REAT: RESIDUAL ACTION TOKENIZATION FOR AUTOREGRESSIVE ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive robot policies rely on action tokenizers to convert continuous action chunks into compact sequences of discrete prediction targets. Existing tokenizers can reconstruct action chunks from discrete codes, but often leave implicit what each individual token contributes to the reconstructed action. We introduce Residual Action Tokenization (ReAT), which gives each token an explicit role in refining the current action reconstruction. It uses the decoded reconstruction as feedback to construct the next token from the remaining action error. This closes the loop between encoding and decoding, making each new token responsive to the errors left by the actual decoded prefix. Across 19 manipulation tasks spanning three simulation benchmarks and real-world settings, 8-token ReAT policies outperform the evaluated baselines in task success; on LIBERO, even 4-token ReAT surpasses the evaluated 8-token action-tokenization baselines at half the token budget. These results suggest a path toward action tokenization that supports self-corrective policy learning, helping autoregressive policies learn to progressively refine their own action plans.

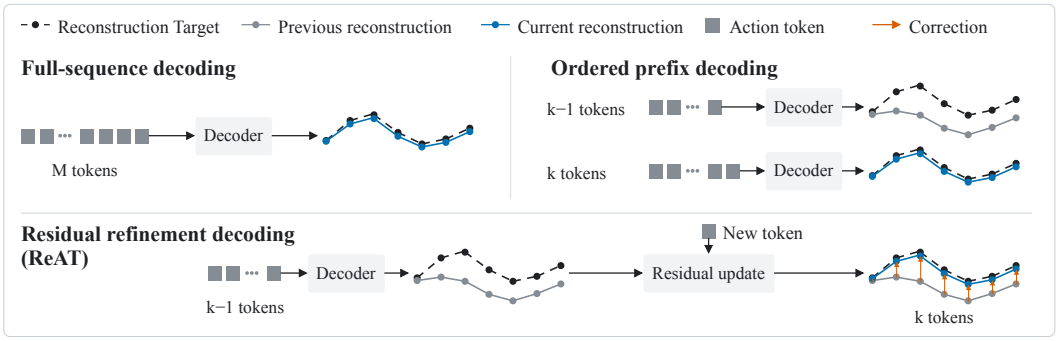

Figure 1: **Comparison of action-token decoding strategies.** Full-sequence decoding requires all M tokens, while ordered prefix decoding reconstructs a complete action chunk from any nonempty prefix. ReAT combines the existing draft with each new token to produce a learned correction. Orange arrows indicate the corrections applied to the previous reconstruction in ReAT; target trajectories are shown only for reference.

1 INTRODUCTION

Autoregressive robot policies generate future actions by predicting discrete tokens one at a time (Brohan et al., 2023; Kim et al., 2024). To connect this prediction process to continuous motion, action tokenizers encode future action chunks into token sequences and decode the predicted tokens back into executable actions (Mete et al., 2024; Pertsch et al., 2025). The encoded sequences provide the policy's training targets, so the tokenizer determines how action information is distributed across tokens and how each token depends on its predecessors (Liu et al., 2026; Dong et al., 2026). The question is then how to structure these targets so that each token has a clear role in reconstructing the action chunk.

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



Figure 2: **Arm trajectory reconstruction with residual tokens.** A 32-step LIBERO example at $K = 2, 4, 6, 8$ using an earlier ReAT tokenizer. Five sampled poses show translucent motion history and an opaque final pose. MSE is computed over all steps and normalized action dimensions of this example.

Learned action tokenizers use reconstruction objectives to compress action chunks into discrete sequences for policy learning (Lee et al., 2024; Mete et al., 2024). Recent work further shows that reconstruction fidelity alone is insufficient to characterize these representations, as token organization also affects downstream policy learning (Dong et al., 2026). Ordering constraints and prefix supervision provide additional structure by encouraging earlier tokens to preserve essential action information and later tokens to add detail (Liu et al., 2026). However, these objectives do not by themselves explicitly define each token's contribution relative to the actions already reconstructed. That contribution can remain implicit in the learned encoder–decoder mapping, leaving an opportunity to ground successive tokens in the action information still missing from the current reconstruction.

The residual between a target action chunk and its current reconstruction provides a concrete description of this missing information. Because it is measured in the reconstructed actions, it reflects what preceding tokens have actually recovered, including the effects of quantization and decoding. Encoding this residual gives the next token an explicit target tied to the reconstruction produced so far. This motivates Residual Action Tokenization (ReAT), which structures action sequences as successive corrections to an evolving action draft.

ReAT implements this formulation through explicit reconstruction feedback. During tokenizer training, the encoder maps the remaining reconstruction error to a discrete token, and the decoder uses that token together with the current action draft to produce a learned correction (Figure 1). The updated draft then provides the basis for constructing the next token. Each additional token thus represents a correction to the existing action draft, with Figure 2 illustrating how reconstruction improves as tokens are added. The resulting residual tokens provide structured prediction targets for observation-conditioned autoregressive policies.

We evaluate ReAT on 19 manipulation tasks spanning LIBERO, MetaWorld, RoboCasa, and real-world bucket stacking. With 8 tokens, ReAT outperforms the evaluated action-tokenization and diffusion-policy baselines in task success on all three simulation benchmarks and the real-world task. On LIBERO, it surpasses the evaluated 8-token action-tokenization baselines using only 4 tokens. LIBERO ablations further support the contribution of explicit reconstruction feedback to policy performance and show that prefix supervision enables effective control with shorter token sequences.

In summary, our contributions are:

- We introduce ReAT, an action tokenizer that constructs each successive token from the residual left by the decoded prefix, explicitly incorporating quantization and decoding effects into token construction.
- We combine a draft-conditioned recurrent decoder with prefix reconstruction supervision to turn residual tokens into successive action corrections. Each nonempty prefix reconstructs a complete action chunk, allowing a single autoregressive policy to use different token budgets without retraining.
- Experiments across three simulation benchmarks and real-world bucket stacking show that 8-token ReAT outperforms the evaluated action-tokenization and diffusion-policy baselines in task success. On LIBERO, 4-token ReAT also surpasses the evaluated 8-token action-

tokenization baselines, while mechanism ablations support the contributions of reconstruction feedback and prefix supervision.

2 Related Work

Generative robot policies. Robot policies generate actions through continuous representations or discrete token sequences. Continuous approaches include variational action-chunk prediction (Zhao et al., 2023), diffusion-based generation (Chi et al., 2023), and flow-matching action experts (Black et al., 2024; NVIDIA et al., 2025). Discrete autoregressive policies predict action symbols sequentially (Kim et al., 2024), while recent extensions explore parallel continuous-chunk decoding (Kim et al., 2025) and discrete-action pretraining followed by flow-matching post-training (Physical Intelligence et al., 2025). HiFlow further combines scale-wise autoregressive generation with flow matching in continuous action space (Yashima et al., 2026). These approaches primarily advance the mechanisms for generating actions, while leaving open how discrete action targets should be structured to support progressive correction. ReAT addresses this representation question by making residual correction the organizing principle of action tokenization, providing autoregressive policies with corrective prediction targets.

Action tokenization and ordered prefixes. Learned action tokenizers compress action chunks into quantized representations for policy learning (Lee et al., 2024; Mete et al., 2024; Wang et al., 2025), while FAST exploits temporal regularity through frequency-domain coding and byte-pair encoding (Pertsch et al., 2025). Beyond compression, CARP organizes action representations into scales for next-scale prediction (Gong et al., 2025), and ActionCodec examines how token structure affects downstream policy optimization (Dong et al., 2026). Ordered tokenizers use nested dropout to prioritize information in early prefixes (Rippel et al., 2014), enabling flexible-length reconstruction in image and action domains (Bachmann et al., 2025; Liu et al., 2026). Compact representations and ordered prefixes improve the usability of action tokens, but do not by themselves make the remaining action reconstruction error an explicit basis for subsequent token construction. ReAT introduces this connection, giving successive tokens a correction target defined by what earlier tokens have actually reconstructed.

Residual coding and reconstruction feedback. Residual quantization successively approximates latent features, with applications in image generation (Lee et al., 2022), neural audio compression (Zeghidour et al., 2021; Défossez et al., 2022), and action tokenization (Lee et al., 2024; Wang et al., 2025; Gong et al., 2025). However, latent approximation error does not directly characterize the error remaining in decoded actions. Recurrent image codecs address the analogous distinction by feeding decoded-output errors back into progressive compression (Toderici et al., 2016a;b). ReAT brings this reconstruction-feedback perspective to action tokenization, constructing corrective representations for autoregressive policy learning.

3 Preliminaries

Let o_t denote an observation containing camera images and proprioception, and let c identify a task. A policy uses an observation history $O_t = o_{t-H_o+1:t}$ of length H_o to predict a chunk of H actions, each with d control dimensions. The robot executes the first $H_e \leq H$ commands before observing and replanning.

A tokenizer maps an expert action chunk $A \in \mathbb{R}^{H\times d}$ to a token sequence $z_{1:M}$, where M is the maximum token budget and $z_k \in \{0, \ldots, V-1\}$ indexes a vocabulary of size V. We seek a prefix-decodable representation: every nonempty prefix $z_{1:K}$, $1 \leq K \leq M$, should decode to a complete reconstruction $\widehat{A}^{(K)} \in \mathbb{R}^{H\times d}$. The tokenizer supplies training labels for an autoregressive policy $\pi(z_k \mid O_t, c, z_{<k})$. At deployment, the policy predicts tokens from observations and the decoder converts them into actions, without access to the expert chunk.

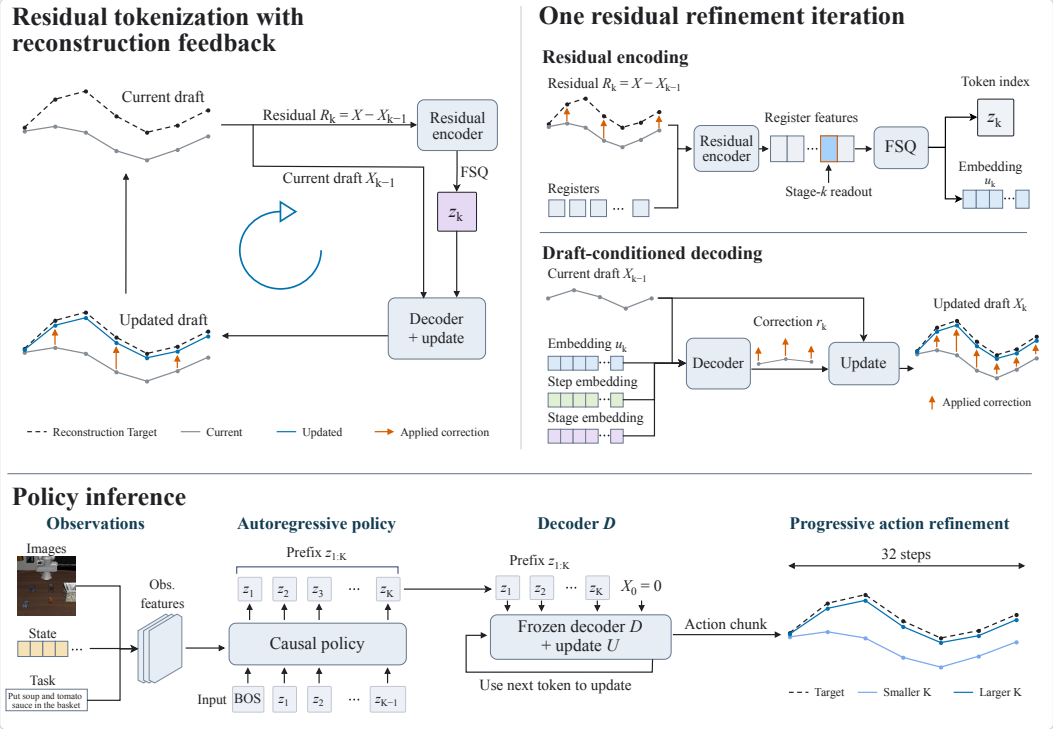

Figure 3: **Architecture of ReAT. Top left:** residual tokenization with reconstruction feedback across tokens. **Top right:** residual encoding and draft-conditioned decoding in one residual refinement iteration. **Bottom:** an observation-conditioned policy predicts tokens, whose prefixes are decoded by the frozen tokenizer.

4 Residual Action Tokenization

ReAT constructs action tokens through a reconstruction feedback loop (Figure 3). Starting from a zero draft, it compares the current reconstruction with the expert action chunk, encodes the remaining error into one token, and uses that token to refine the draft. The updated reconstruction then determines what the next token must encode. The same encoder and decoder are reused at every stage. This loop constructs training labels; at deployment, a policy predicts the tokens from observations and only the decoder is needed to reconstruct actions.

4.1 Residual Token Construction

The next token should describe what the decoded prefix has not yet reconstructed. Let $X \in \mathbb{R}^{H \times d}$ denote the expert chunk in reconstruction coordinates, and let X_{k-1} be the draft produced by the first $k-1$ tokens in those same coordinates. With $X_0 = 0$, stage k constructs a token from the remaining error:

$$R_k = X - X_{k-1}, \qquad (u_k, z_k) = \mathrm{FSQ}(E_\phi(R_k)_k)\,. \tag{1}$$

The Transformer encoder E_ϕ summarizes the residual using M register features, one for each refinement stage. The subscript k selects a stage-specific readout, allowing early and late stages to encode the residual differently. Finite scalar quantization (FSQ) (Mentzer et al., 2024) converts this feature into a discrete index z_k and its quantized embedding u_k. The index is the policy's training label, while the embedding is the decoder's input. Only one token is produced per pass, not M tokens at once. After decoding it, we rerun the encoder on the updated residual to construct the next token.

The encoder receives only the residual, not the draft as an additional input. Crucially, this residual is measured against the *actual decoded reconstruction*: it includes errors left by quantization and decoding, rather than a discrepancy between latent features. Reconstruction feedback therefore gives each successive token a corrective role relative to the decoded prefix.

We normalize action channels using fixed training-set statistics and cumulatively sum translation commands within each chunk to expose accumulated command error. Rotation channels retain per-step values, while gripper controls are updated separately. These coordinates represent commands, not physical poses; cumulative sums start at zero and are inverted before execution. Appendix A.3 defines the transform and recurrent state.

4.2 Reconstruction-Conditioned Refinement

The decoder interprets each new token in the context of what has already been reconstructed. It receives the token embedding u_k, the current draft X_{k-1}, and action-step and refinement-stage position features. A Transformer processes all H action positions and predicts a correction proposal for the entire chunk. Consequently, the same token can produce different corrections for different drafts; it does not denote a fixed standalone motion.

We scale the $\tanh$-bounded proposal to limit each stage's correction magnitude, then update the recurrent draft while bounding decoded per-step commands to the normalized action range. Control channels accumulate corrections before bounding. These bounds make the applied change differ from the raw proposal. Appendix A.4 gives the exact updates, and Table 7 lists the proposal scales.

Every stage produces a complete H-step chunk. Additional tokens can revise any action step before execution, so token count controls refinement depth rather than chunk duration. With deterministic decoding and a fixed initial state, a prefix always gives the same reconstruction regardless of whether more tokens will follow. Retaining the recurrent state allows decoding to continue from that prefix without repeating earlier updates.

Algorithm 1 summarizes token construction. D_θ predicts the draft-conditioned proposal, and Update applies the bounded recurrent updates in Appendix A.4, returning the new draft and action-space reconstruction.

Algorithm 1 ReAT tokenization with reconstruction feedback

Require: Target chunk X in reconstruction coordinates; token budget M
Ensure: Tokens $z_{1:M}$ and prefix reconstructions $\widehat{A}^{(1)}, \ldots, \widehat{A}^{(M)}$
$X_0 \leftarrow 0$; initialize decoder state to zero
for $k = 1, \ldots, M$ **do**
  $R_k \leftarrow X - X_{k-1}$
  $(u_k, z_k) \leftarrow \mathrm{FSQ}(E_\phi(R_k)_k)$
  $r_k \leftarrow D_\theta(u_k, X_{k-1}, k)$
  $(X_k, \widehat{A}^{(k)}, \mathsf{state}) \leftarrow \mathrm{Update}(\mathsf{state}, r_k)$
end for
return $z_{1:M}$ and $\widehat{A}^{(1)}, \ldots, \widehat{A}^{(M)}$

4.3 Training for Progressive Reconstruction

Residual feedback specifies what a token encodes, but does not by itself ensure that short prefixes reconstruct useful actions. We therefore train on reconstructions at different token budgets. We sample a prefix length J uniformly from 1 to M and compare its decoded chunk with the expert actions. The reconstruction loss ℓ_{rec} is a weighted combination of motion and control MSEs, measured after normalization in *per-step action coordinates*, not cumulative coordinates. The objective is

$$\mathcal{L}_{\mathrm{tok}} = \mathbb{E}_J\left[\ell_{\mathrm{rec}}(\widehat{A}^{(J)}, A)\right] + \lambda_p \mathcal{L}_{\mathrm{progress}} + \mathcal{L}_{\mathrm{reg}}. \tag{2}$$

Uniform prefix sampling gives early and late reconstructions equal expected weight, rather than rewarding only the final chunk. The progress term penalizes increases in motion and control MSE between successive prefixes, encouraging additional tokens to improve reconstruction without guaranteeing monotonic improvement. The auxiliary regularizer $\mathcal{L}_{\mathrm{reg}}$ combines penalties on correction magnitude and temporal spread; it does not assign tokens to fixed action steps. Appendix A.5 expands these terms and Table 7 lists their weights.

We jointly train the encoder and decoder through recurrent updates using straight-through gradients through FSQ. The decoded drafts provide both residual feedback and prefix supervision.

4.4 Autoregressive Policy Learning and Inference

After tokenizer training, we freeze the encoder, quantizer, decoder, and normalization statistics. Running the reconstruction feedback loop on each expert chunk produces deterministic target tokens $z^*_{1:M}$. An observation-conditioned policy then learns these tokens by teacher forcing:

$$\mathcal{L}_{\text{policy}} = -\mathbb{E}_{(O,c,A)} \sum_{k=1}^{M} \log \pi_\psi(z^*_k \mid O, c, z^*_{<k}). \tag{3}$$

The policy Transformer predicts all M target tokens, conditioned on camera and proprioceptive features, task identity c, and preceding tokens.

At inference, the policy generates an externally selected budget of K tokens, which the decoder converts into a complete action chunk without expert actions or residual encoding. The controller executes the first H_e actions before replanning. Changing K requires no policy retraining.

Token construction requires M encoder and decoder passes, whereas deployment uses K decoder passes in addition to policy generation. Section 5.2 reports the measured inference cost; Appendix A gives the implementation and deployment settings.

5 Experiments

We evaluate control performance across token budgets, reconstruction quality, inference cost, and the contributions of individual design choices.

5.1 Benchmarks, Data, and Evaluation Protocol

We evaluate ReAT on 19 manipulation tasks spanning LIBERO-10 (Liu et al., 2023), MetaWorld (Yu et al., 2020), RoboCasa (Nasiriany et al., 2024), and real-world bucket stacking (Section 5.3). We train one multitask tokenizer and policy per simulation benchmark. Table 1 summarizes the simulation data and evaluation scope, and Figure 4 shows example environments. Appendix B details task identities, action interfaces, and dataset and simulator provenance.

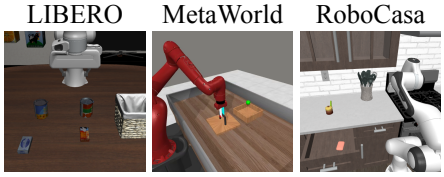

Figure 4: **Three benchmark domains.** Simulator renders (LIBERO, MetaWorld, RoboCasa).

Table 1: **Data and evaluation scope.** d: action dimension. Episodes: total evaluation rollouts.

Benchmark	Tasks	Demos/ task	d	Episodes
LIBERO-10	10	50	7	2500
MetaWorld	4	50	4	1000
RoboCasa	4	50 human + 150 generated	12	1000

All tokenizers and policies share a task-wise episode split, with 10% held out for reconstruction evaluation and normalization statistics computed from training episodes only. All methods use observation horizon $H_o = 2$, prediction horizon $H = 32$, and execution horizon $H_e = 16$. The main ReAT configuration uses a maximum of $M = 8$ content tokens and a vocabulary of $V = 1000$ codes. Main runs use batch size 256 and AdamW; Appendix A gives network dimensions, learning rates, and other implementation settings.

We conduct five complete simulation evaluation rounds for each trained policy, each comprising 50 episodes per task. Autoregressive decoding uses top-$k = 10$ and temperature 1; Diffusion Policy uses 10 DDIM steps. Success is averaged equally over tasks within each round, and tables report the mean $\pm$ standard error of the mean (SEM) across the five round-level averages. Appendix B.3 gives the estimator.

Table 2: **Policy success (%).** Mean ± SEM across five evaluation rounds. Subscripts: token counts; bold: best mean. $*$: prefix length not sampled during OAT training.

Method	LIBERO	MetaWorld	RoboCasa
DP	67.88 ± 0.83	27.90 ± 1.00	44.10 ± 0.80
Bin	12.12 ± 0.29	19.70 ± 1.46	40.10 ± 1.95
FAST	20.96 ± 0.64	20.80 ± 0.86	4.60 ± 0.99
QueST_8	65.52 ± 0.85	22.90 ± 0.53	52.70 ± 1.23
OAT_2	33.80 ± 0.81	27.00 ± 0.88	52.70 ± 1.21
OAT_4	50.96 ± 0.58	26.50 ± 1.92	53.30 ± 1.08
OAT_6^*	24.12 ± 0.39	26.70 ± 0.56	53.90 ± 1.85
OAT_8	61.48 ± 0.92	29.90 ± 1.39	53.60 ± 1.48
ReAT_2	36.20 ± 1.28	19.00 ± 0.79	54.60 ± 2.00
ReAT_4	66.40 ± 0.75	27.10 ± 0.91	54.50 ± 2.63
ReAT_6	68.44 ± 1.38	28.80 ± 1.43	$\mathbf{56.10} \pm 2.54$
ReAT_8	$\mathbf{71.16} \pm 0.68$	$\mathbf{31.70} \pm 1.02$	55.90 ± 0.62

Table 3: **Token count and end-to-end inference latency (ms).**

	LIBERO		MetaWorld		RoboCasa	
Policy	#Tok.	Lat.	#Tok.	Lat.	#Tok.	Lat.
DP	–	24.3	–	23.9	–	27.1
Bin	224	380.7	128	225.1	384	672.3
FAST	50.6	97.8	32.8	64.1	71.8	139.1
QueST	8	19.5	8	18.2	8	21.4
OAT_2	2	9.5	2	7.8	2	12.1
OAT_4	4	13.1	4	11.2	4	16.6
OAT_6	6	16.6	6	14.7	6	20.0
OAT_8	8	21.1	8	18.2	8	22.6
$\mathbf{ReAT}_2$	2	9.3	2	7.6	2	11.0
$\mathbf{ReAT}_4$	4	14.1	4	12.4	4	15.8
$\mathbf{ReAT}_6$	6	18.9	6	17.3	6	20.6
$\mathbf{ReAT}_8$	8	23.7	8	22.2	8	25.4

5.2 Policy Performance and Token-Budget Trade-offs

We first test whether ReAT supports effective control with compact action sequences and how task success changes when the token budget varies. We compare against OAT, QueST, FAST, scalar binning (Bin), and Diffusion Policy (DP), with success rates reported in Table 2. OAT and ReAT are evaluated at two, four, six, and eight tokens; QueST uses eight tokens, while FAST and Bin retain their native sequence lengths. OAT's nested dropout trains power-of-two prefixes, so its six-token result tests an intermediate budget not explicitly supervised during training.

At eight tokens, ReAT outperforms the evaluated baselines on all three benchmarks, exceeding the strongest baseline by 3.28, 1.80, and 2.00 percentage points on LIBERO, MetaWorld, and RoboCasa, respectively.

Varying ReAT's budget over $K \in \{2, 4, 6, 8\}$ without retraining increases success on LIBERO and MetaWorld, while RoboCasa varies little; its six-token mean exceeds the eight-token mean by only 0.20 percentage points, small relative to the SEMs. Prefix supervision optimizes reconstruction, so additional tokens need not improve task success monotonically. At the shared trained budgets $K \in \{2, 4, 8\}$, ReAT outperforms OAT on LIBERO and RoboCasa, with the largest LIBERO gap at four tokens (66.40% versus 50.96%). MetaWorld OAT performs better at two tokens (27.00% versus 19.00%), so the full-budget advantage does not extend to every short-prefix setting.

Deployment cost. Fewer tokens reduce autoregressive generation steps, but ReAT also incurs recurrent decoding cost. We measure end-to-end inference latency for all methods under the same protocol on a single NVIDIA RTX 6000D at batch size one, including observation encoding, token generation, and action decoding. Table 3 reports the results. Eight-token ReAT takes 22.2–25.4 ms versus OAT's 18.2–22.6 ms; reducing ReAT to two tokens lowers latency to 7.6–11.0 ms without shortening the action horizon. Together with Table 2, these results characterize the control–latency trade-off rather than a uniform speed advantage. Appendix C details the timing protocol.

Reconstruction quality. To determine whether additional tokens improve reconstruction independently of policy prediction, we encode held-out demonstration actions and decode their prefixes. Figure 5 compares LIBERO reconstruction MSE against token length. MSE is measured in the original action coordinates rather than the cumulative representation used internally, averaging first over chunks within each task and then equally across tasks. Appendix D provides per-dimension R^2 and token counts.

Increasing ReAT's budget from two to eight tokens reduces MSE from 0.009491 to 0.001763, reaching 20.1% below OAT at eight tokens. FAST and Bin obtain lower MSE with longer sequences but

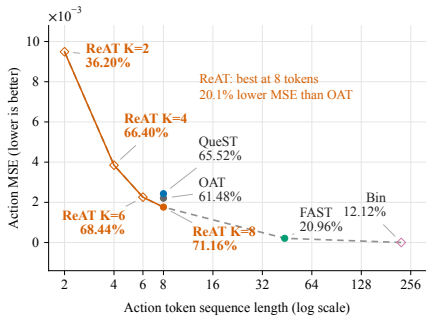

Figure 5: **LIBERO reconstruction error versus token length.** Raw-action MSE on validation chunks; percentages give policy success. Dashed links: observed Pareto front. FAST uses mean token length.

Table 4: **Real-world bucket stacking.** Twenty trials per configuration.

Method	Successes / trials	Success (%)
DP	0/20	0
Bin	0/20	0
FAST	0/20	0
QueST_8	7/20	35
OAT_8	8/20	40
ReAT_2	6/20	30
ReAT_4	5/20	25
ReAT_6	12/20	60
ReAT_8	14/20	**70**

lower policy success (Table 2), showing that reconstruction fidelity alone does not determine control performance.

5.3 Real-World Bucket Stacking

To test whether the representation supports physical manipulation, we evaluate bucket stacking on an X-Trainer robotic arm (Figure 6). Policies are trained on 90 demonstration episodes using a head-mounted camera and one wrist-mounted camera. Bucket positions are randomized before each autonomous trial. Success requires the buckets to remain stacked after gripper release within 30 seconds, without human intervention. We evaluate ReAT at two, four, six, and eight tokens, and OAT and QueST at eight tokens; Table 4 reports 20 trials per configuration.

Eight-token ReAT achieves 70% success, compared with 40% for OAT and 35% for QueST. Larger budgets generally help, although success is not monotonic across budgets.

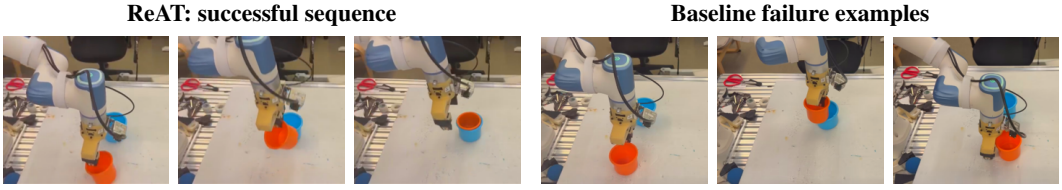

Figure 6: **Real-world bucket stacking.** Left: a successful ReAT sequence on the X-Trainer arm. Right: failure examples in which DP closes its gripper prematurely and descends into the bucket without grasping it, OAT grasps the bucket but misaligns it during placement, and QueST fails to align the gripper for a successful grasp.

Additional observations show that Bin repeatedly approaches the bucket rim without aligning the grasp. FAST encounters decoding failures when policy-generated token sequences yield an incorrect number of action coefficients. A held-out validation check confirms this failure mode in policy generation, while tokens encoded from expert actions decode successfully (Appendix B.4).

5.4 Ablation Studies

Mechanism ablations. The main comparison evaluates the complete design; to examine the roles of feedback, action coordinates, and prefix supervision, we ablate these components on LIBERO. Table 6 reports success under the main training protocol and batch size. Each tokenizer and policy is trained from scratch and evaluated across prefix budgets.

Table 5: **FSQ configuration sensitivity.** LIBERO success (%) ± SEM at $K = 8$.

FSQ levels	Codes	Success
$[8, 6, 5]$	240	61.76 ± 0.55
$[8, 8, 8]$	512	65.16 ± 0.95
$[8, 5, 5, 5]$	1000	$\mathbf{71.16} \pm 0.68$
$[8, 8, 6, 5]$	1920	68.28 ± 0.88
$[7, 5, 5, 5, 5]$	4375	66.80 ± 0.46

Table 6: **LIBERO mechanism ablations.** Success (%) ± SEM; each variant uses the same policy across token budgets. Bold: best mean per budget.

K	Full	Feedback-off	Full-length only	Direct actions
2	$\mathbf{36.20} \pm 1.28$	25.32 ± 0.48	0.04 ± 0.04	33.68 ± 1.42
4	$\mathbf{66.40} \pm 0.75$	51.32 ± 0.83	1.76 ± 0.33	55.88 ± 0.50
6	$\mathbf{68.44} \pm 1.38$	58.72 ± 0.73	8.52 ± 0.69	55.44 ± 1.09
8	$\mathbf{71.16} \pm 0.68$	60.24 ± 1.02	70.80 ± 0.68	56.84 ± 1.05

Feedback-off tests the contribution of reconstruction feedback by encoding the target X once and quantizing all M register features, $(u_{1:M}, z_{1:M}) = \mathrm{FSQ}(E_\phi(X))$, without recomputing encoder inputs from intermediate drafts. It retains the encoder architecture, quantizer, draft-conditioned recurrent decoder, and tokenizer loss. Direct actions tests the cumulative representation by setting $\mathcal{I} = \varnothing$ while retaining feedback; rotation and gripper channels remain direct. Full-length only tests the need for intermediate-prefix supervision by setting random-prefix and progress-loss weights to zero and the full-length reconstruction weight to one.

Feedback-off reduces eight-token success from 71.16% to 60.24% and performs worse at every tested budget. Direct actions reaches 56.84%, supporting a separate contribution from cumulative translation; Appendix E provides a coordinate-matched OAT comparison. Full-length only retains similar eight-token performance but performs poorly at shorter prefixes, supporting the need for prefix supervision.

FSQ configuration and capacity. To test quantization capacity at a fixed eight-token budget, we compare five FSQ configurations on LIBERO (Table 5). The 1000-code configuration performs best; increasing capacity further does not improve policy success.

6 Discussion and Limitations

Reconstruction feedback gives tokens a corrective role, not a fixed motion meaning: the same symbol can produce different updates for different drafts. The ablations support both feedback and prefix supervision, while showing that reconstruction fidelity alone does not determine control performance.

Budgets are externally selected and chunks are generated before execution; learned stopping, asynchronous refinement, and inter-chunk continuity remain open directions. Output bounds do not guarantee safe motion, nor does feedback eliminate policy prediction errors. Physical evaluation covers one task with 20 trials per configuration; additional tasks and hardware platforms would clarify deployment generality.

7 Conclusion

ReAT constructs action tokens from the residual left by the decoded prefix and converts them into draft-conditioned corrections. Prefix supervision makes this representation usable at multiple token budgets with the same autoregressive policy and action horizon. Experiments across three simulation benchmarks and real-world bucket stacking demonstrate its control performance, while LIBERO ablations support reconstruction feedback and prefix supervision. More broadly, these findings highlight explicit residual correction as a design principle for action tokenization, extending the goal beyond compact encoding to structuring how successive tokens contribute to action reconstruction.

Ethics Statement

This work uses existing robot demonstration datasets, simulation, and a physical bucket-stacking task, with no new human-subject study. Physical deployment requires robot-specific safety measures beyond bounded action outputs. Dataset reuse is subject to the original licenses and attribution requirements.

Reproducibility Statement

Sections 4 and 5 specify the method, objectives, and evaluation protocol. Algorithm 1 summarizes token construction; Appendix A provides architecture and optimization settings, exact update equations, and label-construction details. Appendices B and B.3 describe the data interfaces and uncertainty estimator; Appendix C records the timing protocol and model provenance.

Statement on the Use of AI

Generative AI tools, including OpenAI Codex, were used to polish writing, retrieve and discover related literature, assist with research ideation and execution (experiment planning and monitoring), and draft sections of the manuscript.

References

Roman Bachmann, Jesse Allardice, David Mizrahi, Enrico Fini, Oğuzhan Fatih Kar, Elmira Amirloo, Alaaeldin El-Nouby, Amir Zamir, and Afshin Dehghan. FlexTok: Resampling images into 1D token sequences of flexible length. *arXiv preprint arXiv:2502.13967*, 2025. URL https://arxiv.org/abs/2502.13967.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. URL https://arxiv.org/abs/2410.24164.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. URL https://arxiv.org/abs/2307.15818.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin CM Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://roboticsproceedings.org/rss19/p026.html.

Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv preprint arXiv:2210.13438*, 2022. URL https://arxiv.org/abs/2210.13438.

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.

Zhefei Gong, Pengxiang Ding, Shangke Lyu, Siteng Huang, Mingyang Sun, Wei Zhao, Zhaoxin Fan, and Donglin Wang. CARP: Visuomotor policy learning via coarse-to-fine autoregressive prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 13460–13470, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Gong_CARP_Visuomotor_Policy_Learning_via_Coarse-to-Fine_Autoregressive_Prediction_ICCV_2025_paper.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246.

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

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11523–11532, 2022. URL https://arxiv.org/abs/2203.01941.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *International Conference on Machine Learning*, pp. 26991–27008, 2024. URL https://proceedings.mlr.press/v235/lee24y.html.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, 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*, 2026. URL https://arxiv.org/abs/2602.04215.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.15505.

Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. QueST: Self-supervised skill abstractions for learning continuous control. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2407.15840.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of household tasks for generalist robots. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.050. URL https://www.roboticsproceedings.org/rss20/p050.html.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. URL https://arxiv.org/abs/2503.14734.

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/2501.09747.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, 2014. URL https://arxiv.org/abs/1402.0915.

George Toderici, Sean M. O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable rate image compression with recurrent neural networks. In *International Conference on Learning Representations*, 2016a. URL https://arxiv.org/abs/1511.06085.

George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. *arXiv preprint arXiv:1608.05148*, 2016b. URL https://arxiv.org/abs/1608.05148.

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. *arXiv preprint arXiv:2507.01016*, 2025. URL https://arxiv.org/abs/2507.01016.

Daichi Yashima, Koki Seno, Shuhei Kurita, Yusuke Oda, and Komei Sugiura. HiFlow: Tokenization-free scale-wise autoregressive policy learning via flow matching. *arXiv preprint arXiv:2603.27281*, 2026. URL https://arxiv.org/abs/2603.27281.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100, 2020. URL https://proceedings.mlr.press/v100/yu20a.html.

Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. SoundStream: An end-to-end neural audio codec. *arXiv preprint arXiv:2107.03312*, 2021. URL https://arxiv.org/abs/2107.03312.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems*, 2023. URL https://arxiv.org/abs/2304.13705.

A Implementation Details

A.1 Architecture and Optimization

Table 7 records the main ReAT configuration.

Table 7: **Main ReAT settings.** EMA is enabled in both stages.

Setting	Value
Observation / prediction / execution horizon	2 / 32 / 16
Maximum content tokens / vocabulary	8 / 1000
FSQ scalar levels	$[8, 5, 5, 5]$
Register encoder	width 256, head dimension 64
Edit decoder	width 256, head dimension 64
Autoregressive policy	4 layers, width 256, 4 heads
Encoder / decoder / policy dropout	0.1 / 0.1 / 0.1
Proposal scales α / α_g	8 / 4
Full / random-prefix reconstruction weight	0 / 1; uniform lengths 1–8
Progress / control MSE weight λ_p / λ_g	0.2 / 0.1
Locality / edit weight λ_{loc} / λ_{edit}	10^{-3} / 10^{-4}
Batch size, each training stage	256
Optimizer	AdamW, $\beta = (0.9, 0.95)$, no weight decay
Tokenizer / policy / observation learning rate	5×10^{-5} / 5×10^{-5} / 10^{-5}
Schedule / warmup / gradient norm cap	Constant / 100 steps / 1.0
Frozen teacher label precision / mode	FP32 / evaluation mode
Policy and visual encoder training	BF16 mixed precision

Tokenizer dropout is active during training. The intermediate reconstructions used to define encoder residuals also supply the prefix losses, and gradients flow through the recurrent updates and straight-through quantizer.

Visual encoder. Each camera uses a separate ResNet-18 convolutional backbone initialized without pretrained weights. Batch normalization is replaced by GroupNorm, and spatial softmax pooling with 32 keypoints produces a 64-dimensional feature per camera. The visual encoder is optimized jointly with the policy at learning rate 10^{-5}; it is not frozen. Appendix B.2 describes image cropping and evaluation preprocessing.

Baseline configurations. The simulation policies use four-layer, width-256 action Transformers with four attention heads. We use the tokenizer architectures described in the OAT and QueST papers (Liu et al., 2026; Mete et al., 2024), trained under the protocol described in Appendix A.2. FAST fits a benchmark-specific byte-pair vocabulary of 1024 entries with DCT quantization scale 10. Bin uses scalar binning configured with 256 bins over normalized actions. DP uses a cosine noise schedule with 100 training timesteps and predicts noise, with 10 DDIM steps at inference.

A.2 Training Duration and Checkpoint Evaluation

Tokenizer training stops at a stage boundary when the minimum validation loss decreases by less than 0.5% between two consecutive 100-epoch windows. ReAT, OAT and QueST use validation reconstruction MSE. The checkpoint minimizing the corresponding criterion is frozen before policy training. FAST fits its vocabulary without gradient-based tokenizer training, Bin uses fixed scalar bins, and DP has no tokenizer-training stage. Policies are trained for 500 epochs, and checkpoints are selected by the lowest validation loss: cross-entropy for autoregressive policies and denoising MSE for DP. Evaluation follows Section 5.1.

A.3 Action Coordinates and Recurrent State

Let $\widetilde{A} = N(A)$ be the expert chunk after per-channel affine normalization. For each action channel j, training-episode extrema $a_j^{\min}$ and $a_j^{\max}$ define

$$N(A)_{t,j} = 2\frac{A_{t,j} - a_j^{\min}}{a_j^{\max} - a_j^{\min}} - 1.$$

The same statistics are retained for validation and deployment. Channels with range below 10^{-4} use unit scale and offset $-a_j^{\min}$ to avoid division by a near-zero range. The inverse affine map restores the benchmark action units after decoding.

We partition normalized action dimensions into motion channels $\mathcal{C}$ and separately updated control channels $\mathcal{G}$, including gripper commands. Translation channels $\mathcal{I} \subseteq \mathcal{C}$ use cumulative sums; rotation channels retain per-step values. The transform $C_{\mathcal{I}}$ and its inverse $B_{\mathcal{I}}$ are

$$(C_{\mathcal{I}}x)_{t,j} = \begin{cases} \sum_{s=1}^{t} x_{s,j}, & j \in \mathcal{I}, \\ x_{t,j}, & j \notin \mathcal{I}, \end{cases} \qquad (B_{\mathcal{I}}q)_{t,j} = \begin{cases} q_{t,j} - q_{t-1,j}, & j \in \mathcal{I}, \\ q_{t,j}, & j \notin \mathcal{I}, \end{cases} \tag{4}$$

with $q_{0,j} = 0$, where t indexes action steps and j indexes channels. The motion target is $Q = C_{\mathcal{I}}\widetilde{A}_{\mathcal{C}}$, and the control target is $g = \widetilde{A}_{\mathcal{G}}$. Cumulative sums restart at zero per chunk and represent normalized commands, not physical poses. Appendix B specifies the channel groups.

The recurrent state comprises a motion draft $S^{(k)}$ and control preactivations $L^{(k)}$. After k tokens, $S^{(k)}$ reconstructs Q, while $\tanh L^{(k)}$ reconstructs g. Control corrections accumulate in $L^{(k)}$ before $\tanh$ bounds their output. Both states start at zero. The target and visible draft used in Equation 1 are

$$X = \mathrm{Concat}(Q, g), \qquad X_k = \mathrm{Concat}\Big(S^{(k)}, \tanh L^{(k)}\Big). \tag{5}$$

Here Concat joins channels in their original order, not along time; $X, X_k \in \mathbb{R}^{H\times d}$ and $X_0 = 0$. The decoder retains $L^{(k)}$ in its recurrent state even though its visible draft contains the bounded value $\tanh L^{(k)}$.

A.4 Encoder Features and Bounded Decoder Updates

The Transformer encoder E_ϕ produces M register feature vectors from each residual R_k. Only vector k is quantized at stage k, yielding the index z_k and embedding u_k in Equation 1. The encoder is rerun on the updated residual at each stage.

For action step t and token stage k, the decoder combines the draft row $X_{k-1,t}$ with u_k and positional features:

$$h_{k,t} = p_t^{\mathrm{time}} + W_s X_{k-1,t} + W_u u_k + p_k^{\mathrm{tok}} + s_k, \qquad r_k = \tanh(W_r F_\theta(h_k)). \tag{6}$$

Here p_t^{time} and p_k^{tok} encode action and token positions, s_k is a learned stage embedding, and h_k collects all H input vectors. The Transformer F_θ operates across action positions; the W matrices are learned projections. Its bounded proposal $r_k \in \mathbb{R}^{H\times d}$ covers all action steps.

For motion channels, we add the proposal, convert the temporary draft $\overline{S}^{(k)}$ to per-step commands, bound them with $\tanh$, and transform back. Control channels accumulate corrections before $\tanh$. The normalized predictions $\widehat{a}^{(k)}$ are

$$\overline{S}^{(k)} = S^{(k-1)} + \alpha r_{k,\mathcal{C}}, \qquad \widehat{a}_{\mathcal{C}}^{(k)} = \tanh\Big(B_{\mathcal{I}}\overline{S}^{(k)}\Big), \qquad S^{(k)} = C_{\mathcal{I}}\widehat{a}_{\mathcal{C}}^{(k)}, \tag{7}$$

$$L^{(k)} = L^{(k-1)} + \alpha_g r_{k,\mathcal{G}}, \qquad \widehat{a}_{\mathcal{G}}^{(k)} = \tanh L^{(k)}, \tag{8}$$

where α and α_g scale the proposals. Undoing normalization gives $\widehat{A}^{(k)} = N^{-1}(\widehat{a}^{(k)})$. Deployment also applies empirical action bounds and benchmark-controller clipping. Because of bounding, the actual draft change $X_k - X_{k-1}$ need not equal the scaled proposal. Retaining $(S^{(K)}, L^{(K)})$ permits further decoding without repeating the first K updates.

A.5 Expanded Tokenizer Objective

The motion and control reconstruction errors at prefix k are

$$e_k = \frac{\left\| \widehat{a}_{\mathcal{C}}^{(k)} - \widetilde{A}_{\mathcal{C}} \right\|_F^2}{H|\mathcal{C}|}, \qquad g_k = \frac{\left\| \tanh L^{(k)} - \widetilde{A}_{\mathcal{G}} \right\|_F^2}{H|\mathcal{G}|}. \tag{9}$$

Both are MSEs in normalized per-step action coordinates; $\|\cdot\|_F$ denotes the Frobenius norm. The reconstruction term in Equation 2 is $\ell_{\mathrm{rec}}(\widehat{A}^{(k)}, A) = e_k + \lambda_g g_k$. For $J \sim \mathrm{Uniform}\{1, \dots, M\}$, the expanded objective is

$$\begin{aligned} \mathcal{L}_{\mathrm{tok}} = \mathbb{E}_J[e_J + \lambda_g g_J] + \frac{\lambda_p}{M-1} \sum_{k=1}^{M-1} \left([e_{k+1} - e_k]_+ + \lambda_g [g_{k+1} - g_k]_+ \right) \\ + \lambda_{\mathrm{loc}} \mathcal{L}_{\mathrm{loc}} + \lambda_{\mathrm{edit}} \mathcal{L}_{\mathrm{edit}}. \end{aligned} \tag{10}$$

Training estimates the expectation by uniform prefix sampling. The progress term $\mathcal{L}_{\mathrm{progress}}$ is the average of the bracketed penalties, with $[x]_+ = \max(x, 0)$; increases are penalized separately for motion and control errors. The auxiliary term is $\mathcal{L}_{\mathrm{reg}} = \lambda_{\mathrm{loc}} \mathcal{L}_{\mathrm{loc}} + \lambda_{\mathrm{edit}} \mathcal{L}_{\mathrm{edit}}$. Table 7 gives all weights.

Applied-correction regularization. Let δ_k concatenate $S^{(k)} - S^{(k-1)}$ and $\tanh L^{(k)} - \tanh L^{(k-1)}$. For $k \geq 2$, define $w_{k,t} = \sqrt{\|\delta_{k,t}\|_2^2 + 10^{-6}}$. The locality term averages $(\sum_t w_{k,t})^2 / (H \sum_t w_{k,t}^2)$, and the edit term averages squared elements of δ_k. These regularizers encourage moderate, concentrated corrections without fixing their temporal support. They operate after the bounding projection; because of this projection, a zero proposal is not generally an identity update.

A.6 Token Construction and Deployment Details

In Algorithm 1, the input X is the normalized, coordinate-transformed expert chunk defined in Appendix A.3. The internal decoder state is $(S^{(k)}, L^{(k)})$, and D_θ denotes the edit proposal in Equation 6. The Update operation applies Equations 7–8, forms X_k, and undoes normalization to produce $\widehat{A}^{(k)}$. Tokenizer training differentiates through this recurrent computation with straight-through quantization and the prefix objective in Equation 2. Policy-label construction instead runs the frozen encoder, quantizer, and decoder in FP32 with dropout disabled.

At deployment, a beginning-of-sequence symbol initializes the policy, which generates K content tokens while masking BOS and STOP from its output vocabulary. Each integer code maps to its FSQ embedding; the decoder applies the same proposal and update operations from a zero-initialized state, without expert actions or residual encoding. The reconstructed chunk is clamped to the empirical and benchmark-controller action bounds. The controller executes its first 16 actions, obtains new observations, and repeats inference.

B Benchmark and Evaluation Details

B.1 Tasks and Action Interfaces

LIBERO uses the original LIBERO-10 task suite and its seven-dimensional action interface. MetaWorld uses Box Close, Coffee Pull, Disassemble, and Stick Pull. RoboCasa uses Close Drawer, Coffee Press Button, Turn Off Microwave, and Turn Off Sink Faucet.

The RoboCasa dataset combines 50 human demonstrations and 150 machine-generated demonstrations per task. We evaluate in RoboCasa v0.2 using its twelve-dimensional action interface, with dataset action channels reordered to the simulator controller's expected order.

The separately updated control indices $\mathcal{G}$ (zero-based) are $[6]$ for LIBERO, $[3]$ for MetaWorld, and $[4, 11]$ for RoboCasa. The integrated translation indices $\mathcal{I}$ are $[0, 1, 2]$ for LIBERO and MetaWorld,

and $[5, 6, 7]$ for RoboCasa in dataset order. Rotation and all remaining channels retain direct per-step action coordinates. All twelve RoboCasa dimensions are retained before the controller permutation.

The main method uses per-channel affine action normalization and the action representation defined in Appendix A.3.

B.2 Observation Processing and Reset Handling

All compared methods use the same benchmark-specific camera and proprioceptive inputs. Simulation images are randomly cropped to 76×76 during training and center-cropped to 76×76 during validation and rollout evaluation before ResNet encoding. For environment initialization, the Meta-World evaluator clears residual bottle velocity after reset, while LIBERO lets the scene settle before acquiring the initial observation.

OAT's short-prefix evaluations use the same policy and tokenizer as its eight-token result within each benchmark, without retraining or prefix-specific model selection. LIBERO, MetaWorld and RoboCasa use center cropping and the same evaluation seed set as ReAT. Each added budget uses five rounds of 50 episodes per task; the eight-token results are retained from the existing evaluations.

B.3 Statistical Reporting

Let $Y_{t,r,i} \in \{0, 1\}$ denote success for task t, evaluation round $r \in \{1, \ldots, 5\}$, and episode $i \in \{1, \ldots, 50\}$. We first average episodes and tasks within each round, then average the five round-level means:

$$\widehat{p}_r = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{1}{50} \sum_{i=1}^{50} Y_{t,r,i} \right), \qquad \widehat{p} = \frac{1}{5} \sum_{r=1}^{5} \widehat{p}_r. \tag{11}$$

Tables 2 and 6 report $\widehat{p}$ and the standard error across round-level means:

$$s_{\text{round}} = \sqrt{\frac{1}{4} \sum_{r=1}^{5} (\widehat{p}_r - \widehat{p})^2}, \qquad \text{SEM} = \frac{s_{\text{round}}}{\sqrt{5}}. \tag{12}$$

Both $\widehat{p}$ and SEM are multiplied by 100 for percentage reporting.

B.4 FAST Decoding Validity on Bucket Stacking

We examine all 300 validation windows from nine held-out bucket-stacking demonstrations using the exported FAST policy and tokenizer. The check verifies that the exported policy weights match the saved EMA weights. For each window, we separately decode tokens obtained by encoding the expert action chunk and tokens freely generated by the observation-conditioned policy, without supplying expert token prefixes. This FAST configuration predicts 16 steps with seven action dimensions, requiring exactly 112 DCT coefficients after token decoding.

All 300 expert-action token sequences decode successfully. In contrast, policy generation with top-$k = 10$ and temperature 1 produces 263/300 invalid sequences (87.67%); greedy generation produces 251/300 (83.67%). Every invalid sequence has an incorrect coefficient count and cannot be reshaped into the required action chunk. These results identify a failure of policy-generated sequence validity in this check, rather than a failure to decode tokens encoded from expert actions. The percentages describe offline validation chunks, separately from the physical trial success rates in Table 4.

C Inference-Latency Measurement Protocol

We measure synchronized host-wall time on one RTX 6000D with batch size one, gradients disabled, and FP32 parameters (PyTorch 2.10.0, CUDA 12.8). Each benchmark cycles through 20 GPU-resident observation windows. Each of five repetitions consists of 50 warmup calls followed by 100 timed calls. Table 3 reports the mean across all timed calls. Timing includes observation encoding, token generation, action decoding, and final synchronization; it excludes simulator execution, RPC,

and external data loading. Jobs run serially with one CPU thread pinned to a single core and GPU isolation checked during measurement.

All methods are timed under eager execution. ReAT retains fixed-prefix decoding, cached static features, and final-prefix-only output computation: it executes only the requested K stages without changing the decoded prefix.

Native action backbones have four layers, width 256, and four heads. DP uses 10 DDIM steps; Bin uses 256 bins per scalar; autoregressive baselines use KV caching, top-$k = 10$, and temperature 1. The ReAT timing variants share parameters and differ only in the requested token budget. FAST timings replay valid demonstration codes through the autoregressive generation path and include CPU action decoding, so its token counts are demonstration-conditioned rather than policy-predicted stopping lengths.

OAT's short-prefix measurements generate exactly $K \in \{2, 4, 6\}$ tokens and decode a complete 32-step chunk, replacing the unused latent positions with the learned dropout mask. Each OAT budget uses the same five-repeat protocol and physical GPU as the other rows.

D Per-Dimension Tokenizer Reconstruction

Figure 5 reports LIBERO reconstruction MSE, which depends on channel scale. To compare translation, rotation, and gripper quality, Table 8 reports per-dimension R^2 on the same validation split, normalized by target variance within each task and dimension. For task t and dimension j, we compute

$$R^2_{t,j} = 1 - \frac{\sum_i (a_{t,i,j} - \widehat{a}_{t,i,j})^2}{\sum_i (a_{t,i,j} - \bar{a}_{t,j})^2}, \qquad \bar{a}_{t,j} = \frac{1}{n_t} \sum_i a_{t,i,j}, \tag{13}$$

Here i indexes action steps in held-out chunks, n_t is their count, and $\widehat{a}$ comes from expert-action tokens rather than policy predictions. $R^2 = 1$ denotes exact reconstruction; 0 matches the task-specific channel mean; negative values are worse. We average equally across the ten tasks and seven dimensions. Every task–dimension pair has nonzero target variance.

Table 8: **LIBERO tokenizer reconstruction by action dimension.** R^2 values are task-averaged; MSE is measured in original action coordinates. $\bar{K}$ counts content tokens.

Action dimension / summary	ReAT	OAT	QueST	FAST	Bin
Translation x	0.94889	0.93819	0.94711	0.99463	0.99994
Translation y	0.96745	0.95924	0.96496	0.99662	0.99996
Translation z	0.96882	0.95807	0.96940	0.99708	0.99997
Rotation x	0.78624	0.73519	0.35841	0.97437	0.99967
Rotation y	0.88375	0.83200	0.53693	0.98556	0.99982
Rotation z	0.89567	0.85831	0.68065	0.98659	0.99971
Gripper	0.99976	0.99881	0.99619	0.99963	0.99998
Mean $R^2 \uparrow$	0.92151	0.89712	0.77909	0.99064	0.99986
Action MSE $\downarrow$	0.001763	0.002207	0.002430	0.000211	0.000004
Mean tokens $\bar{K}$	8	8	8	43.75	224
Token efficiency $\uparrow$	0.11519	0.11214	0.09739	0.02265	0.00446

At eight tokens, ReAT exceeds OAT in every dimension, most strongly in rotation. FAST and Bin achieve higher mean R^2 with longer sequences. We define *token efficiency* as $E_{\text{tok}} = \overline{R^2}/\bar{K}$, mean R^2 divided by mean content-token count. FAST's $\bar{K}$ is averaged over validation chunks. This descriptive ratio favors shorter sequences and should be read alongside absolute quality and sequence length; it does not measure inference speed.

E OAT with Matched Action Coordinates

To examine whether the action-coordinate representation accounts for ReAT's advantage over OAT, we train an additional OAT tokenizer and policy on LIBERO using the same cumulative translation coordinates as ReAT. Normalized translation commands are cumulatively summed within

each chunk, while rotation and gripper channels retain per-step values. Decoded translations are differenced back into per-step commands before computing OAT's reconstruction loss and before execution. The coordinate transform adds no parameters; OAT's encoder and decoder architectures, quantizer, and nested-dropout scheme are retained.

Table 9 compares all three configurations at eight tokens. The coordinate-matched OAT achieves 62.56% success, compared with 61.48% for the original OAT and 71.16% for ReAT. The remaining 8.60-percentage-point gap indicates that matching translation coordinates alone does not account for ReAT's advantage in this comparison.

Table 9: **LIBERO comparison with matched action coordinates.** Success (%) at $K = 8$, reported as mean $\pm$ SEM across five evaluation rounds. Rotation and gripper channels remain per-step in every row.

Method	Translation coordinates	Success
OAT	Per-step	61.48 ± 0.92
OAT	Cumulative	62.56 ± 0.67
ReAT	Cumulative	$\mathbf{71.16} \pm 0.68$