HOLOBRAIN-DISCRETE: SHARED ACTION TOKENIZATION FOR CROSS-EMBODIMENT ROBOT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning a single policy across robot embodiments requires an action representation that retains joint-level motion while accommodating different robot structures. Joint-graph trajectories, which record each joint's angle and link pose, expose this structure but create many correlated prediction targets. We present HoloBrain-Discrete, which couples a virtual-joint action tokenizer with a code/cue delayed autoregressive policy. Virtual joints serve as latent bottleneck nodes positioned on the robot graph, compressing trajectories into discrete codes with one encoder and decoder shared across embodiments. The policy generates these codes in groups along temporal, virtual-joint, and coarse-to-fine refinement axes, advancing one group per step rather than one code at a time. A separate prediction cue for each code gathers its context from earlier groups. Pretraining on approximately 10,000 hours spanning 18 embodiments yields higher success than diffusion-based HoloBrain at every measured checkpoint across five evaluation settings. Cross-embodiment demonstrations also boost few-shot learning: co-training two UR5 demonstrations per task with Aloha data raises success from 6% to 39%, versus 7% to 17% for HoloBrain. Model inference is 1.9–2.1× faster than HoloBrain on LIBERO and RoboTwin. Compact, structured action targets thus support efficient cross-embodiment pretraining and few-shot transfer.

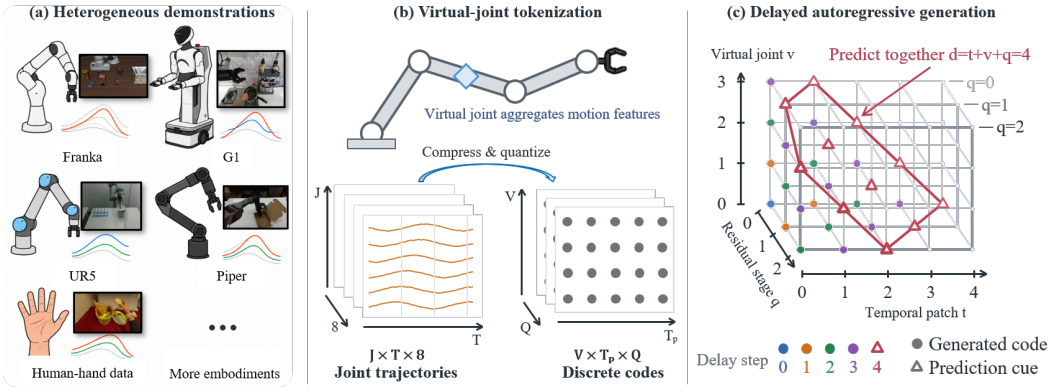


Figure 1: **HoloBrain-Discrete: shared action tokenization and delayed autoregressive generation.** (a) Demonstrations span robots and human hands. (b) Virtual joints compress $J \times T \times 8$ trajectory values into $V \times T_p \times Q$ codes: J/V count real/virtual nodes, T/T_p count frames/patches, and Q counts residual stages. Each real node stores a joint scalar and link pose. (c) Cells with equal $d = t + v + q$ are predicted together. Circles denote earlier codes, triangles mark current cues ($d = 4$), and colors distinguish steps; cues condition on permitted earlier codes and observations.

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.

1 INTRODUCTION

Large-scale pretraining on diverse robot demonstrations offers a promising path toward generalist robot policies (Open X-Embodiment Collaboration et al., 2024; Octo Model Team et al., 2024). Scaling this approach requires learning across different joint configurations, geometries, and control interfaces (Figure 1a). An action representation for shared pretraining must therefore accommodate heterogeneous robot structures while retaining the motion information needed for control.

Existing action interfaces differ in the motion information they retain. End-effector (EE) commands describe manipulation by the pose of a single point, a quantity shared across manipulators (Kim et al., 2025b), but leave joint motion to robot-specific inverse kinematics or controllers. Another approach places joint commands in a fixed-width vector, padding unused entries for robots with fewer joints (Pertsch et al., 2025; Fang et al., 2026b). This retains joint motion, but matching vector widths alone does not identify the geometry or connectivity associated with each coordinate.

Joint-centric graph representations make these relationships explicit (Lin et al., 2025; 2026). Each node carries a joint's angle and associated link pose over a short future trajectory, or *action chunk*, and edges describe joint connectivity. Together, these quantities associate motion with the robot's geometry and internal joint configuration, providing information absent from an endpoint command or an unannotated coordinate vector. This richer description also enlarges the prediction space: representing every joint scalar and link pose over time introduces many correlated quantities. We seek to accelerate learning from these trajectories during large-scale cross-embodiment pretraining through compression and discretization. Compression gives the policy fewer targets to predict, while discrete codes enable teacher-forced categorical prediction (Pertsch et al., 2025).

To this end, we introduce **HoloBrain-Discrete**, which combines a *virtual-joint tokenizer* with a *code/cue delayed autoregressive policy*. The tokenizer converts joint trajectories into compact prediction targets conditioned on robot structure (Figure 1b). Its information bottleneck consists of virtual joints: latent nodes positioned on the robot graph. Residual quantization produces codes organized by virtual joint, time, and coarse-to-fine refinement. Conditioned on the robot's graph and current state, the decoder reconstructs joint and link-pose trajectories from these codes.

The policy generates groups of codes in parallel while conditioning on earlier groups, reducing sequential generation depth relative to one-code-at-a-time prediction. These groups form a diagonal wavefront through the three-dimensional code array, advancing across temporal patches, virtual joints, and residual refinement stages (Figure 1c). Each wavefront is a *delay step*: codes on it are predicted together, and autoregression proceeds between steps. A separate prediction position, or *cue*, identifies each target code and gathers its permitted context. Once predicted, the code values become available as context for subsequent delay steps.

Our contributions are:

- A graph-based action tokenizer that compresses joint trajectories into discrete codes through virtual joints, using one encoder and decoder across robots with different joint counts.
- A code/cue policy that predicts groups of graph-indexed action codes through three-axis delayed generation, conditioning each group on earlier predictions and cutting sequential generation depth from one code per step to one group per step.
- A pretraining study spanning approximately 10,000 hours and 18 embodiments, in which HoloBrain-Discrete attains higher success than diffusion-based HoloBrain at every measured checkpoint and generates action chunks 1.9–2.1$\times$ faster, with comparisons of action representations, compression levels, and generation schedules.

2 RELATED WORK

Sharing policies across embodiments. Learning from pooled robot data requires a way to accommodate heterogeneous observations and controls (Open X-Embodiment Collaboration et al., 2024; Octo Model Team et al., 2024). One family separates shared processing from interface-dependent parameters: embodiment/task-specific stems and heads in HPT, embodiment-specific decoders in UniAct, control-type heads in CrossFormer, and learned domain prompts in X-VLA (Wang et al.,

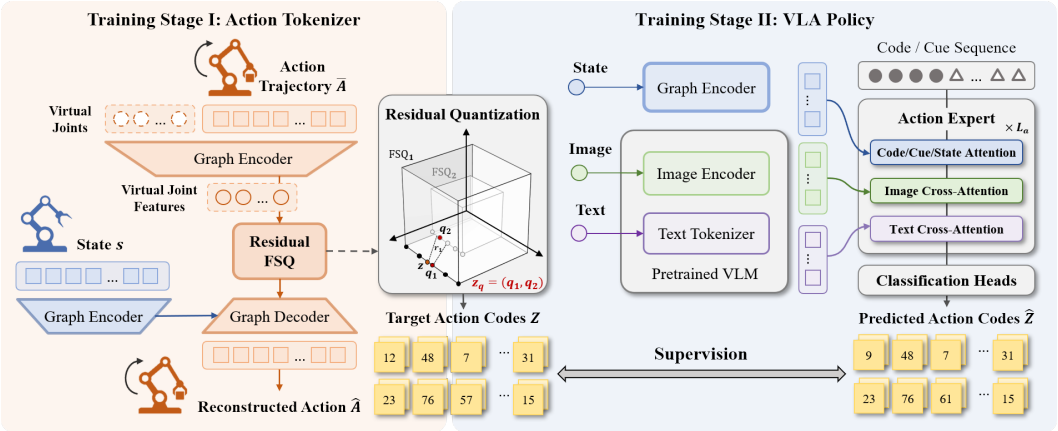


Figure 2: **HoloBrain-Discrete two-stage training. Stage I:** graph attention and residual FSQ compress trajectories through virtual joints; a state-conditioned decoder reconstructs them. **Stage II:** the frozen tokenizer supplies targets for an image-, language-, and state-conditioned policy. Codes carry values; cues feed residual-stage heads shared across virtual joints. The inset shows two of the model's three quantization stages; normalization and restoration are omitted.

2024; Zheng et al., 2025a; Doshi et al., 2025; Zheng et al., 2025b). A complementary choice is to expose robot structure to the model through limb descriptions or graph connectivity (Gupta et al., 2022; Patel & Song, 2025). SEM and HoloBrain associate joint motion with link geometry and graph relationships; we inherit their real-joint representation and graph attention (Lin et al., 2025; 2026). Structure can also be supplied through a standardized interface, as in UniDexTok's shared hand-state representation (Fang et al., 2026a). Our focus is compressing trajectories on a variable-size joint graph into discrete targets for policy learning.

Constructing discrete action targets. Action tokenizers differ in both what they compress and how their codes are trained. Signal-based compression uses a frequency transform and byte-pair encoding in FAST, while learned quantization maps action chunks into hierarchical latent codes in VQ-BeT (Pertsch et al., 2025; Lee et al., 2024). Structured bottlenecks organize compression around latent registers with prefix reconstruction in OAT, or around coordinate subspaces and motion parts in OmniSAT and G0.5 (Liu et al., 2026a; Lyu et al., 2025; Liu et al., 2026b). Beyond reconstruction fidelity, policy-oriented tokenizer design considers the predictability of codes and their alignment with visual-language context, as studied in ActionCodec and X-Tokenizer (Dong et al., 2026; Kang et al., 2026). Conditioning is another design choice: SA-VLA incorporates robot state through cross-attention or action modulation (Jiang et al., 2026). Our graph-positioned bottleneck adapts to each embodiment's structure, with residual quantization of virtual-joint features and graph/state-conditioned reconstruction.

Parallel and autoregressive action generation. Generation schedules trade sequential passes against conditioning on earlier outputs. OpenVLA-OFT studies single-pass action-chunk prediction from shared context (Kim et al., 2025a). FASTer's blockwise action decoding and MusicGen's delayed audio-code interleaving condition on earlier groups (Liu et al., 2025; Copet et al., 2023). Our policy schedules prediction over the graph-indexed code array: diagonal delay steps advance across time, virtual joints, and refinement, and each code keeps its own prediction position, a cue, under temporal and residual visibility masks.

3 METHOD

HoloBrain-Discrete couples a graph-conditioned trajectory tokenizer with a policy conditioned on images, language, and robot state (Figure 2). The tokenizer learns trajectory reconstruction, then supplies frozen targets for policy training. At inference, the policy generates code groups and the decoder reconstructs an action chunk for execution.

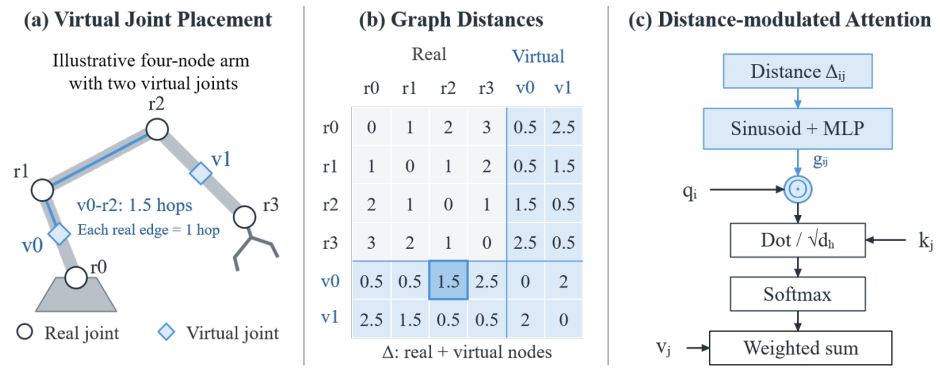


Figure 3: **Virtual joints and distance-modulated attention.** **(a)** A four-node arm with two midpoint virtual joints; real edges have unit hop length, and the highlighted path spans 1.5 hops. **(b)** Augmented distances Δ, with row v_0 highlighted. **(c)** HoloBrain's distance embedding and MLP gate query–key interactions before softmax and value aggregation (Eq. (3); temporal indices and masks omitted). Virtual joints share an initial vector but have distinct graph distances. Appendix Figure 9 gives the network.

3.1 Joint-Graph Action Representation

We use the joint-centric representation of SEM and HoloBrain (Lin et al., 2025; 2026). An action chunk is $A \in \mathbb{R}^{J\times T\times 8}$, describing the next T frames. Its J nodes represent joints or grippers, each storing a joint angle or scalar gripper opening together with the associated link's 3D position and unit quaternion. Link poses are expressed in the head-camera or third-person-camera coordinate frame. The joint graph G supplies a distance matrix Δ^{real}, whose entries count shortest-path hops between real nodes. EE-only data use a single pose-and-opening node. The current state $s \in \mathbb{R}^{J\times 1\times 8}$ is frame zero, and A contains frames 1 through T.

We divide an action chunk into T_p non-overlapping *temporal patches*, each spanning a fixed number of consecutive frames. The tokenizer maps the normalized absolute trajectory $\overline{A}$ to codes $Z \in \{0,\dots,K-1\}^{V\times T_p\times Q}$, using V virtual joints and Q residual stages with K possible code values per stage. Codes are indexed by virtual joint v, temporal patch t, and residual stage q, with the policy using the tokenizer's virtual-joint order. Writing $\mathcal{E}_\phi$ and $\mathcal{D}_\psi$ for the learned encoder and decoder, respectively, gives

$$Z = \mathcal{E}_\phi(\overline{A}; G), \qquad \widehat{R} = \mathcal{D}_\psi(Z; s, G). \tag{1}$$

The decoder conditions on s to predict relative joint and pose changes $\widehat{R}$, which are converted to absolute trajectories $\widehat{A}$ using the current state for either joint or EE control. Appendix A.1 specifies normalization and the gripper-opening convention.

3.2 Virtual-Joint Action Tokenization

Virtual joints. To summarize trajectories using their graph structure, we introduce latent aggregation nodes, termed *virtual joints* (Figure 3). Assigning them positions on the graph distinguishes their relationships to the real joints through distance-based positional encoding (Figure 3a,b). For a motion chain c with n_c declared link and finger entries, we allocate

$$V_c = \lfloor n_c/\rho \rfloor + 1, \qquad V = \sum_c V_c, \tag{2}$$

where ρ controls compression. All tokenizer parameters are shared across these embodiment-dependent counts. We seek an even distribution by minimizing the maximum distance from a real joint to its nearest virtual joint, with farthest-point initialization for large candidate sets followed by placement refinement. Candidates include vertices and fractional edge positions; Appendix A.1 gives the search procedure. Augmenting the real-node matrix Δ^{real} with distances involving virtual joints gives $\Delta \in \mathbb{R}^{(J+V)\times(J+V)}$, used for graph attention below.

Graph aggregation and discretization. Each temporal patch of a real-node trajectory is embedded into a d-dimensional feature. We append V virtual nodes initialized from a shared learned vector and process the $J+V$ node streams with graph attention. Following HoloBrain, a sinusoidal embedding and MLP map each distance Δ_{ij} to a channel-wise gate $g(\Delta_{ij})$ (Figure 3c). For one attention head, this gate modulates the query–key interaction:

$$\ell_{ij}^{tt'} = \frac{\langle \mathbf{q}_{i,t} \odot g(\Delta_{ij}), \mathbf{k}_{j,t'} \rangle}{\sqrt{d_h}}, \qquad t' \le t. \tag{3}$$

Here $\mathbf{q}$ and $\mathbf{k}$ are attention query and key vectors, and d_h is the head width. Temporal rotary encoding is applied to queries and keys; softmax over the permitted node–time keys gives the attention weights. We retain the virtual-node features $H \in \mathbb{R}^{V \times T_p \times d}$ as the compression bottleneck. Residual finite scalar quantization (FSQ) (Mentzer et al., 2024) discretizes each $H_{v,t}$: the first stage encodes a coarse approximation, and each subsequent stage quantizes the remaining residual. Each virtual joint and temporal patch thus yields Q codes, $Z_{v,t,q}$ for $q = 0, \dots, Q-1$, one per refinement stage.

Reconstruction. Real-node queries initialized from the current state attend to the quantized virtual features to predict the relative trajectory. Training every residual prefix encourages earlier stages to support coarse reconstruction and later stages to refine it. Let $\widehat{A}^{(r)}$ be the restored trajectory using the first r stages. We minimize

$$\mathcal{L}_{\text{tok}} = \sum_{r=1}^{Q} \lambda_r \, \mathcal{L}_{\text{rec}}(\widehat{A}^{(r)}, A). \tag{4}$$

Here λ_r weights the reconstruction loss of the r-stage prefix. The reconstruction loss compares joint scalars, positions, and rotations, with forward-kinematics supervision and pose-consistency terms where kinematics are available. Quantization details, loss definitions, and implementation settings are in Appendices A.2–A.5.

3.3 Code/Cue Delayed Autoregressive Policy

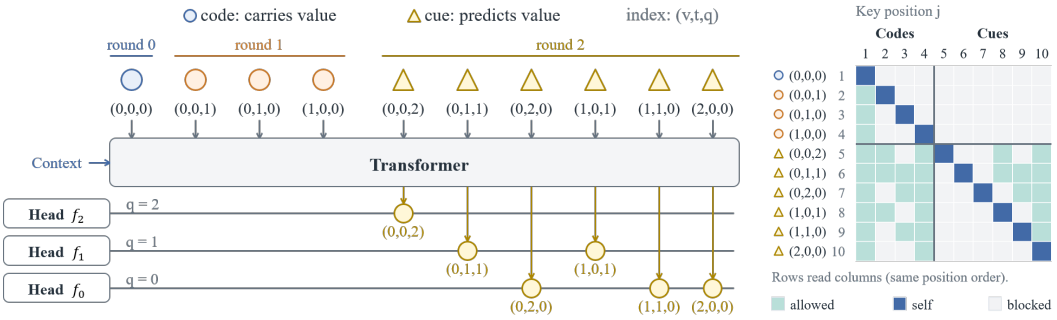
The tokenizer represents each action chunk with codes for multiple virtual joints, times, and refinement stages. Generating every code serially requires VT_pQ sequential passes, stretching the interval between successive control updates. Predicting all codes in a single pass removes this sequential cost but cannot condition on code values sampled elsewhere in the same chunk. We instead predict groups of codes together, using permitted earlier predictions as context (Figure 4). A separate prediction position for each target retains its temporal, refinement-stage, and graph context within this grouped schedule.

Code and cue positions. Each cell $i = (v, t, q)$ occupies two positions in the transformer sequence: a *code position* carrying its value Z_i, and a *cue position* from which that value is predicted. Their initial features are $\mathbf{h}_i^{\text{code},0} = e(Z_i) + e_q^{\text{code}}$ and $\mathbf{h}_i^{\text{cue},0} = e_q^{\text{cue}}$. Here e is a learned embedding lookup for code values, and e_q^{code} and e_q^{cue} are learned embeddings for residual stage q. Temporal rotary encoding identifies the patch, and graph distances identify the virtual joint's relationships to other nodes. The transformer conditions on visual features, language features, an encoding of the current robot state, and graph G, collectively $\mathcal{C}$. The output feature at each cue gives $p_{\theta,i} = \text{softmax}(f_q(\mathbf{h}_i^{\text{cue}}))$, where f_q is the classification head for residual stage q.

Three-axis delayed generation. We advance diagonally through the code array: predictions whose virtual-joint, temporal-patch, and residual-stage indices have the same sum are made together. Each diagonal group defines a *delay step*. Using zero-based indices, we assign each cell a delay step d_i and collect the cells predicted at step d in $\mathcal{G}_d$:

$$d_i = t + v + q, \quad \mathcal{G}_d = \{i : d_i = d\}, \quad B(i,j) = \mathbf{1}[t' \le t]\mathbf{1}[q' \le q], \tag{5}$$

Here $j = (v', t', q')$ is a key cell, and $B(i,j)$ is the temporal/residual visibility mask; $\mathbf{1}[\cdot]$ denotes an indicator. Holding the other coordinates fixed, an earlier time or a coarser stage falls in an earlier group and is available as context. The additional mask B prevents conditioning on a later time or finer residual stage, even if its delay is smaller.



(a) Codes, cues, and shared residual-stage heads.

(b) Attention visibility mask.

Figure 4: **Code/cue delayed generation.** (a) Four known codes (circles, steps 0–1) and six prediction cues (triangles, step 2) for $d = t + v + q$. Each cue uses its residual-stage head f_q, shared across joints and patches. (b) Visibility among these positions: rows query columns; teal allows attention, blue marks self-attention, and gray blocks it. Positions 1–4 carry codes; 5–10 are cues whose target values remain unavailable. Appendix Figure 8 decomposes the mask.

A cue can read earlier code values satisfying $d_j < d_i$ and $B(i, j) = 1$. Within its own group, it can interact with other cues under B, but cannot access their unknown code values. Code positions read their own values and permitted earlier codes. Figure 4(b) shows the resulting visibility for these positions; Appendix Figure 8 separates the delay, time, and residual conditions. Let $\mathcal{P}(i) = \{j : d_j < d_i,\ B(i, j) = 1\}$ denote the permitted earlier code positions. The policy factorizes as

$$p_\theta(Z \mid \mathcal{C}) = \prod_{d} \prod_{i \in \mathcal{G}_d} p_\theta(Z_i \mid Z_{\mathcal{P}(i)}, \mathcal{C}). \tag{6}$$

Predictions within a group factorize conditional on earlier codes and observations. The schedule generates VT_pQ codes in $T_p + Q + V - 2$ rounds. Appendix A.4 gives a complete example.

Policy training and inference. The frozen tokenizer supplies demonstration codes for teacher-forced policy training. All code and cue positions are processed in one pass under the generation mask, with a cross-entropy objective and auxiliary visual depth supervision inherited from HoloBrain (Lin et al., 2026). Appendix A.3 specifies the full training objective and regularization. At inference, codes sampled in one group become context for later groups; attention keys and values are cached. The completed code array is decoded into one action chunk.

4 Experiments

4.1 Experimental Setup

Pretraining and benchmark evaluation. We train the tokenizer on approximately 10,000 hours of real-robot, simulated, and human-hand trajectories spanning 18 embodiments, then freeze it for policy pretraining. We compare checkpoints with diffusion-based HoloBrain (Lin et al., 2026) on LIBERO (Liu et al., 2023) and RoboTwin (Mu et al., 2025), then post-train separate benchmark policies, including seven real-world tasks scored by milestone progress. Inference latency is measured on an RTX 5090.

Learning from limited demonstrations. We compare UR5-only training with Aloha co-training, reusing the full-corpus tokenizer and initializing each policy from the same vision-language model. The tokenizer has seen UR5 motion, so this measures policy data efficiency given a pretrained representation.

Mixed-embodiment pool studies. Design comparisons train tokenizers and policies from scratch on a pool sampling 50% RoboTwin, 25% real-world, and 25% LIBERO trajectories. It includes dual-arm Aloha and UR5 (both 14 joint commands with different physical meanings), EE-controlled Franka, and Piper/PiperX. We vary compression, compare our representation with adaptations of

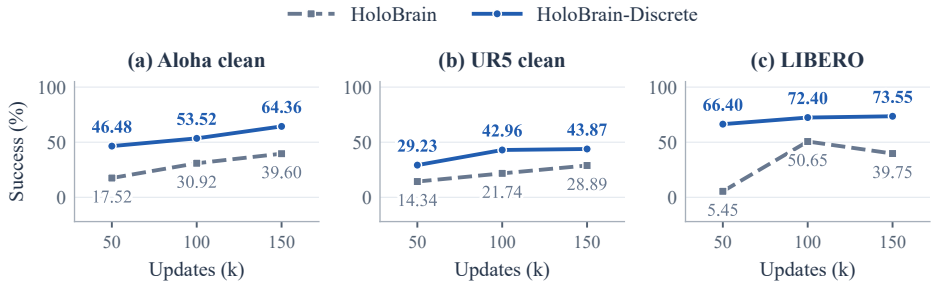


Figure 5: **Pretraining progress.** Closed-loop success at three training checkpoints. HoloBrain-Discrete leads in all five settings (randomized Aloha and LIBERO-Plus in Appendix B.2). RoboTwin uses 50 Aloha and 48 UR5 tasks, excluding two hanging UR5 environments (Appendix B.1).

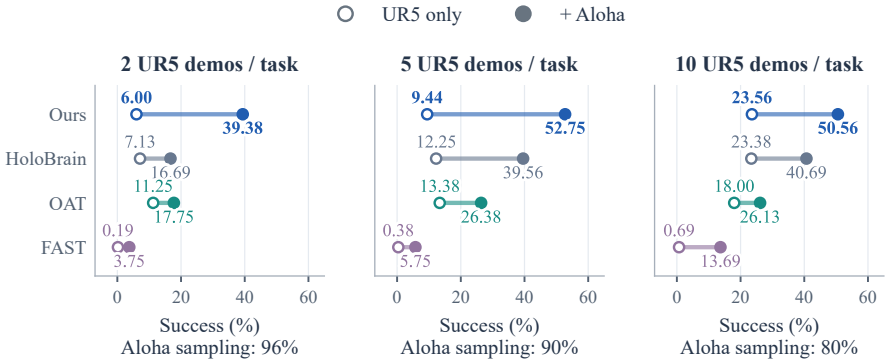


Figure 6: **Benefit from Aloha demonstrations in UR5 policy learning.** Open/filled markers show UR5-only/Aloha+UR5 training on 16 UR5 tasks, paired by method and target budget; labels give success rates (%). Aloha supplies 200 demonstrations per task; sampling fractions appear below each panel.

OAT (Liu et al., 2026a) and FAST (Pertsch et al., 2025), and ablate generation schedules. Appendix B.1 details task sets, trials, demonstration budgets, and training updates.

4.2 Faster Learning during Cross-Embodiment Pretraining

HoloBrain-Discrete learns effective policies earlier in pretraining (Figure 5). At 150k updates, it exceeds HoloBrain by 24.76 percentage points on Aloha clean, 14.98 on UR5, and 33.80 on LIBERO. The advantage is present at all three checkpoints, including under randomized Aloha observations and LIBERO-Plus perturbations.

4.3 Learning from Cross-Embodiment Demonstrations

Adding Aloha data at two UR5 demonstrations per task raises our success from 6.00% to 39.38%, compared with 7.13% to 16.69% for HoloBrain (Figure 6). At five and ten demonstrations, our co-trained policies reach 52.75% and 50.56%, improving over UR5-only training by 43.31 and 27.00 points. All four methods benefit from Aloha data, but our model achieves the highest co-training score at each target budget.

We also evaluate adaptation to PiperX with ten demonstrations per task and a 10% target sampling fraction. Progress scores range from 41.1 on towel folding to 91.1 on bowl stacking (Appendix B.3). This demonstrates deployment with limited target data.

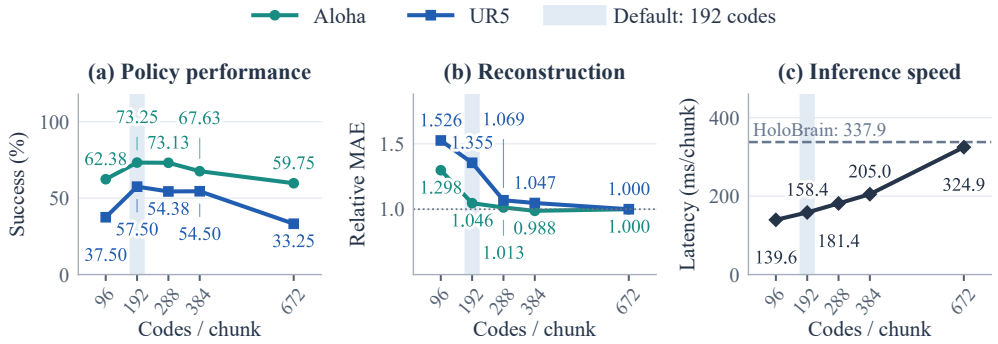


Figure 7: **Compression balances reconstruction, policy performance, and inference cost.** RoboTwin codes per 64-frame chunk correspond to 2/4/6/8/14 joint-axis positions; shading marks the default 192 codes. MAE is normalized per dataset to the 672-code value. RTX 5090 model latency excludes loading; the dashed line gives HoloBrain's 337.9 ms/chunk under the same conditions.

Table 1: **Action representation comparisons on the mixed-embodiment pool.** Success (%) on clean Aloha/UR5, LIBERO, and LIBERO-Plus. OAT compresses per patch with 400k tokenizer updates (extended from 200k); all policies use 200k updates. Appendix B.4 gives other OAT variants. *FAST counts are dataset averages; code counts, vocabularies, and budgets differ. Bold/underline mark best/second-best success.

Representation	Codes/chunk RoboTwin / LIBERO	Vocabulary (classes/code)	Aloha	UR5	LIBERO	Plus
OAT	64 / 64	1000	59.00	33.25	72.80	56.87
FAST	63.2 / 113.9*	2050	<u>73.13</u>	29.88	59.30	14.93
Ours (96 codes)	96 / 48	1000	62.38	<u>37.50</u>	**94.15**	<u>63.91</u>
Ours (192 codes)	192 / 48	1000	**73.25**	**57.50**	<u>93.60</u>	**66.62**

4.4 Ablations on the Mixed-Embodiment Pool

All variants train tokenizers and policies from scratch on the mixed-embodiment pool (Section 4.1), comparing how each representation and schedule uses heterogeneous demonstrations.

Virtual-joint compression. We first map how far the virtual-joint bottleneck can tighten (Figure 7). Reducing the joint-axis count from 14 to 4 cuts the number of codes from 672 to 192, while increasing Aloha success from 59.75% to 73.25% and UR5 success from 33.25% to 57.50%. The same change increases tokenizer MAE on both datasets but reduces model latency from 324.9 to 158.4 ms/chunk. Compressing further to 96 codes, one virtual joint per arm, reduces Aloha/UR5 success to 62.38%/37.50%. The best reconstruction is therefore not the most useful policy representation in this sweep: a moderate bottleneck gives a better balance of reconstruction, prediction, and cost. LIBERO uses a single joint-axis position throughout; its full results are reported separately in Appendix B.5.

Action representations. Our default representation achieves 73.25% on Aloha, 57.50% on UR5, 93.60% on LIBERO, and 66.62% on LIBERO-Plus (Table 1). FAST is nearly tied on Aloha at 73.13%, while trailing by 27.63, 34.30, and 51.69 points on the other three evaluations. OAT trails our default on all four benchmarks, by 9.75 to 24.25 points. The sweep's 96-code endpoint, whose RoboTwin code count is closest to the baselines', stays above OAT on all four evaluations and above FAST on UR5, LIBERO, and LIBERO-Plus, while FAST retains the higher Aloha score. These complete systems differ in code counts, vocabularies, budgets, and topology; FAST's lower reconstruction error on several datasets does not predict better closed-loop success (Appendix B.4).

Table 2: **Generation schedule ablation with a shared tokenizer.** Mixed-pool success (%) and RTX 5090 model latency (± SD). Parallel Q/V removes residual-stage/virtual-joint delay. Shading marks the default; bold/underline mark best/second-best success.

	Aloha			LIBERO		
Schedule	Succ. ↑	Rounds ↓	ms/chunk ↓	Succ. ↑	Rounds ↓	ms/chunk ↓
Full AR	67.13	192	553.3 ± 1.9	92.60	48	186.6 ± 1.2
Full delay	<u>73.25</u>	21	158.4 ± 1.4	**93.60**	18	126.0 ± 1.8
Parallel Q	70.00	19	153.8 ± 2.5	85.45	16	121.2 ± 1.5
Parallel V	**74.00**	18	158.3 ± 2.0	<u>93.25</u>	18	129.1 ± 1.8

Table 3: **Post-training performance.** C/R: clean/randomized Aloha; Plus: LIBERO-Plus. Real-world values average seven task progress scores, distinct from binary success (protocols and scores in Appendices B.1 and B.3).

	Simulation success (%) ↑				Real progress (%) ↑
Method	Aloha C	Aloha R	LIBERO	Plus	7-task mean
HoloBrain	**91.90**	**92.30**	**97.40**	72.60	**85.30**
Ours	90.60	90.74	96.35	**76.33**	74.29

Generation schedules. Full delay reduces RoboTwin generation from 192 serial rounds to 21 and improves Aloha success from 67.13% to 73.25% (Table 2). Measured latency falls from 553.3 to 158.4 ms/chunk, a 3.49× speedup; on LIBERO, the speedup is 1.48×, with success increasing from 92.60% to 93.60%. Parallelizing residual stages saves only 4.6 ms on RoboTwin and 4.8 ms on LIBERO, while reducing success by 3.25 and 8.15 points. In contrast, parallelizing virtual joints yields comparable latency and success, including a slightly higher Aloha score. The results favor retaining residual-stage dependencies.

A separate study with a frozen pretrained tokenizer and 200 Aloha demonstrations per task obtains 88.1% with code/cue positions versus 77.0% with MusicGen-style fused delay (Copet et al., 2023). Appendix B.6 details this protocol and complementary I/O and augmentation studies. Together, these results support efficient grouped generation with individually addressable predictions.

4.5 Benchmark Post-Training and Deployment

After benchmark-specific post-training, the comparison with HoloBrain is more mixed (Table 3). Our model reaches 90.60%/90.74% on clean/randomized Aloha and 96.35% on LIBERO, trailing HoloBrain by 1.30/1.56 and 1.05 points, respectively. LIBERO-Plus improves from 72.60% to 76.33%, with gains concentrated in robot initial states and camera viewpoints rather than language perturbations (Appendix B.2). Model inference is 2.13× faster than HoloBrain on RoboTwin (Figure 7c) and 1.90× faster on LIBERO (Appendix B.1).

The seven-task real-world mean is 74.29 progress points versus HoloBrain's 85.30, with the largest gaps in towel folding, bottle disposal, and block stacking. We hypothesize that shared compression sacrifices expressiveness needed for fine, embodiment-specific manipulation, consistent with the success decline under stronger compression (Figure 7). Recovering this precision while retaining shared pretraining remains an open direction.

5 Conclusion

HoloBrain-Discrete combines virtual-joint compression and code/cue delayed generation for shared policy learning from joint-level trajectories. It achieves stronger measured pretraining performance and faster model inference than HoloBrain; the compression sweep shows that reconstruction fidelity alone does not determine policy quality. Post-training gains remain uneven, with lower real-world manipulation scores. Future work should improve fine manipulation while retaining shared compression and evaluate generalization to embodiments absent from tokenizer training.

AI USE DISCLOSURE

We used generative AI tools for brainstorming and refining the conceptual framing, implementing code, providing feedback on methodology and experimental comparisons, and analyzing and interpreting experimental results. We also used these tools to identify and summarize relevant literature, draft and revise text, and create or edit LaTeX, figure-generation code, and scientific diagrams. Numerical tables and plots were prepared from experimental results, with AI assistance for formatting and consistency checks. The authors take responsibility for the final content, including AI-assisted text, claims, code, and figures.

REFERENCES

Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. *arXiv preprint arXiv:2306.05284*, 2023. URL https://arxiv.org/abs/2306.05284v3.

Zibin Dong, Yicheng Liu, Shiduo Zhang, Baijun Ye, Yifu Yuan, Fei Ni, Jingjing Gong, Xipeng Qiu, Hang Zhao, Yinchuan Li, and Jianye Hao. ActionCodec: What makes for good action tokenizers. *arXiv preprint arXiv:2602.15397*, 2026. URL https://arxiv.org/abs/2602.15397.

Ria Doshi, Homer Rich Walke, Oier Mees, Sudeep Dasari, and Sergey Levine. Scaling Cross-Embodied Learning: One Policy for Manipulation, Navigation, Locomotion and Aviation. In *Proceedings of The 8th Conference on Robot Learning*, pp. 496–512. PMLR, January 2025. URL https://arxiv.org/html/2408.11812v1.

Dong Fang, Youjun Wu, Yuanxin Zhong, Rui Zhang, Yunlong Wang, Xiaosong Jia, and Yu-Gang Jiang. UniDexTok: A unified dexterous hand tokenizer from real data. *arXiv preprint arXiv:2606.10683*, 2026a. URL https://arxiv.org/abs/2606.10683v2.

Haoquan Fang, Jiafei Duan, Donovan Clay, Sam Wang, Shuo Liu, Weikai Huang, Xiang Fan, Wei-Chuan Tsai, Shirui Chen, Yi Ru Wang, Shanli Xing, Jaemin Cho, Jae Sung Park, Ainaz Eftekhar, Peter Sushko, Karen Farley, Angad Wadhwa, Cole Harrison, Winson Han, Ying-Chun Lee, Eli VanderBilt, Rose Hendrix, Suveen Ellawela, Lucas Ngoo, Joyce Chai, Zhongzheng Ren, Ali Farhadi, Dieter Fox, and Ranjay Krishna. MolmoAct2: Action Reasoning Models for Real-world Deployment, May 2026b. URL https://arxiv.org/html/2605.02881v1.

Agrim Gupta, Linxi Fan, Surya Ganguli, and Li Fei-Fei. MetaMorph: Learning Universal Controllers with Transformers. In *International Conference on Learning Representations*, 2022. URL https://arxiv.org/html/2203.11931v1.

Tengyue Jiang, Chunpu Xu, Jiayue Kang, and Yao Mu. SA-VLA: State-aware tokenizer for improving vision-language-action models' performance. *arXiv preprint arXiv:2606.30113*, 2026. URL https://arxiv.org/abs/2606.30113v1.

Miracle Kang, Lights Shi, Lucy Liang, Roy Gan, Dongxiu Liu, Pushi Zhang, Sylas Chen, Shawn Qin, Yinan Zheng, Jinliang Zheng, Hao Wang, Xianyuan Zhan, and Hang Su. X-Tokenizer: A multimodal action tokenizer for vision-language-action pretraining. *arXiv preprint arXiv:2606.14752*, 2026. URL https://arxiv.org/abs/2606.14752v2.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025a. URL https://arxiv.org/abs/2502.19645v2.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of The 8th Conference on Robot Learning*, pp. 2679–2713. PMLR, January 2025b. URL https://arxiv.org/html/2406.09246v1.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. *Proceedings of Machine Learning Research*, 235:26991–27008, 2024. URL https://arxiv.org/abs/2403.03181v2.

Xuewu Lin, Tianwei Lin, Lichao Huang, Hongyu Xie, Yiwei Jin, Keyu Li, and Zhizhong Su. SEM: Enhancing spatial understanding for robust robot manipulation. *arXiv preprint arXiv:2505.16196*, 2025. URL https://arxiv.org/abs/2505.16196v1.

Xuewu Lin, Tianwei Lin, Yun Du, Hongyu Xie, Yiwei Jin, Jiawei Li, Shijie Wu, Qingze Wang, Mengdi Li, Mengao Zhao, Ziang Li, Chaodong Huang, Hongzhe Bi, Lichao Huang, and Zhizhong Su. HoloBrain-0 technical report. *arXiv preprint arXiv:2602.12062*, 2026. URL https://arxiv.org/abs/2602.12062v1.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023. URL https://arxiv.org/abs/2306.03310.

Chaoqi Liu, Xiaoshen Han, Jiawei Gao, Yue Zhao, Haonan Chen, and Yilun Du. OAT: Ordered action tokenization. *arXiv preprint arXiv:2602.04215*, 2026a. URL https://arxiv.org/abs/2602.04215v1.

Yicheng Liu, Shiduo Zhang, Zibin Dong, Baijun Ye, Tianyuan Yuan, Xiaopeng Yu, Linqi Yin, Chenhao Lu, Junhao Shi, Luca Jiang-Tao Yu, Liangtao Zheng, Jingjing Gong, Tao Jiang, Xipeng Qiu, and Hang Zhao. FASTer: Toward powerful and efficient autoregressive vision–language–action models with learnable action tokenizer and block-wise decoding. In *Fourteenth International Conference on Learning Representations*, 2025.

Yicheng Liu, Zibin Dong, Baijun Ye, Tianyuan Yuan, Tao Jiang, Anqi Yang, Shicheng Cao, Haonan Liu, Yue Sun, Zihan Guo, Xiao Liu, Dong Ke, Changxun Pan, Chenru Wu, Tailai Cheng, Xiaoshu Ren, Xinlei Zhang, Jianning Cui, Zijie Zhao, Haoyu Zhang, Kaiming Xu, Haodong Yang, Bowen Zhang, Jiahui Niu, Shaoting Zhu, Shiduo Zhang, and Hang Zhao. G0.5: One autoregressive stream for robot reasoning and action. *arXiv preprint arXiv:2608.11739*, 2026b. URL https://arxiv.org/abs/2608.11739v1.

Huaihai Lyu, Chaofan Chen, Senwei Xie, Pengwei Wang, Xiansheng Chen, Shanghang Zhang, and Changsheng Xu. OmniSAT: Compact action token, faster auto regression. *arXiv preprint arXiv:2510.09667*, 2025. URL https://arxiv.org/abs/2510.09667v1.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *Twelfth International Conference on Learning Representations*, 2024.

Yao Mu, Tianxing Chen, Zanxin Chen, Shijia Peng, Zhiqian Lan, Zeyu Gao, Zhixuan Liang, Qiaojun Yu, Yude Zou, Mingkun Xu, Lunkai Lin, Zhiqiang Xie, Mingyu Ding, and Ping Luo. RoboTwin: Dual-arm robot benchmark with generative digital twins. *arXiv preprint arXiv:2504.13059*, 2025. URL https://arxiv.org/abs/2504.13059.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024. URL https://arxiv.org/abs/2405.12213v2.

Open X-Embodiment Collaboration et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903, 2024.

Austin Patel and Shuran Song. GET-Zero: Graph Embodiment Transformer for Zero-Shot Embodiment Generalization. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14262–14269, May 2025. doi: 10.1109/ICRA55743.2025.11127922. URL https://arxiv.org/abs/2407.15002.

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. URL https://arxiv.org/abs/2501.09747v1.

Lirui Wang, Xinlei Chen, Jialiang Zhao, and Kaiming He. Scaling Proprioceptive-Visual Learning with Heterogeneous Pre-trained Transformers. *Advances in Neural Information Processing Systems*, 37:124420–124450, December 2024. doi: 10.52202/079017-3952. URL https://arxiv.org/html/2409.20537v1.

Jinliang Zheng, Jianxiong Li, Dongxiu Liu, Yinan Zheng, Zhihao Wang, Zhonghong Ou, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. Universal Actions for Enhanced Embodied Foundation Models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22508–22519, June 2025a. doi: 10.1109/CVPR52734.2025.02096. URL https://arxiv.org/abs/2501.10105.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model, 2025b. URL https://arxiv.org/abs/2510.10274.

A METHOD DETAILS

A.1 VIRTUAL-NODE PLACEMENT AND INPUT CONVENTIONS

Table 4: Selected benchmark action interfaces and default latent counts. These five instances are a subset of the 18 pretraining embodiments. The first four are the selected dual-arm evaluation instances. LIBERO supplies one end-effector node; no unobserved intermediate joints are introduced.

Embodiment	Represented nodes	J	V
Aloha	2×(six arm joints + gripper)	14	4
UR5	2×(six arm joints + gripper)	14	4
Piper	2×(six arm joints + gripper)	14	4
PiperX	2×(six arm joints + gripper)	14	4
LIBERO	End-effector pose + gripper opening	1	1

Robot declarations specify ordered arm links, finger entries, inter-chain connections, and optional mirror groups. The allocation count n_c need not equal the output-node count when multiple finger entries share a gripper output. Quaternions use $wxyz$ order. Real-node graph distances use unit-hop edges and declared inter-chain connections. Placement candidates subdivide each physical graph edge into 20 equal intervals. The placement heuristic searches for a set of V candidates using the lexicographic criteria: maximum real-to-nearest-virtual distance, mean such distance, and negative minimum pairwise virtual distance. Declared mirror groups restrict the candidate sets and refinement moves. Large candidate sets use greedy farthest-point initialization before refinement; the construction does not certify a globally optimal placement. For a single articulated chain, the construction uses a synthetic mirror twin to resolve placement ties consistently with a dual-arm layout. The EE-only LIBERO interface uses one represented node and one virtual node. Real-to-virtual and virtual-to-virtual entries of Δ use shortest-path distances in physical-hop units, including fractional offsets; the real-to-real block retains the original graph distances.

The four dual-arm instances in Table 4 have $J = 14, V = 4$ at the default ratio, whereas LIBERO has $J = 1, V = 1$. Other corpus embodiments have different joint counts; for example, the dual-arm Franka interface has $J = 16$.

Link poses use the head-camera or third-person-camera frame. They are supplied by the dataset or computed from joint angles by forward kinematics when a robot kinematic model is available. Before encoding, quaternions are normalized and standardized to a common hemisphere. Positions are centered at the mean represented-node position in frame-zero state s and scaled by the declared arm length. Node scalars use per-robot scale/shift tables derived from declared limits. These preprocessing operations produce the normalized absolute trajectory $\overline{A}$ in Eq. (1). The encoder receives $\overline{A}$ without current-state feature tokens; state is used for centering during preprocessing and for conditioning the decoder. The decoder network predicts joint-angle differences and pose changes relative to s. These predictions are converted to absolute joint and link-pose trajectories using the current state. All predictions refer to frame zero, rather than accumulating frame-to-frame increments. Gripper openings are the exception: their scalars are predicted as absolute values. The network output $\widehat{R}$ and postprocessed absolute trajectory $\widehat{A}$ are therefore distinct. The output head uses temporal upsampling and centered convolutions; attention causality therefore does not establish exact decoded-prefix invariance.

Inherited graph attention. In Eq. (3), the distance embedder uses sinusoidal features followed by a two-layer MLP with a SiLU activation. Its output is reshaped into one channel-wise gate per head and node pair. The same distance modulates interactions across temporal patches; temporal rotary encoding supplies the time dependence. Encoder attention uses real and virtual nodes, decoder cross-attention uses the real-to-virtual block, and real-node decoder attention uses the real-to-real block.

A.2 RESIDUAL QUANTIZATION

For each latent $h = H_{v,t}$, a shared affine projection produces $u \in \mathbb{R}^4$. Let $L = (8, 5, 5, 5)$, $b = 1 + (L - 1)^{-1}$, and initialize $r_0 = b \odot \tanh(u \oslash b)$, with elementwise operations. For residual

stage q, define $s_q = L^{-q}$ and

$$c_q = \mathrm{FSQ}(r_q \oslash s_q), \qquad \widetilde{c}_q = s_q \odot c_q, \qquad r_{q+1} = r_q - \mathrm{sg}(\widetilde{c}_q). \tag{7}$$

For scalar coordinate x with L_d levels, the configured FSQ mapping is

$$\mathrm{FSQ}_{L_d}(x) = \frac{2}{L_d - 1}\left\lfloor \frac{L_d - 1}{2}\big(\mathrm{clip}(x, -1, 1) + 1\big) + \frac{1}{2}\right\rfloor_{\mathrm{STE}} - 1, \tag{8}$$

where rounding uses a straight-through estimator. The four level indices are packed with mixed-radix basis $(1, 8, 40, 200)$ into one index $Z_{v,t,q} \in \{0, \ldots, 999\}$. An r-stage prefix reconstructs the latent as $\overline{h}^{(r)} = W_{\mathrm{out}} \sum_{q<r} \widetilde{c}_q + b_{\mathrm{out}}$. The output bias is included once per prefix. This residual scaling and bounding configuration specifies our implementation; the underlying scalar quantization follows FSQ (Mentzer et al., 2024).

A.3 Reconstruction and Policy Loss Details

Let $\ell_w(B, A)$ sum weighted smooth-L_1 errors on node scalars, positions, and rotation-matrix entries, using $\beta = 10^{-4}$ and channel weights w. Errors are summed over node/channel entries and averaged over valid frames. A temporal patch is valid only if all its frames are valid. Scalar errors use the normalized scalar space, whereas position errors use the robot's metric frame; quaternion outputs are converted to 3×3 rotation matrices for the rotation term. Channel weights are dataset-dependent.

When kinematics are supplied, define $F(B)$ to preserve B's scalar channels and replace its pose channels with forward kinematics of the predicted joint scalars, expressed in the same loss space. The reconstruction objective is

$$\mathcal{L}_{\mathrm{rec}}(B, A) = \ell_w(B, A) + \mathbf{1}_{\mathrm{FK}}\left[\ell_w(F(B), A) + \ell_w\big(B, \mathrm{sg}(F(B))\big)\right]. \tag{9}$$

The scalar component of $\ell_w(F(B), A)$ repeats direct scalar supervision, while its pose terms differentiate through forward kinematics to the predicted scalars. In the consistency term, the scalar difference is identically zero and the pose terms pull predicted poses toward detached forward-kinematics poses. The same objective applies to each restored residual-prefix reconstruction in Eq. (4). When kinematics are absent, only the direct reconstruction term is used.

The clean-context factorization in Eq. (6) defines the model; the implemented training objective additionally uses corrupted input codes, label smoothing, and residual-stage masks:

$$\mathcal{L}_{\mathrm{policy}} = \mathbb{E}_{\widetilde{Z}|Z}\left[\sum_{q=0}^{Q-1} \alpha_q \frac{\sum_{v,t} M_{v,t,q}\ \mathrm{CE}_\epsilon\Big(Z_{v,t,q}, p_{\theta,v,t,q}(\widetilde{Z}; \mathcal{C})\Big)}{\max(1, \sum_{v,t} M_{v,t,q})}\right] + \mathcal{L}_{\mathrm{depth}}. \tag{10}$$

Here M combines valid-patch and cascading corruption masks, $\epsilon = 0.1$, and $\alpha = (1, 0.8, 0.6)$; the inherited visual branch supplies auxiliary depth supervision. Input codes are independently replaced by uniform indices with stage probabilities $(0.2, 0.5, 0.7)$, while targets remain unchanged. For a given (v, t), corruption at stage q removes loss terms at later residual stages, not at stage q itself.

For policy learning, let $U_{v,t,q}$ mark an input-code replacement, and let $m_{v,t}$ mark a valid temporal patch. The cascading mask is

$$M_{v,t,q} = m_{v,t} \prod_{k=0}^{q-1} (1 - U_{v,t,k}). \tag{11}$$

Masked entries are removed from both the numerator and denominator of each stage's average. A cue never reads its own code, so replacing that code affects later queries that use it as context, rather than the prediction at that same cell. The configured policy also drops state conditioning with probability 0.05 during training. These training perturbations are disabled for generation.

A.4 Attention Mask and Enumerated Generation Example

The virtual-joint index contributes to the generation step but does not impose a unidirectional virtual-joint attention mask. The virtual-joint axis admits bidirectional relationships subject to the delay

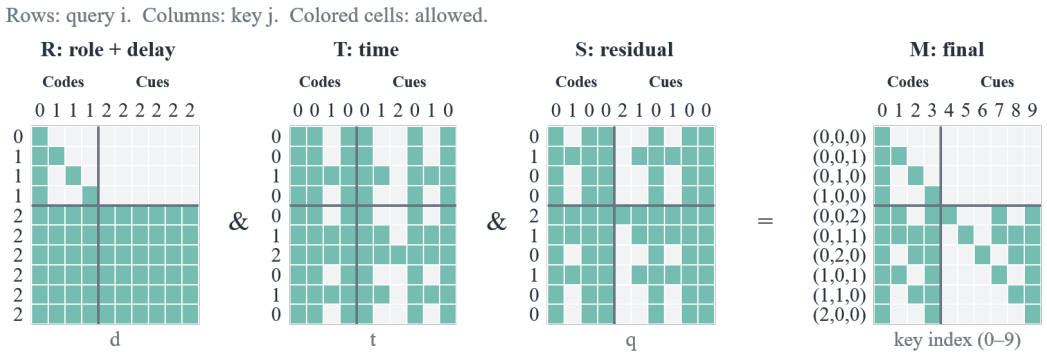


Figure 8: **Factorization of the visibility mask in Figure 4(b).** R applies the code/cue role and delay rules, including self-attention; T admits keys at time $t' \leq t$; S admits keys at residual stage $q' \leq q$. Their intersection gives M. Rows and columns of the first three matrices are labeled by the corresponding coordinate; the final matrix labels query rows by (v, t, q) and key columns by zero-based position index.

and the causal time/residual masks. The number of generation rounds is $T_p + Q + V - 2$. At $V = 4, T_p = 16, Q = 3$, full-sequence training uses 384 code/cues for 192 code targets and generation takes 21 rounds.

Write a position as $x = (i, \tau)$ with type $\tau \in \{\text{code}, \text{cue}\}$. For query $x = (i, \tau)$ and key $y = (j, \tau')$, grid attention permits

$$\begin{aligned} \text{allow}(x, y) = \mathbf{1}[x = y] \vee \\ (B(i, j) \wedge [(\tau' = \text{code} \wedge d_j < d_i) \vee \\ (\tau = \tau' = \text{cue} \wedge d_j = d_i)]) . \end{aligned} \tag{12}$$

The diagonal self-edge is separate from the code/cue pair at the same cell: a cue's self-edge contains no target value. State keys are available outside this grid mask, subject to the training-time state drop. Visual and language features are supplied through separate attention branches.

Figure 8 decomposes the step-2 visibility in Figure 4(b). The sequence contains four known code positions from steps 0 and 1, followed by six cues for the step-2 targets. Rows are queries, columns are keys, and colored cells permit attention. The R panel includes position roles, delay steps, and self-attention; intersecting it with the time panel T and residual panel S gives the final visibility M.

Take $V = T_p = Q = 2$ for illustration. The eight codes occupy four delay groups:

$$\begin{aligned} \mathcal{G}_0 &= \{(0, 0, 0)\}, \\ \mathcal{G}_1 &= \{(0, 0, 1), (0, 1, 0), (1, 0, 0)\}, \\ \mathcal{G}_2 &= \{(0, 1, 1), (1, 0, 1), (1, 1, 0)\}, \\ \mathcal{G}_3 &= \{(1, 1, 1)\}. \end{aligned} \tag{13}$$

Even earlier-round codes can be excluded: cue $(1, 0, 1)$ cannot read code $(0, 1, 0)$ because its time index is later.

Table 5 enumerates the example in Eq. (13). Each row lists the keys a cue may read directly, excluding its own cue self-edge.

Codes from the preceding group are embedded and made available before the next group's predictions; each position is processed under its own grid mask. Image, language, state, and previously computed grid keys/values are cached for reuse. The visibility construction excludes paths from a cue's target code or later-group codes to that cue.

A.5 Implementation Settings

Figure 9 details the tokenizer's computation and pre-norm blocks.

Table 5: Complete visibility for $V = T_p = Q = 2$. Letters denote (v, t, q) cells. Code columns contain only strictly earlier groups; same-group cue visibility remains subject to both axis constraints.

Cell	(v, t, q)	Round	Readable codes	Other readable cues
a	$(0, 0, 0)$	0	none	none
b	$(0, 0, 1)$	1	a	d
c	$(0, 1, 0)$	1	a	d
d	$(1, 0, 0)$	1	a	none
e	$(0, 1, 1)$	2	a, b, c, d	f, g
f	$(1, 0, 1)$	2	a, b, d	none
g	$(1, 1, 0)$	2	a, c, d	none
h	$(1, 1, 1)$	3	a, b, c, d, e, f, g	none

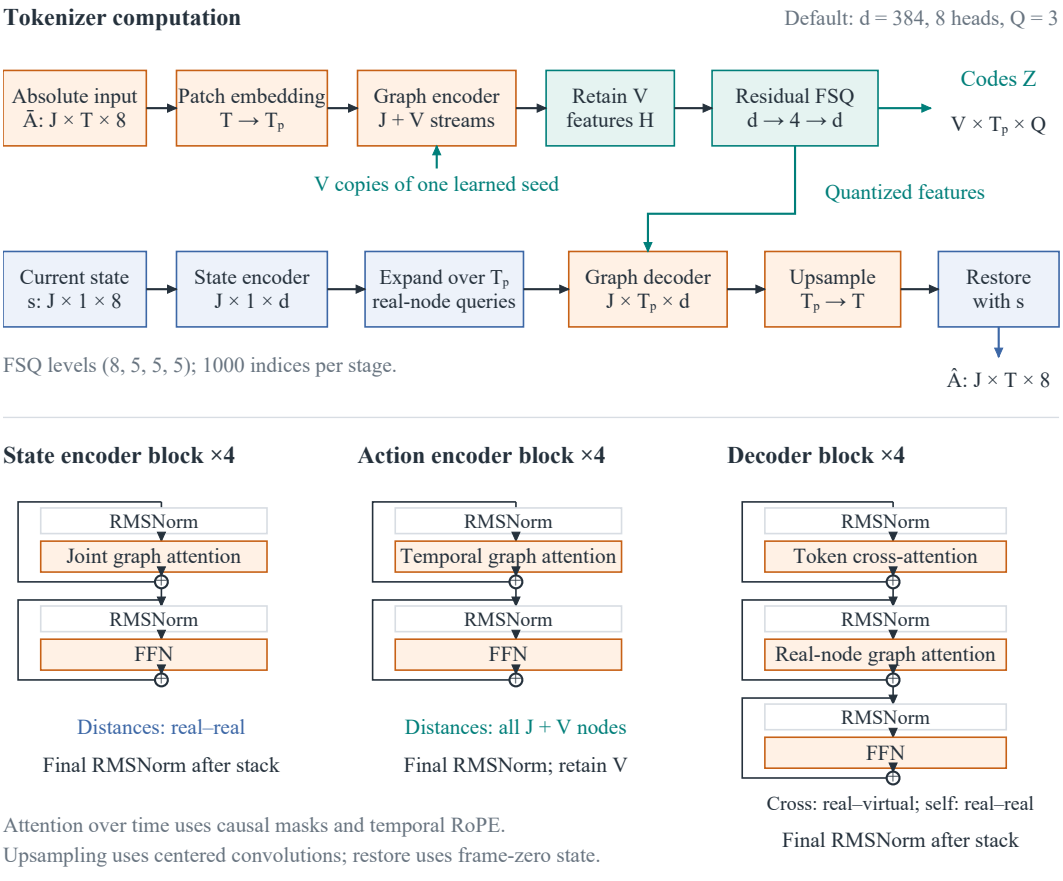


Figure 9: **Tokenizer architecture.** The encoder embeds temporal patches of normalized absolute trajectories, appends virtual nodes initialized from one shared learned vector, and retains their features after graph attention. Residual FSQ returns discrete indices and quantized features through shared input/output projections. State-encoder features initialize real-node queries at every temporal patch; decoder cross-attention reads quantized virtual features. Expanded blocks show pre-normalization and residual additions; each stack has four blocks and a final RMSNorm. Graph attention uses the indicated subblocks of Δ. The upsampling head predicts relative joint and pose changes, and restoration with frame-zero state recovers absolute trajectories; gripper scalars follow Appendix A.1. The diagram uses the default settings in Table 6.

Distance encoding uses 256 sinusoidal features with maximum period 10,000 and input scale one. Temporal rotary encoding uses base 10,000. The policy uses a Qwen3.5 vision-language backbone. Its robot-state encoder is initialized from the tokenizer and then trained independently while the tokenizer remains frozen. Each of the Q classification heads is shared across virtual joints and

Table 6: Default tokenizer settings. One action chunk contains the full future trajectory; a temporal patch is the shorter input unit.

Setting	Value
Action chunk length T / current-state length	64 / 1 frames
Frames per temporal patch / patch count T_p	4 / 16
Hidden width d / attention heads	384 / 8
Tokenizer encoder / decoder blocks	4 / 4
Virtual-joint allocation ratio ρ	6
Residual stages Q / indices per stage K	3 / 1000
FSQ scalar levels	$(8, 5, 5, 5)$
Residual-prefix loss weights λ	$(0.25, 0.5, 1)$

temporal patches. For $V = 4$, the default settings produce 192 code targets and 384 code/cues, generated in 21 rounds.

B Additional Experimental Results

B.1 Evaluation Protocol and Reporting

Table 7 distinguishes the task sets used for pretraining evaluation, downstream post-training, and design studies. In UR5 pretraining evaluation, `handover block` and `put object cabinet` are excluded because their evaluation environments consistently hang. The denominator is therefore 48 tasks; the two exclusions are not counted as policy failures. The mixed-embodiment pool and few-shot experiments use 16 tasks per evaluated RoboTwin embodiment. The mixed-embodiment pool samples 50% RoboTwin (Aloha and UR5), 25% real-world seven-task (Piper/PiperX), and 25% LIBERO data; representation studies train with 200 Aloha and 50 UR5 demonstrations per task. Few-shot co-training adds 200 Aloha demonstrations per task at UR5 budgets of 2, 5, and 10. Aloha post-training uses 600 demonstrations per task. All success differences in the text are absolute percentage-point differences.

Table 7: Evaluation scope and nominal trial counts. The 50 trials/task convention applies to simulation, not the real-robot progress scores.

Study	Evaluation	Tasks	Trials/task
Pretraining checkpoints	Aloha clean/randomized	50	50
Pretraining checkpoints	UR5 clean	48	50
Aloha post-training	Aloha clean/randomized	50	50
Mixed-embodiment pool / few-shot	RoboTwin per embodiment	16	50
Simulation studies	LIBERO	Suite-dependent	50
Simulation studies	LIBERO-Plus	Perturbation task set	1
Real deployment	Piper / PiperX	7	Progress scoring

Timing. Timing comparisons use an RTX 5090 and measure model computation per 64-frame action chunk, excluding data loading. They do not measure the complete sensing, communication, and robot-control loop. Reported speedups divide the central latency values: 337.9/158.4 on RoboTwin and 239.6/126.0 on LIBERO for HoloBrain versus HoloBrain-Discrete. The corresponding mean $\pm$ standard deviation values are 337.9 ± 2.4 versus 158.4 ± 1.4 ms/chunk on RoboTwin and 239.6 ± 1.8 versus 126.0 ± 1.8 ms/chunk on LIBERO. The separate Full AR comparison uses 553.3/158.4 and 186.6/126.0. These comparisons answer different questions and are not combined into a single speedup. The $\pm$ values denote standard deviation.

Few-shot scope. Few-shot policies use a tokenizer pretrained with UR5 motion; they do not measure tokenizer generalization to an unseen embodiment. Task overlap and the contribution of source data should likewise be distinguished from held-out-task generalization.

B.2 Complete Pretraining and Simulation Results

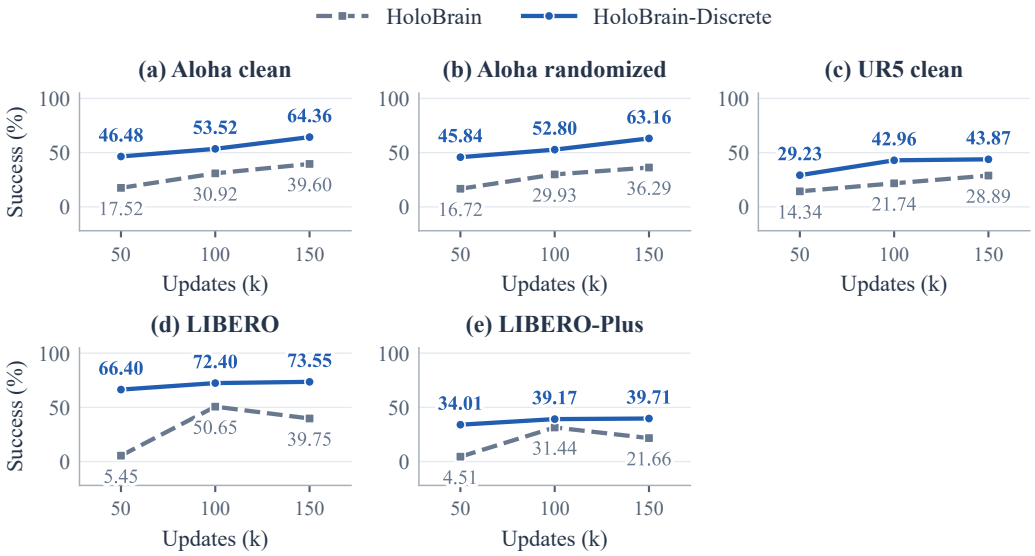


Figure 10: All five pretraining evaluations at 50k, 100k, and 150k updates. Lines connect observed checkpoints without smoothing or interpolation of additional measurements. Task sets are listed in Table 7.

Table 8: Post-training LIBERO success rates (%). Long denotes LIBERO-10.

Method	Spatial	Object	Goal	Long	Average
HoloBrain	97.20	99.60	97.60	95.20	97.40
Ours	95.20	99.40	97.60	93.20	96.35

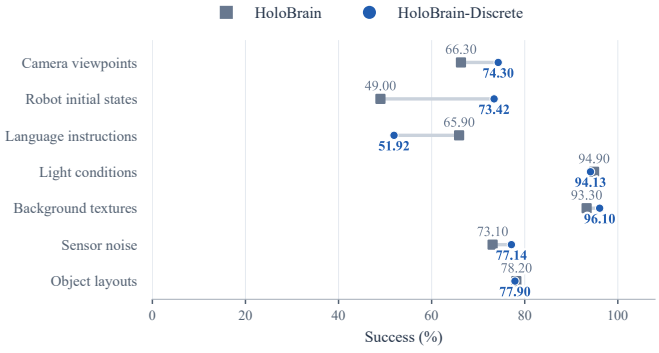


Figure 11: Post-training success under the seven LIBERO-Plus perturbation categories. Square and circular markers denote HoloBrain and HoloBrain-Discrete. Initial-state and camera-viewpoint gains coexist with a decline under language perturbations. The separately reported aggregate scores are 72.60% and 76.33%; they are not recomputed as an unweighted mean of these categories.

HoloBrain-Discrete improves robot-initial-state success by 24.42 points and camera-viewpoint success by 8.00 points, but decreases language-instruction success by 13.98 points. The aggregate improvement therefore has a specific scope rather than indicating a uniform gain across perturbations.

B.3 Few-Shot Results and Real-World Deployment

Table 9: Complete UR5 few-shot success rates (%), rounded to two decimals. The 50-demo target-only column is a reference, not an upper bound. Aloha sampling fractions for the 10-/5-/2-demo mixtures are 80%/90%/96%. OAT uses the tokenizer trained for 400k updates. The best success in each column is bold and the second best underlined.

	50 demos	10 demos		5 demos		2 demos	
Method	Only	Only	+Aloha	Only	+Aloha	Only	+Aloha
FAST	24.88	0.69	13.69	0.38	5.75	0.19	3.75
OAT	29.75	18.00	26.13	**13.38**	26.38	**11.25**	<u>17.75</u>
HoloBrain	<u>40.75</u>	<u>23.38</u>	<u>40.69</u>	<u>12.25</u>	<u>39.56</u>	<u>7.13</u>	16.69
Ours	**54.00**	**23.56**	**50.56**	9.44	**52.75**	6.00	**39.38**

Table 10: Real-world progress scores (%). PiperX uses ten demonstrations per task at a 10% target sampling fraction. Piper and PiperX columns are different deployment conditions, not a controlled source-on/off comparison. The unweighted seven-task Piper means are 85.30 for HoloBrain and 74.29 for HoloBrain-Discrete.

Task	HoloBrain (Piper)	Ours (Piper)	Ours (PiperX)
Place empty cup	78.5	84.0	81.5
Place shoe	93.2	93.2	81.3
Stack three blocks	83.3	65.6	51.1
Stack three bowls	100.0	96.4	91.1
Put bottles in dustbin	95.2	75.7	53.8
Place to slot	62.2	49.8	73.6
Fold towel	84.7	55.3	41.1

Our largest Piper deficits relative to HoloBrain occur in towel folding (-29.4 points), bottle disposal (-19.5), and block stacking (-17.7). Cup placement improves by 5.5 points and shoe placement is unchanged. These taskwise differences motivate further analysis of execution failures; the aggregate scores alone do not identify whether errors originate in perception, token prediction, reconstruction, or control. PiperX reaches 91.1 progress points on bowl stacking and 81.5 on cup placement, with lower scores on towel folding and block stacking.

Table 11: Real-task milestone scoring. Progress is the attained score divided by the task maximum, expressed as a percentage.

Task	Milestones	Maximum
Place empty cup	Grip 0.5; on mat 0.2; correctly placed 0.3.	1.0
Place shoe	Lift 0.4; partially on mat 0.2; fully on mat 0.2; direction 0.2.	1.0
Stack three blocks	Per block: pick 0.25; place 0.25.	1.5
Stack three bowls	Per bowl: pick 0.25; misaligned stack 0.15; correct stack adds 0.1. Up to three attempts per bowl.	1.5
Put bottles in dustbin	Per bottle: pick 0.3; place 0.2; three bottles.	1.5
Place to slot	Per block: pick 0.25; placement within 2 cm adds 0.15; correct placement adds 0.1. Up to three attempts per block.	1.5
Fold towel	Pick corner 0.1; first fold 0.3; right-side pick 0.1; second fold 0.3; quality 0.2.	1.0

B.4 Extended Representation Comparisons

Table 12: Extended mixed-embodiment pool representation comparisons: success (%), codes per 64-frame chunk, and vocabulary (classes per code). Codes per chunk times classes per code gives each representation's nominal index storage, not its empirical information content. OAT is the main-text configuration: registers compress each temporal patch, with tokenizer training extended to 400k updates; OAT (per-patch, 200k) is its standard-budget counterpart. The whole-chunk variants apply OAT's native whole-trajectory compression to the full 64-frame chunk, using eight registers. All representation policies use 200k updates. FAST lengths (*) are dataset averages. An unavailable result is shown as –. The best success in each column is bold and the second best underlined.

Representation	Codes/chunk RoboTwin / LIBERO	Vocabulary (classes/code)	Aloha	UR5	LIBERO	Plus
OAT (whole-chunk)	8 / 8	1000	10.00	0.25	65.85	49.66
OAT (whole-chunk, 400k)	8 / 8	1000	38.25	20.38	69.95	–
OAT (per-patch, 200k)	64 / 64	1000	34.00	1.25	72.90	51.91
OAT	64 / 64	1000	59.00	33.25	72.80	56.87
FAST	63.2 / 113.9*	2050	<u>73.13</u>	29.88	59.30	14.93
Ours (96 codes)	96 / 48	1000	62.38	<u>37.50</u>	**94.15**	<u>63.91</u>
Ours (192 codes)	192 / 48	1000	**73.25**	**57.50**	<u>93.60</u>	**66.62**

OAT's native compression unit is the whole trajectory; the whole-chunk variants apply it to the full 64-frame chunk with eight registers, while the OAT of the main text compresses each temporal patch with 64, and our configurations use 96 (one virtual joint per arm) and 192 codes on RoboTwin. Both changing the compression unit and extending tokenizer training to 400k updates improve the RoboTwin scores, which argues against evaluating OAT only at the weakest setting: the per-patch run reaches 59.00% Aloha and 33.25% UR5 at the 400k tokenizer budget, versus 34.00% and 1.25% at 200k. Its LIBERO score is essentially unchanged, at 72.80% versus 72.90%. These results do not imply that code count alone explains the remaining gap.

Table 13: Tokenizer reconstruction errors, reported as MAE / MSE. Normalized aggregate metrics are not converted to physical units or averaged across datasets.

MAE / MSE	LIBERO-10	Real seven	Aloha	UR5
Ours (192)	0.052495 / 0.045266	0.040122 / 0.015675	0.003635 / 0.001805	0.005339 / 0.002181
Ours (96)	0.052204 / 0.043562	0.030600 / 0.007911	0.004510 / 0.002336	0.006012 / 0.002452
OAT whole	0.091317 / 0.075645	0.020480 / 0.004223	0.020655 / 0.009480	0.030637 / 0.008247
OAT whole ext.	0.090344 / 0.074580	0.017228 / 0.003123	0.014142 / 0.006444	0.015255 / 0.005849
OAT 200k	0.085481 / 0.073238	0.014754 / 0.002372	0.015203 / 0.006511	0.025491 / 0.004480
OAT	0.083609 / 0.072175	0.011239 / 0.001543	0.009494 / 0.003578	0.009472 / 0.001926
FAST	0.011170 / 0.005515	0.003241 / 0.000178	0.003972 / 0.001084	0.004351 / 0.000144

FAST reconstructs LIBERO and real-seven-task trajectories with lower reported MAE than our tokenizer, yet achieves lower LIBERO closed-loop success. Likewise, the OAT variants have lower real-seven-task reconstruction MAE without establishing superior real-world closed-loop policies in this mixed-embodiment pool study, where real-robot rollout results are not available. Offline error and deployment performance answer different questions.

B.5 Complete Compression and Schedule Measurements

Table 14: Compression sweep: success (%) and RoboTwin latency (ms/chunk). The 14-position configuration uses 672 codes; virtual-joint counts of 8, 6, 4, and 2 yield 384, 288, 192, and 96 codes. LIBERO retains one joint-axis position and 48 codes across all runs. The best success in each column is bold and the second best underlined.

Condition	Codes	Aloha	UR5	LIBERO	Plus	ms/chunk
14 joint-axis positions	672	59.75	33.25	**95.35**	63.13	324.9 ± 2.1
$V = 8$	384	67.63	<u>54.50</u>	<u>95.00</u>	<u>66.40</u>	205.0 ± 1.5
$V = 6$	288	<u>73.13</u>	54.38	93.60	57.94	181.4 ± 1.9
$V = 4$	192	**73.25**	**57.50**	93.60	**66.62**	158.4 ± 1.4
$V = 2$	96	62.38	37.50	94.15	63.91	139.6 ± 1.4

Table 15: Tokenizer reconstruction MAE / MSE across the compression sweep. Figure 7 divides each Aloha/UR5 MAE by its own 672-code baseline.

MAE / MSE	LIBERO-10	Real seven	Aloha	UR5
14 positions	0.053121 / 0.046152	0.025855 / 0.006223	0.003474 / 0.001562	0.003940 / 0.000528
$V = 8$	0.055330 / 0.046873	0.025203 / 0.006495	0.003431 / 0.001806	0.004127 / 0.000569
$V = 6$	0.050516 / 0.043784	0.025003 / 0.005283	0.003520 / 0.001797	0.004210 / 0.000671
$V = 4$	0.052495 / 0.045266	0.040122 / 0.015675	0.003635 / 0.001805	0.005339 / 0.002181
$V = 2$	0.052204 / 0.043562	0.030600 / 0.007911	0.004510 / 0.002336	0.006012 / 0.002452

The default does not maximize every benchmark: the largest joint-axis configuration reaches 95.35% on LIBERO versus 93.60% for the default. Because LIBERO's own code count does not change, this variation cannot be attributed to changing its local compression ratio. The common multi-source training and run variability are alternative explanations.

On LIBERO, where $V = 1$, removing only the virtual-joint delay leaves the schedule unchanged. Small observed differences between full delay and parallel V should therefore not be interpreted as evidence for a virtual-joint scheduling effect on this interface.

B.6 Input/Output, Prediction Positions, and Augmentation

These complementary studies use Aloha's 16-task evaluation with distinct training protocols. For the I/O comparison, tokenizer and policy training use 50 demonstrations per task. The prediction-position and augmentation comparisons freeze a pretrained tokenizer and train policies with 200 demonstrations per task. Their absolute scores should not be compared directly with the mixed-embodiment pool schedule table.

Table 16: Tokenizer I/O and state-feature conditioning on Aloha. All three configurations condition the decoder on the current state and predict relative outputs. The best success is bold and the second best underlined.

Encoder input	Encoder state	Decoder state	Output	Success (%)
Absolute	No	Yes	Relative	**57.6**
Absolute	Yes	Yes	Relative	<u>52.6</u>
Relative	Yes	Yes	Relative	48.4

Absolute input without encoder state conditioning achieves 57.6% success, compared with 52.6% when encoder conditioning is added. With encoder conditioning present, changing the input to

relative coordinates gives 48.4%. These comparisons support the selected input/conditioning combination; decoder conditioning and relative output are not varied here.

Table 17: Prediction-position and augmentation comparisons, each on the frozen-tokenizer Aloha protocol described above.

Configuration	Success (%)
Code/cue	88.1
Fused delay	77.0

Configuration	Success (%)
Full	**88.1**
Without token augmentation	<u>86.1</u>
Without either augmentation	80.6

The code/cue implementation improves over MusicGen-style fused delay by 11.1 points. This is an action-policy implementation comparison, not an evaluation of a pretrained music-generation model, and does not establish that embedding fusion inevitably loses information. Removing token augmentation decreases success by 2.0 points; removing the remaining action augmentation reduces it by a further 5.5 points. The sequential removals do not measure independent additive effects of the two augmentations.