\hat{\mathbf{A}}_s$ denotes the reconstruction obtained using all token scales up to s. For each method, we apply an orthonormal DCT-II along the temporal dimension of these differences and average squared coefficients over sampled windows and action dimensions. Writing the resulting energy at frequency

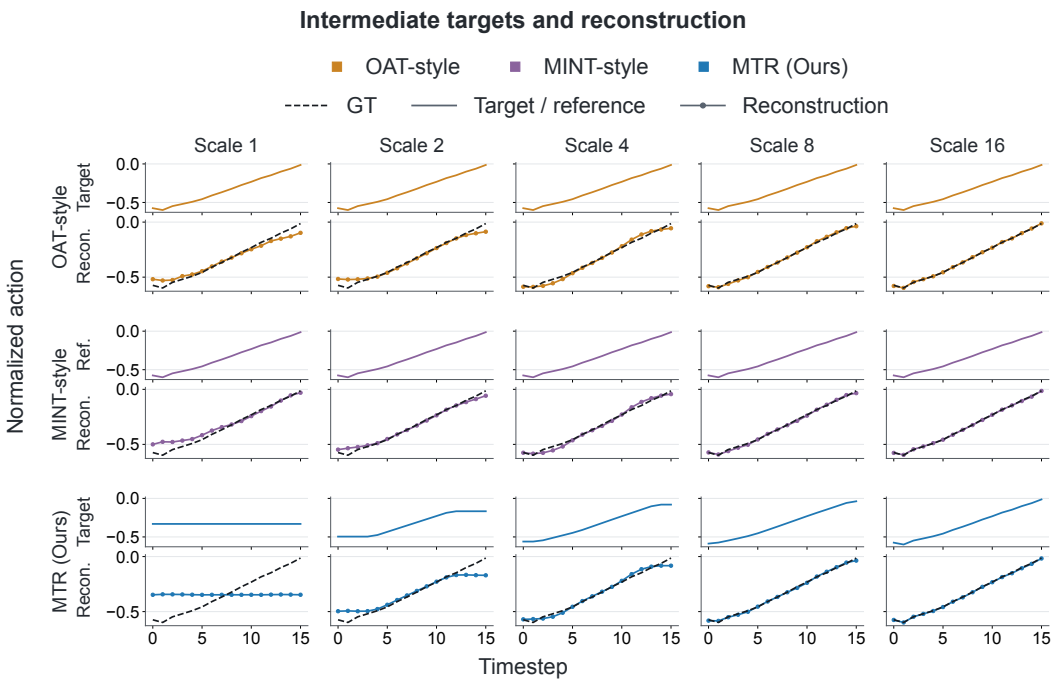


Figure 5: **Intermediate targets and cumulative reconstructions.** Each method occupies two rows: the target or its time-domain reference above, and the reconstruction with ground truth below. Colors identify methods; solid lines indicate targets or references, circular markers identify reconstructions, and black dashed lines denote ground truth. The MINT-style reference corresponds to its full-spectrum target and does not imply time-domain supervision. All panels share the same sample and axis limits.

index k as $E_{s,k}$, we visualize,

$$P_{s,k} = \frac{E_{s,k}}{\sum_{j=1}^{H-1} E_{s,j}}, \qquad k = 1, \ldots, H-1, \tag{14}$$

where $H = 16$. We exclude the DC component to focus on temporal variation. Each heatmap row therefore sums to one, and all methods share the same color scale.

As shown in Figure 6, the peak non-DC frequency in the pooled MTR spectra progresses through $1, 2, 4, 8$ as scales $2, 4, 8, 16$ are added. The corresponding peaks for both OAT-style and MINT-style supervision are $1, 1, 1, 8$. These aggregate patterns are consistent with a clearer coarse-to-fine allocation of temporal structure under MTR, although the frequency bands are not strictly separated. Because the spectra are normalized within each row, they describe relative frequency allocation rather than absolute correction magnitude.

D Real-World Experimental Details

D.1 Real-World Hardware and Task Details

Hardware Setup. We conduct real-world experiments using an xArm manipulator equipped with a parallel-jaw gripper, as shown in Figure 7(a). Two wrist-mounted RGB cameras provide complementary observations of the grasping region and the target USB socket; their mounting locations are highlighted in the figure. Both image streams are stored at a resolution of 160×160 pixels in the demonstration dataset. Actions are seven-dimensional, comprising three end-effector translation increments, three rotation increments, and one gripper command.

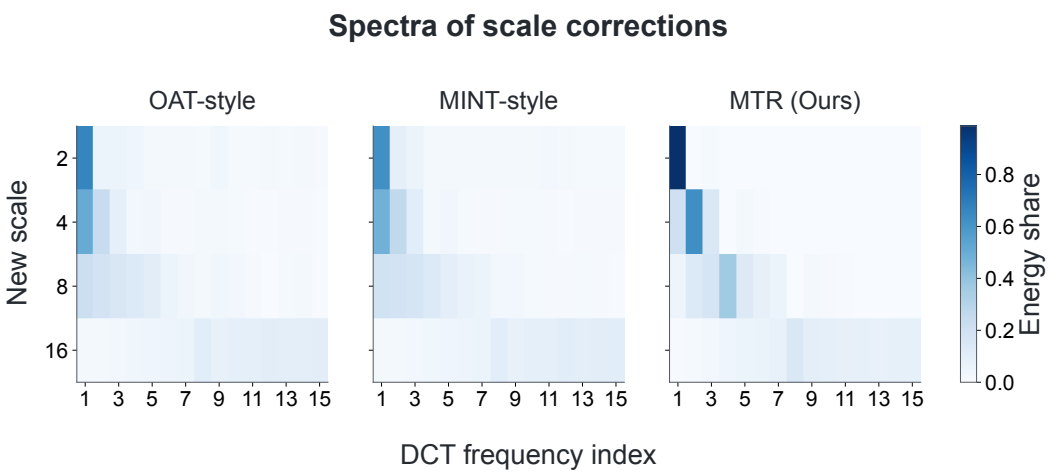


Figure 6: **Frequency content of successive reconstruction increments.** Rows correspond to newly added token scales, and columns to DCT frequency indices. Darker colors indicate a larger fraction of non-DC correction energy at a given frequency. Energy is aggregated over the same 508 action windows and all 29 action dimensions, then normalized independently within each row. Higher frequency indices represent more rapid temporal variation.

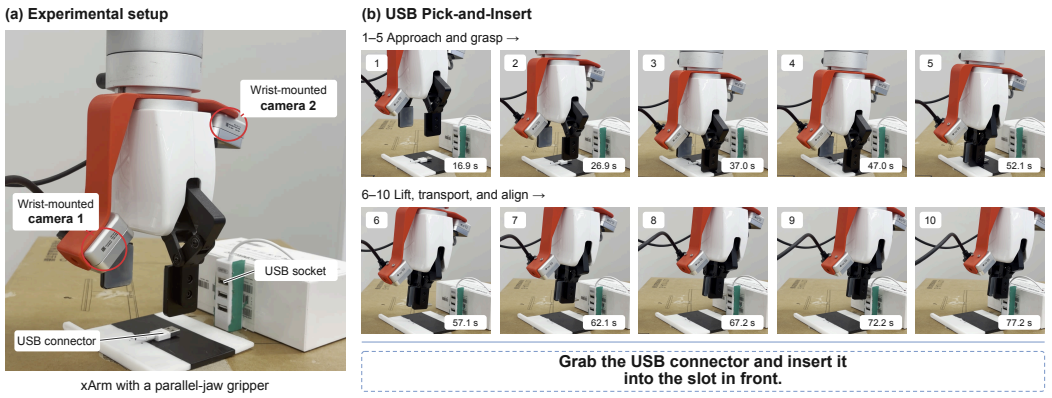


Figure 7: **Real-world experimental setting.** (a) An xArm manipulator equipped with a parallel-jaw gripper and two wrist-mounted RGB cameras. (b) Ten keyframes from a single USB Pick-and-Insert recording, ordered from left to right across the top row and then the bottom row, showing approach, grasping, lifting, transport, and alignment with the target USB socket.

USB Pick-and-Insert Task. The workspace contains a USB connector resting on a tabletop platform and a USB socket mounted on a fixed support, both identified in Figure 7(a). The language instruction is "Grab the USB connector and insert it into the slot in front." The robot must approach and grasp the connector, lift it from the platform, and transport it toward the socket before adjusting its position and orientation for insertion. Figure 7(b) illustrates this progression through ten keyframes from a single external-view recording: frames 1–5 show approach and grasping, while frames 6–10 show lifting, transport, and alignment with the socket. This task combines reaching and object transport with precise local alignment, since small positional or angular errors can prevent insertion. A trial is successful only if the robot grasps the connector and fully inserts it into the socket without human intervention.

Table 6: Proposed training hyperparameters for the real-world experiments. AE denotes continuous autoencoder initialization, and VQ+MTR denotes subsequent quantized tokenizer training.

Hyperparameter	Stage 1: AE	Stage 1: VQ+MTR	Stage 2: Policy
Training budget	30 epochs	100 epochs	10,000 steps
Global batch size	256	256	64
Optimizer	AdamW	AdamW	AdamW
VLM learning rate	—	—	1×10^{-5}
Other modules' learning rate	1×10^{-4}	5×10^{-5}	2×10^{-5}
AdamW (β_1, β_2)	$(0.9, 0.999)$	$(0.9, 0.999)$	$(0.9, 0.95)$
AdamW ϵ	1×10^{-8}	1×10^{-8}	1×10^{-8}
Weight decay	1×10^{-5}	1×10^{-5}	1×10^{-8}
Learning-rate warm-up	—	—	500 steps
Learning-rate schedule	Constant	Constant	Cosine decay
Minimum learning rate	—	—	1×10^{-6}
Gradient clipping norm	1.0	1.0	1.0
Label smoothing	—	—	0.02

D.2 Demonstration Collection and Training

Demonstration Collection. We collect 720 demonstrations of USB Pick-and-Insert through VR teleoperation. The operator controls the robot to approach and grasp the connector, transport it toward the socket, and complete the alignment and insertion. The dataset contains 116,298 recorded time steps at 10 Hz, corresponding to approximately 3.24 hours of demonstration data. Each demonstration includes RGB observations from both wrist-mounted cameras, robot states, and seven-dimensional actions. All 720 demonstrations are used for training.

Training Procedure. We follow the same two-stage training procedure as in the simulation experiments. In Stage 1, we initialize the action encoder and decoder through continuous autoencoder training, then train the residual Product- VQ tokenizer with MTR. In Stage 2, we freeze the tokenizer and precompute multi-scale token labels for the demonstrated action chunks. We then fine-tune the Qwen3-VL-4B backbone and the scale-autoregressive prediction head using label-smoothed cross-entropy with ground-truth token groups from preceding scales as context.

Hyperparameters. For the real-world configuration, we use an action horizon of $H = 8$ with temporal scales $\{1, 2, 4, 8\}$. Following the LIBERO tokenizer configuration, we set the latent dimension to 32 and use 16 Product-VQ groups with 512 entries per codebook. The loss coefficients are $\lambda_{\text{vel}} = 0.5$, $\lambda_{\text{vq}} = 0.08$, and $\lambda_{\text{mtr}} = 0.1$, with commitment coefficient $\beta = 0.25$.During quantized tokenizer training, the encoder and decoder are frozen for the first three epochs, and the VQ loss coefficient is linearly increased to its target value over the first ten epochs. Table 6 lists the proposed optimization settings; batch sizes denote effective global batch sizes.

D.3 Real-World Evaluation Protocol

In-Distribution Evaluation. ID trials use initial connector and socket configurations within the demonstration distribution. We evaluate each method over 30 trials, using the same initial layouts across methods. The robot must complete the entire USB Pick-and-Insert task, including grasping the connector, transporting it toward the socket, and aligning it for insertion.

Out-of-Distribution Evaluation. OOD trials vary the initial placement of the connector and socket beyond the demonstrated configurations. The connector is displaced laterally by up to $3\,\text{cm}$ to either side of its nominal position and rotated by up to 30° clockwise or counterclockwise. The socket position is randomized within a $3\,\text{cm} \times 3\,\text{cm}$ planar region. Each method is evaluated over 30 trials using the same initial layouts.

Success Criterion. We apply the same binary success criterion under both conditions. A trial is successful only if the robot grasps the connector and fully inserts it into the target socket within the

without human intervention. Successful transport, contact with the socket, or partial insertion alone does not count as success. Trials that require human assistance are counted as failures.

D.4 Detailed Results and Representative Rollouts

Representative Behaviors. Figure 8 shows a successful TrajScale-VLA rollout and a failed baseline rollout from two camera views. TrajScale- VLA grasps the connector, lifts it from the platform, and transports it toward the socket while retaining the grasp. It then aligns and inserts the connector, which remains in the socket after the gripper opens. The baseline also grasps the connector and brings it near the socket, but does not fully align it with the opening. During the insertion attempt, the misaligned connector is pushed out of the gripper upon contact with the socket and falls onto the platform. The gripper opens only after the connector has fallen. The failure therefore occurs at the transition from approach to precise insertion.

Relation to Trajectory Supervision. Our OAT-style and MINT-style comparisons use a shared tokenizer and policy backbone, with different supervision for intermediate reconstructions. OAT trains variable-length token prefixes to reconstruct the same full-resolution action chunk (Liu et al., 2026), while MINT supervises cumulative reconstructions against the full action spectrum (Huang et al., 2026). Both encourage progressive refinement, but neither assigns a distinct temporal-resolution target to each cumulative reconstruction. MTR instead supervises each intermediate reconstruction with a trajectory obtained by temporal pooling and interpolation at its corresponding scale. This gives coarse representations an explicit target for the overall motion within each action chunk, while progressively finer targets introduce more localized action variations.

This failure highlights the need to preserve fine alignment adjustments alongside the overall approach motion. Even when grasping and transport succeed, a small positional or angular mismatch can prevent entry into the socket and cause the connector to be dislodged during the insertion attempt. MTR explicitly supervises the overall motion at coarse temporal resolutions and progressively introduces more localized variations at finer scales. This organization may help the policy retain the small translational and rotational adjustments needed to align the connector before insertion. The successful alignment and insertion achieved by TrajScale-VLA in this rollout are consistent with this intended role of trajectory-aligned supervision.

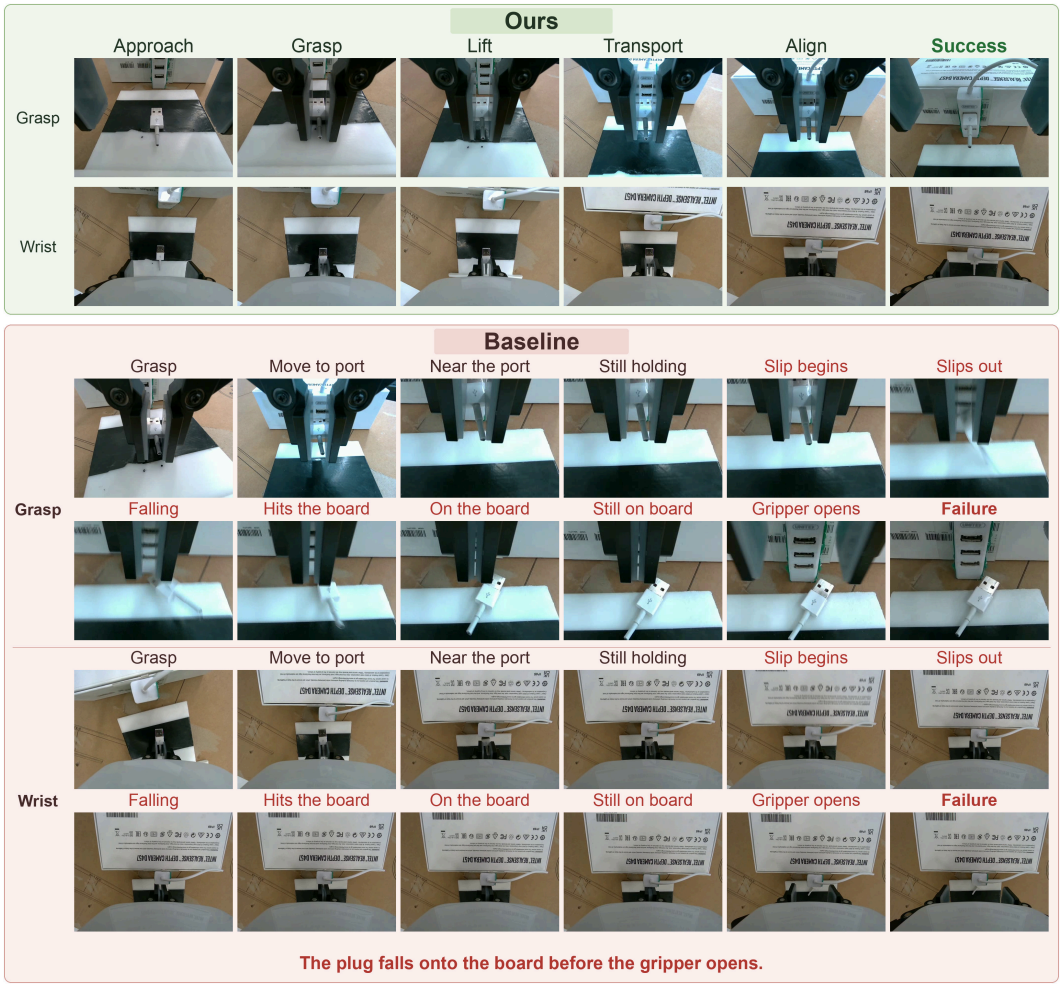


Figure 8: **Qualitative comparison of real-world USB insertion.** Gripper and wrist camera views show a successful rollout of our method and a failed rollout of the baseline. Our method completes the insertion, whereas the baseline loses its grasp on the USB plug, which falls onto the platform before the gripper opens. Additional keyframes highlight the baseline's failure process.