Ordered Action Tokens for Visuomotor Policy Learning

Chaoqi Liu[1,2] Yue Zhao[2] Haonan Chen[1,2] Xiaoshen Han[1] Jiawei Gao[1]
Ehsan Adeli[2] Yilun Du[1]

[1]Harvard University [2]Stanford University

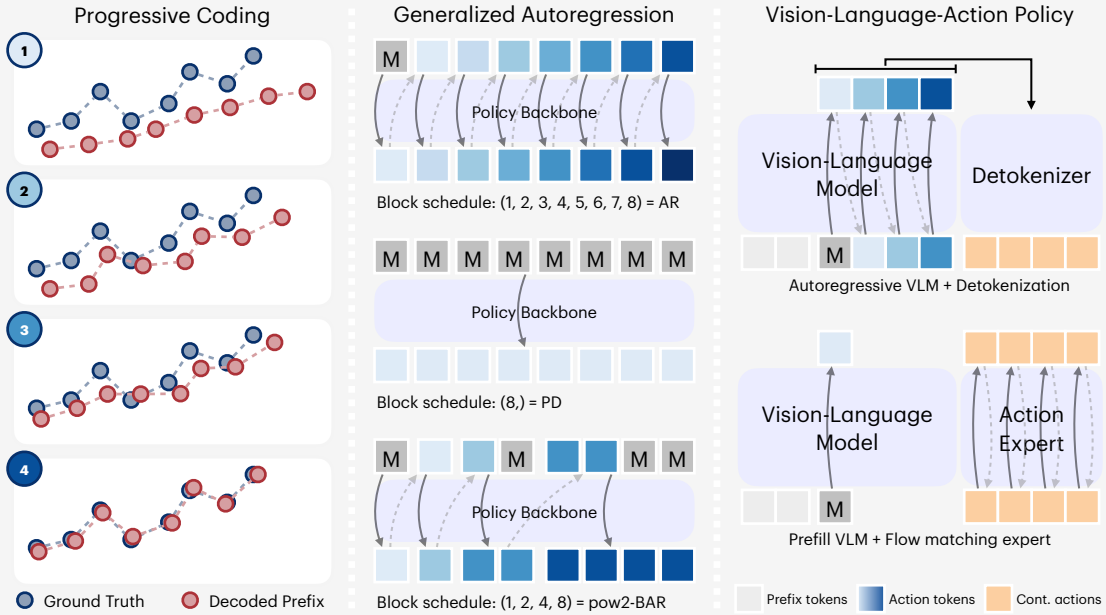


Figure 1: OAT tokens for visuomotor policy learning. Left: OAT encodes each action chunk as an ordered token sequence whose prefixes decode to plausible action chunks, with later tokens refining control. **Middle:** In autoregressive policies, OAT tokens can be block-wise generated under schedules spanning token-wise autoregression (**top**), fixed-size blocks, power-of-two grouping (**middle**), and parallel decoding (**bottom**). **Right:** We validate OAT in two prevailing uses of tokens: autoregressive policies that generate tokens decoded into control (**top**), and token co-training policies, where tokens supervise the vision-language model while a flow-matching expert generates actions from the model's prefill context (**bottom**).

Website: ordered-action-tokenization.github.io
Code: Chaoqi-LIU/oat; Chaoqi-LIU/praxis-vla

Abstract

Action tokenization maps continuous robot action chunks to discrete tokens and has become an important interface for modern visuomotor policies. Existing approaches either rely on analytical discretization methods that produce prohibitively long token sequences or learned latent tokenizers that lack structure, limiting their compatibility with downstream policies. In this work, we identify three desiderata for action tokenization – high compression, total decodability, and an ordered token space – and introduce Ordered Action Tokenization (OAT), a learned action tokenizer that satisfies all three. OAT discretizes action chunks into an ordered sequence of tokens using a transformer with registers, finite scalar quantization, and ordering-inducing training mechanisms. By training each token prefix to decode into a valid action chunk, OAT

places coarse control information in early tokens and uses later tokens to refine residual detail, yielding an anytime tradeoff between inference cost and action fidelity. We validate OAT in two prevailing uses of action tokens: autoregressive policies that generate tokens for control, and token co-training policies that use token losses to shape the vision-language model context consumed by a flow-based action expert. Across three policy backbones and more than 60 tasks spanning five simulation benchmarks and real-world settings, OAT consistently improves policy performance while offering significantly greater flexibility at inference time.

1 Introduction

Action tokens are the interface between continuous robot control and sequence modeling, yet the design of this interface remains under-examined. Experience from language and vision shows that tokenization shapes learning dynamics, model capacity, scalability, and downstream performance (Sennrich et al., 2016; van den Oord et al., 2017; Mentzer et al., 2024; Bachmann et al., 2025). Likewise, in robot control, action tokenization is not merely offline compression: tokens determine the output length, the validity of arbitrary policy samples, and how control-relevant information is organized for prediction or supervision. Therefore, tokens must be designed with the policy interface in mind, so that they are easy for policies to predict or learn from. This raises a basic question: what properties should an action tokenizer satisfy to serve as a policy interface?

In this paper, we study *what constitutes good action tokens* and argue that an effective action tokenizer must simultaneously satisfy three key desiderata: high compression, total decodability, and ordered token structure. **(1) High compression** keeps token sequences compact. **(2) Total decodability** ensures that arbitrary policy outputs map to executable actions when tokens are decoded. **(3) Ordered structure** places control-relevant information early, implicitly forming a coarse-to-fine representation.

Prior action tokenization methods satisfy subsets of these desiderata, but not all simultaneously. Per-dimension binning (`Bin`) is simple and reliably decodable, but it serializes every action dimension at every step, producing long token sequences as action dimension and prediction horizon grow (Brohan et al., 2023; Zitkovich et al., 2023; Kim et al., 2024). Frequency-domain tokenizers such as `FAST` introduce a useful low-to-high-frequency order, but byte-pair encoding compression makes detokenization only partially defined: an unconstrained policy sample is not guaranteed to expand into the fixed-shape coefficient array required for control (Gage, 1994; Sennrich et al., 2016; Pertsch et al., 2025). Learned latent tokenizers such as `QueST` and `ACodec` compress action chunks through discrete bottlenecks (Mete et al., 2024; Lee et al., 2024; Dong et al., 2026), but the reconstruction quality they optimize for does not necessarily improve closed-loop policy rollout success at inference.

To bridge this gap, we propose Ordered Action Tokenization (OAT), a learned tokenizer that discretizes continuous action chunks into compact, totally decodable, and ordered token sequences. OAT employs transformer-based register tokens to aggregate temporal information, finite scalar quantization to construct a discrete bottleneck, and nested dropout to train prefixes at multiple budgets to decode into plausible action chunks. This prefix training implicitly induces a coarse-to-fine structure: early tokens capture high-impact control information, while later tokens refine residual detail.

We validate the effectiveness of OAT in two prevailing uses of action tokens. In autoregressive policies, OAT tokens are generated and detokenized into actions; their ordering provides an inductive bias aligned with next-token prediction and supports variable inference budgets through prefix-based decoding. In token co-training policies, action token losses supervise the vision-language model (VLM), whose context conditions a flow-matching action expert at inference. Because OAT trains its one-token prefix to reconstruct the complete action chunk, predicting the first target directly

Figure 2: Prefix reconstruction on an iiwa arm. Columns show action chunk reconstructions decoded from the first 1, 2, 4, and 8 OAT tokens, followed by the ground-truth action chunk. Increasing the prefix budget progressively refines the reconstructed trajectory while every mask-padded prefix detokenizes to an executable action chunk. See the interactive prefix lab on the project website.

from the VLM prefill places a plan-like, action-chunk-level objective on the representation consumed by the expert. Sec. 2 gives the background for both policy roles.

For scalable autoregressive inference with OAT, we further introduce a scheduling framework for block autoregressive decoding that unifies token-wise autoregression, one-shot parallel decoding, fixed-size block prediction (Liu et al., 2026; Dong et al., 2026), and intermediate schemes. We focus on two variants in this paper: $\texttt{OAT}^{\text{sing}}$ for token-wise autoregression and $\texttt{OAT}^{\text{pow2}}$ for power-of-two block decoding, where the latter reduces inference complexity from linear to logarithmic.

Contributions. In summary, this paper makes three contributions, as illustrated in Fig. 1:

1. We analyze representative action tokenizers as policy interfaces and formalize three desiderata for visuomotor policy learning: high compression, total decodability, and ordered structure.
2. We propose OAT, a learned action tokenizer that satisfies these desiderata with compact, totally decodable, ordered tokens whose prefixes decode to executable action chunks, and further introduce a framework for scalable block autoregressive decoding.
3. We conduct extensive experiments and ablations showing that OAT is effective across lightweight policies and vision-language-action (VLA)-scale systems, covering both autoregressive (AR) policies that generate action tokens and token co-training (TC) policies that use action tokens as supervision.

2 Preliminaries

We first define notation and background used throughout the paper.

Action chunks and tokens. Robot policies commonly execute control through short chunks of continuous actions. We write one action chunk as

$$A = a_{1:H_a} \in \mathbb{R}^{H_a \times D_a},$$

where H_a is the action horizon and D_a is the action dimension. Tokenized autoregressive policies expose this continuous chunk as a discrete sequence of policy symbols (Brohan et al., 2023; Zitkovich et al., 2023; Kim et al., 2024; Bonatti et al., 2022; Fu et al., 2025). For a tokenizer setting with token horizon H_l, the action tokenizer is

$$\mathcal{T} : a_{1:H_a} \mapsto T_{1:H_l}, \qquad T_i \in \mathcal{V}.$$

Here $\mathcal{V}$ is the action token vocabulary. A corresponding detokenizer, written as $\mathcal{T}^{-1}$ for notation, maps a token sequence back to a continuous action chunk,

$$\mathcal{T}^{-1} : T_{1:H_l} \mapsto \hat{a}_{1:H_a}.$$

Token-wise autoregressive policies. Let $o_{1:H_o}$ denote the observation history available to the policy, with observation horizon H_o. The standard token-wise autoregressive policy models the action token sequence left to right:

$$p_\pi(T_{1:H_l} \mid o_{1:H_o}) = \prod_{i=1}^{H_l} p_\pi(T_i \mid T_{<i}, o_{1:H_o}).$$

After sampling $T_{1:H_l}$, the policy detokenizes it with $\mathcal{T}^{-1}$ and executes the resulting chunk, typically using receding-horizon control (Zhao et al., 2023; Chi et al., 2025; Zhang et al., 2025). Sec. 5.1 introduces block autoregressive decoding, which keeps this tokenizer–detokenizer interface but groups token positions during generation instead of generating one position at a time.

Token co-training policies. We use *token co-training* to denote a VLA training setup that separates token-based VLM supervision from continuous action prediction (Driess et al., 2025; Black et al., 2025; Fang et al., 2026; Dong et al., 2026). During training, an action token loss supervises the VLM, while a flow-matching loss trains a separate action expert conditioned on detached VLM context (Driess et al., 2025; Fang et al., 2026; Dong et al., 2026). Its objective has the schematic form

$$\mathcal{L}_{\text{TC}} = \mathcal{L}_{\text{tok}} + \lambda \mathcal{L}_{\text{flow}}.$$

At inference, the VLM is prefilled once, its action token logits are discarded, and the expert generates continuous action chunks from the cached context. Action tokens therefore serve as supervision targets rather than the representation executed by the controller. Sec. 5.2 and Appendix B provide details about this paradigm.

3 Action Tokenization as a Policy Interface

Following the notation in Sec. 2, this section analyzes what makes an action tokenizer a useful policy interface. The tokenizer determines the discrete targets presented to the policy, how many targets it must model, whether arbitrary generated sequences can be detokenized into valid actions, and how token structure interacts with the policy objective.

3.1 Tokenizer Desiderata

Rate and distortion are a general lens for lossy compression (Shannon, 1948; Blau and Michaeli, 2019) and are widely used to analyze learned discrete representations (Tschannen et al., 2018; van den Oord et al., 2017; Mentzer et al., 2024; Bachmann et al., 2025; Zhao et al., 2025). For action tokenization, compression remains a policy requirement because long token sequences increase the number of prediction targets and, for autoregressive policies, generation depth. Policy learning also adds requirements that compression alone does not capture: sampled tokens must decode reliably, and token order should expose control structure that helps policies generate tokens or learn from token supervision.

P.1 High compression. The token horizon H_l should be small relative to the raw action size $H_a D_a$. Long token sequences increase training difficulty and, for autoregressive policies, inference latency because the policy must model more token targets and generate more tokens. A suitable tokenizer keeps the rate low while preserving motion relevant to control.

P.2 Total decodability. The detokenizer should be a total function over the policy's discrete output space. When tokens are decoded into control, a policy can emit any token sequence supported by its output distribution. The detokenizer must therefore map arbitrary policy samples, not only training codes produced by the encoder, to valid continuous actions.

Scheme	Compact	Total decodable	Ordered structure	Block compat.	Policy implication
`Bin` [12, 82]	×	✓	×	✓	Long coordinate sequences raise inference cost and give weak conditioning structure.
`FAST` [60]	✓	×	✓	×	Unconstrained samples can decode invalidly; constrained decoding changes the policy interface.
`QueST` [54]	✓	✓	×	×	A reconstruction loss need not place control-relevant information early.
`ACodec` [20]	✓	✓	–	✓	Parallel decoding does not require serial ordering, but the full latent block is a hard joint target.
`OAT`$^{\text{sing}}$ [49]	✓	✓	✓	×	Causal register ordering supports token-wise prediction, but is not aligned with grouped block prediction.
`OAT`$^{\text{pow2}}$	✓	✓	✓	✓	Block-causal register ordering makes each token group jointly predictable.

Table 1: Action tokenizer comparison. Rows compare representative action tokenizers as policy interfaces. The first three columns mark the core desiderata: compactness, total decodability over unconstrained policy samples, and ordered token structure. The final column records block compatibility, a side property for scalable grouped autoregressive inference. A dash marks one-shot parallel decoding, where serial ordered structure is not applicable.

P.3 **Ordered token structure.** Reconstruction error alone does not guarantee useful policy targets. An ordered token structure places control-relevant information early and leaves residual detail to later tokens. For autoregressive policies, this creates learnable generation targets that align with the inductive bias of next-token prediction. For token co-training policies, an ordered representation can make the first supervised target describe global action-chunk structure rather than a single coordinate or a latent without an explicit global role. Many such orders are possible; for example, a coarse-to-fine order places high-impact motion early and leaves later tokens to refine residual detail.

Block compatibility. Block-wise generation is not a core tokenizer desideratum, but it is useful for scalable autoregressive inference. As formalized in Sec. 5.1, a block autoregressive schedule can ask the policy to emit several new tokens at the same prefix budget in a single call (Stern et al., 2018). Those tokens must be jointly predictable from the observation, the realized prefix, and prediction masks, without relying on realized within-block tokens as serial inputs. A compact, total, ordered tokenizer can still fail this property if its token structure was trained only as a strict token-wise chain. Table 1 summarizes how the tokenizers discussed in the paper satisfy these interface properties; the final rows show the token-wise and power-of-two `OAT` variants defined in Sec. 4.4.

3.2 Where Existing Tokenizers Fall Short

Existing action tokenizers satisfy different parts of this interface, but no standard baseline satisfies the three core desiderata while also providing block compatibility as shown in Table 1. Per-dimension `Bin` is reliable because every generated bin index maps back to a scalar action value, and coordinate groups can be decoded into valid actions. Its cost is rate and ordering: the token horizon grows with $H_a D_a$, and the manual coordinate serialization does not place coarse trajectory information early in the sequence.

`FAST` addresses rate and ordering by representing action trajectories through frequency-domain coefficients and then applying byte-pair encoding (BPE). Low-frequency components appear before high-frequency components, so early tokens tend to describe coarse motion. The difficulty is that byte-pair encoding expands tokens into variable-length coefficient streams, while the inverse frequency transform expects a fixed coefficient topology. Unconstrained policy samples can therefore

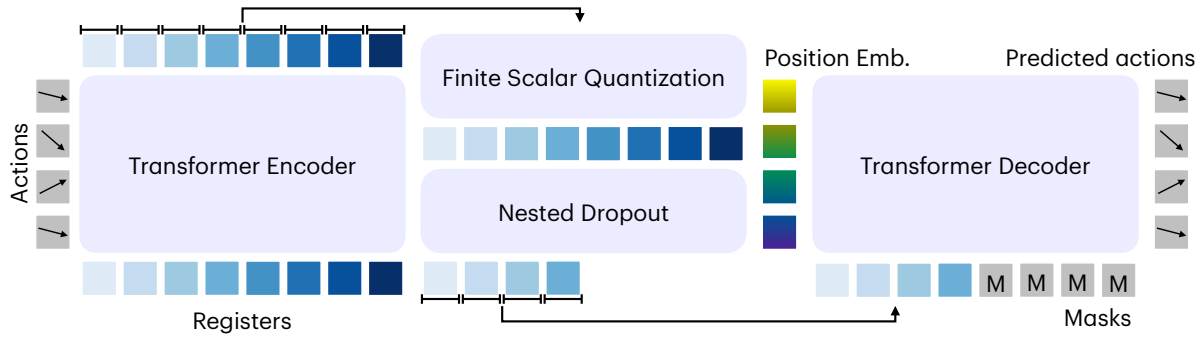


Figure 3: OAT tokenizer pipeline. OAT encodes a continuous action chunk with register tokens, discretizes the register states with finite scalar quantization, and trains a decoder to reconstruct the full action chunk from retained token prefixes. Nested dropout samples budgets and replaces suffixes with learned masks, forcing early tokens to carry coarse executable control while later tokens refine residual detail.

decode invalidly or require padding, truncation, rejection, or constrained decoding, each of which changes the policy interface.

Learned latent tokenizers such as QueST and ACodec decode through bottlenecks based on vector quantization or finite scalar quantization (Mentzer et al., 2024; Mete et al., 2024; Dong et al., 2026). QueST-style sequential latents are attractive from a rate–distortion viewpoint, but full-sequence reconstruction alone does not explicitly assign the first latent a global role over the action chunk. ACodec instead predicts fixed latent blocks jointly, reducing serial depth but making each block a harder joint prediction problem.

Thus reconstruction quality alone does not determine whether an action tokenizer is a good policy interface: the representation must also be compact, total over policy samples, and ordered for generation and supervision. For scalable autoregressive inference, compatibility with the intended block generation pattern is an additional side property. A detailed discussion of these baseline tokenizers is provided in Appendix D.

4 OAT: Ordered Action Tokenization

The tokenizer analysis above motivates OAT as a tokenizer for compact, totally decodable, ordered representations. This section defines the encoder–quantizer–decoder backbone, the ordered prefix training objective, and the token-wise and power-of-two variants studied in this paper.

4.1 Tokenization $\mathcal{T}$ and Detokenization $\mathcal{T}^{-1}$

OAT maps a continuous action chunk to a sequence of discrete action tokens and decodes token sequences back to continuous control. The tokenizer $\mathcal{T}$ is instantiated by an encoder E_ϕ and a quantization bottleneck, while the detokenizer $\mathcal{T}^{-1}$ is instantiated by a decoder D_θ, as summarized in Fig. 3. This autoencoder architecture ensures total decodability for the policy interface: every vocabulary index sequence maps to a continuous action chunk. We next describe the bottleneck design that gives OAT its compact and ordered token structure.

Register bottleneck. The encoder uses learnable registers $r_{1:H_l}$, inspired by ViT registers (Darcet et al., 2024), to compress a continuous action chunk $a_{1:H_a}$ into a fixed sequence of register states. Each register cross-attends (Vaswani et al., 2017) to all action positions, and the ordered prefix objective in the next subsection specifies the register self-attention mask. The encoder returns the register states $z_{1:H_l} = E_\phi(a_{1:H_a}, r_{1:H_l})$. Each latent $z_i \in \mathbb{R}^{D_l}$ is discretized into token T_i using finite scalar quantization (FSQ) (Mentzer et al., 2024). The resulting token sequence $T_{1:H_l}$ serves as the

token target for policy learning. The FSQ levels determine the vocabulary size, and H_l determines how many action tokens the policy must predict.

The decoder is a cross-attention Transformer: action position embeddings query the quantized register tokens and are projected into the H_a output actions, with no self-attention among action position queries. Next, we discuss training OAT with an ordered prefix objective.

4.2 Ordered Prefix Training for Progressive Tokens

Effective action tokenization requires more than compact reconstruction: the token sequence should have an order that policies can exploit. Our goal is to make early tokens capture coarse, globally salient aspects of an action chunk, while later tokens refine residual details. We use two complementary mechanisms to induce this ordering and support variable token budgets.

Nested Prefix Reconstruction OAT induces order through an increasing set of reconstruction budgets. Let $\mathcal{K} = \{k_1, \ldots, k_M\}$ denote the trained budget set, with $0 = k_0 < k_1 < \cdots < k_M = H_l$. During tokenizer training, we sample $K \sim \text{Uniform}(\mathcal{K})$, retain only the prefix $T_{1:K}$, and replace the suffix with learned mask tokens MASK before decoding. This produces the masked decoder input

$$\widetilde{T}^{(K)}_{1:H_l} = T_{1:K} \oplus \langle \texttt{MASK} \rangle_{K+1:H_l}.$$

The decoder must reconstruct the full action chunk from this partial code:

$$\mathcal{L}_{\text{prefix}} = \mathbb{E}_{a,K}\left[\left\| D_\theta(\widetilde{T}^{(K)}_{1:H_l}) - a_{1:H_a} \right\|_2^2 \right].$$

This is nested dropout over action tokens (Rippel et al., 2014; Kusupati et al., 2022; Cai et al., 2025; Bachmann et al., 2025). Unlike an autoencoder objective trained only on full token sequences, it trains every sampled prefix as an executable reconstruction point. These budgets specify the prefixes that receive direct reconstruction supervision; generated prefixes at intermediate budgets can also be decoded. Algo. 1 summarizes this tokenizer training loop.

Register Flow Constraints The register self-attention mask supplies an architectural ordering constraint independently of the reconstruction budget set. It determines whether registers introduced within the same reconstruction interval form a causal chain or have no direct cross-register dependencies. In both cases, each register can attend to earlier budget groups and itself. This separates two aspects of ordering: nested prefix reconstruction determines which budgets receive direct supervision, while the register mask determines the dependency structure among token positions. Fig. 4 visualizes the two variants studied in the paper, which are detailed in Sec. 4.4.

4.3 Progressive Information Allocation

The two mechanisms above give token ordering a source-coding-inspired information allocation interpretation. In classical source coding, common source patterns can be represented with shorter expected descriptions (Shannon, 1948). Here, the analogue of code length is the number of retained action tokens. Let $\varepsilon(K)$ denote the expected reconstruction error when only the first K tokens are retained, with $\varepsilon(0)$ denoting the all-mask error, and let $\Delta_i = \varepsilon(i-1) - \varepsilon(i)$ be the marginal gain from retaining token i. The expected nested dropout objective expands as

$$\mathbb{E}_K[\varepsilon(K)] = \varepsilon(0) - \sum_{i=1}^{H_l} \mathbb{P}(K \geq i)\, \Delta_i.$$

Thus token i is weighted by its survival probability $w_i = \mathbb{P}(K \geq i)$. The weights are nonincreasing and are equal for positions introduced between the same pair of successive budgets. Nested

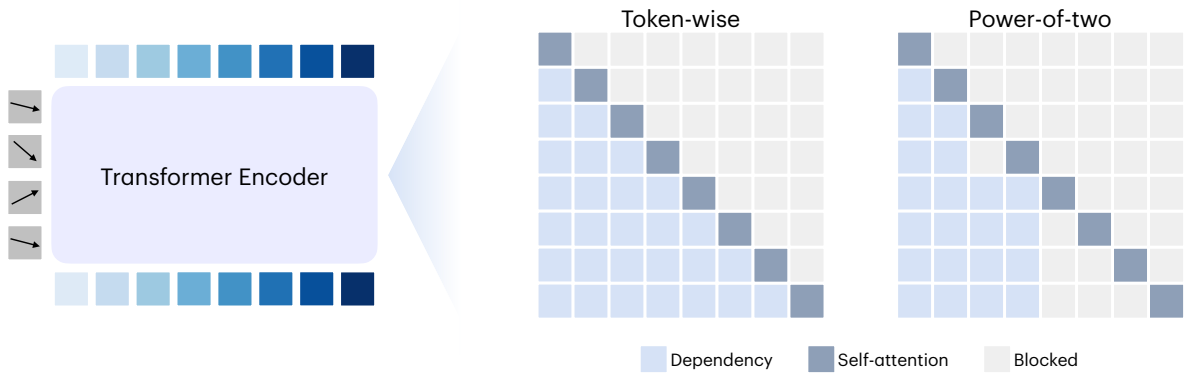


Figure 4: OAT encoder attention masks. Each matrix entry at row i and column j indicates whether register i can attend to register j. The token-wise mask gives causal register attention. The power-of-two mask preserves attention to earlier budget groups and self-attention, while blocking other registers in the same new group. The action inputs, registers, quantizer, nested dropout, and decoder remain unchanged.

prefix reconstruction therefore prioritizes earlier budget intervals without imposing an additional order among positions introduced at the same budget. This pressure favors placing coarse trajectory structure in earlier budgets and residual refinements in later budgets. The resulting information allocation is learned rather than assigned to coordinates, timesteps, or frequencies.

The resulting survival weights specify how reconstruction pressure is distributed over token positions, but do not uniquely determine the register dependency graph within each budget interval.

4.4 Token-Wise and Power-of-Two Attention Masks

We study two representative register attention masks for token-wise and power-of-two ordering. Both variants use $\mathcal{K} = \{1, 2, 4, \ldots, H_l\}$, sample K uniformly from this set, and otherwise share the same tokenizer architecture and training objective. Other budget sets and register groupings are possible; Sec. 5.1 develops the broader generation schedule space and motivates the power-of-two choice.

Token-wise ordering. $\texttt{OAT}^{\text{sing}}$ uses ordinary causal register attention, imposing a dependency order at every token position, including positions introduced within the same budget interval. Intermediate positions can therefore participate in token-wise generation, although only budgets in $\mathcal{K}$ receive direct reconstruction supervision.

Power-of-two ordering. $\texttt{OAT}^{\text{pow2}}$ groups positions introduced between successive reconstruction budgets and uses the block-causal register mask in Fig. 4. For $H_l = 16$ in our VLM experiments, the reconstruction budgets are $1, 2, 4, 8, 16$ with $k_0 = 0$. Positions introduced at the same budget k_m share the survival weight $\mathbb{P}(K \geq k_m)$, and direct cross-register attention within the group is blocked. They can therefore be encoded from the action input and earlier groups without within-group register dependencies, and subsequently treated as one generation block. $\texttt{OAT}^{\text{pow2}}$ imposes ordering across budget groups without introducing an additional order within each group. The variants therefore differ in the granularity of register dependencies.

Because both variants share the same trained budgets, the information-allocation view also defines a common diagnostic. For a fixed retained prefix length K, the token budget is a proxy for rate[1], and distortion is the reconstruction error $\varepsilon(K)$ defined above. Autoencoders trained only on full token sequences optimize only the endpoint $\varepsilon(H_l)$. OAT instead trains and evaluates this curve at multiple budgets $K \in \mathcal{K}$. Fig. 2 gives a qualitative example of prefix refinement, and Fig. 7 later

[1]The corresponding code length is proportional to $K \log_2(|\mathcal{C}|)$, where $\mathcal{C}$ is the quantization codebook.

Algorithm 1: OAT tokenizer training. Encode an action chunk, mask the suffix at a sampled budget, and reconstruct from the prefix.

Input: dataset $\mathcal{D}$; encoder E_ϕ; registers $r_{1:H_l}$; FSQ quantizer; decoder D_θ; mask token MASK; reconstruction budget set $\mathcal{K}$.

1. **while** not converged **do**
2. Sample $a_{1:H_a} \sim \mathcal{D}$ and $K \sim \text{Uniform}(\mathcal{K})$.
3. $z_{1:H_l} \leftarrow E_\phi(a_{1:H_a}, r_{1:H_l})$.
4. $T_{1:H_l} \leftarrow \text{FSQ}(z_{1:H_l})$.
5. $\widetilde{T} \leftarrow T_{1:K} \oplus \langle \texttt{MASK} \rangle_{K+1:H_l}$.
6. $\hat{a}_{1:H_a} \leftarrow D_\theta(\widetilde{T})$.
7. Update $\{\phi, \theta, r, \texttt{MASK}\}$ on $\|\hat{a}_{1:H_a} - a_{1:H_a}\|_2^2$.
8. **end while**
9. **return** $\mathcal{T}$ and $\mathcal{T}^{-1}$.

Algorithm 2: Autoregressive OAT generation. Generate tokens up to a target budget, mask the suffix, and return the decoded action chunk.

Input: observation history $o_{1:H_o}$; action token policy π; $\mathcal{T}^{-1} = \{D_\theta, \texttt{MASK}\}$; generation endpoint list $\mathbf{b} = (b_0 = 0, \ldots, b_m = K)$ with $g_s = b_s - b_{s-1}$.

1. Initialize $\widehat{T}_{1:0} \leftarrow \varnothing$ and $g_0 \leftarrow 0$.
2. **for** $s = 1, \ldots, m$ **do**
3. $\widehat{X}^{(s)} \leftarrow \widehat{T}_{1:b_{s-1}} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}}$.
4. $p_s(\cdot) \leftarrow \pi(\cdot \mid \widehat{X}^{(s)}, o_{1:H_o})$.
5. Sample $\widehat{G}_s$ from the final g_s logit-read slots.
6. $\widehat{T}_{1:b_s} \leftarrow \widehat{T}_{1:b_{s-1}} \oplus \widehat{G}_s$.
7. **end for**
8. $\widetilde{T} \leftarrow \widehat{T}_{1:K} \oplus \langle \texttt{MASK} \rangle_{K+1:H_l}$.
9. **return** action chunk $\hat{a}_{1:H_a} = \mathcal{T}^{-1}(\widetilde{T})$.

quantifies whether the learned prefixes reduce distortion smoothly as the token budget increases. The ordering ablation in Fig. 10 further tests whether this learned ordering is important for downstream policy performance. We next describe how visuomotor policies use these token structures for generation and supervision.

5 OAT for Visuomotor Policies

We instantiate OAT in the two policy interfaces introduced in Sec. 2. In autoregressive control, block-wise autoregression (BAR) specifies how ordered token positions are grouped into policy calls, while OAT provides the progressive, prefix-decodable action representation. In token co-training, the full OAT sequence supervises the VLM during training, while a flow-matching expert generates continuous action chunks from detached VLM context. The following subsections develop these interfaces in turn.

5.1 Block-wise Autoregressive OAT Generation

In the autoregressive role, a policy must choose how many action tokens to predict per policy call. We formulate BAR as a generation schedule over a fixed-length token sequence $T_{1:H_l}$. An endpoint list $\mathbf{b} = (b_0, b_1, \ldots, b_S)$, with $0 = b_0 < b_1 < \cdots < b_S = H_l$, partitions the sequence into blocks $G_s = T_{b_{s-1}+1:b_s}$ of size $g_s = b_s - b_{s-1}$. Stage s predicts G_s in one policy call conditioned on the realized prefix $T_{1:b_{s-1}}$. Thus S sets the sequential policy depth, while g_s sets the number of tokens predicted in parallel. Token-wise autoregression, fixed-size block prediction (Liu et al., 2026; Dong et al., 2026), one-shot parallel decoding (Dong et al., 2026), and intermediate variable-size patterns are all special cases, as illustrated in Fig. 5.

For the power-of-two horizons considered here, $H_l = 2^{S-1}$, we use endpoints $(1, 2, 4, \ldots, H_l)$ to preserve more serial conditioning for early tokens while increasing parallelism at later stages. Full generation then requires $S = 1 + \log_2 H_l$ policy calls, reducing the depth from $O(H_l)$ to $O(\log H_l)$. Appendix A shows that this depth is minimal when each new block is no larger than the realized prefix.

To train a nondecreasing schedule $g_1 \le \cdots \le g_S$, we set $T_{1:b_0} = \varnothing$ and $g_0 = 0$, and use the block-shifted input

$$X^{(s)} = T_{1:b_{s-1}} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}}.$$

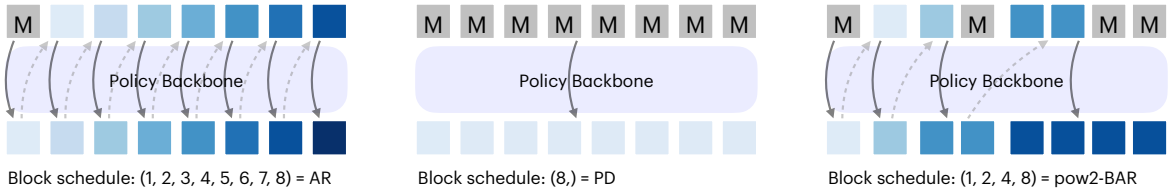


Figure 5: `BAR` block patterns. `BAR` generates action tokens block by block according to an endpoint list. For action token horizon $H_l = 8$, token-wise autoregression uses endpoints $(1, 2, 3, 4, 5, 6, 7, 8)$, parallel decoding uses endpoint (8), and the power-of-two pattern uses endpoints $(1, 2, 4, 8)$. Gray tokens marked "M" are masks, and blue tokens are newly predicted action tokens.

The final g_s positions provide the logit-read slots for predicting the current block G_s. For $s > 1$, these slots comprise the previous block G_{s-1} followed by $g_s - g_{s-1}$ new masks; at $s = 1$, they contain g_1 masks. Let $p^{(s)}_{\pi,j}(\cdot \mid X^{(s)}, o)$ denote the categorical distribution read from the j-th such slot under policy π and observation context o. The training objective is

$$\mathcal{L}_{\text{BAR}} = -\frac{1}{S}\sum_{s=1}^{S}\frac{1}{g_s}\sum_{j=1}^{g_s}\log p^{(s)}_{\pi,j}\big(T_{b_{s-1}+j} \mid X^{(s)}, o\big).$$

Thus, we average within each block and weight all generation stages equally. This objective produces all g_s logits in one policy forward pass without exposing ground-truth tokens from G_s. At inference, the same construction uses the generated prefix and appends the predicted block $\widehat{G}_s$ after each call.

Both evaluated variants train the tokenizer decoder at $\mathcal{K} = \{1, 2, 4, \ldots, H_l\}$. The `BAR` endpoint list instead specifies how the policy reaches a target budget K and may include intermediate generation endpoints. `OAT`$^{\text{sing}}$ uses singleton blocks with $b_s = s$, requiring K policy calls to reach a length-K prefix. `OAT`$^{\text{pow2}}$ uses endpoints $(1, 2, 4, 8, 16)$ for $H_l = 16$, requiring five calls for complete generation. At any generation endpoint, the generated prefix can be suffix-padded with learned mask tokens and decoded immediately. Budgets in $\mathcal{K}$ receive direct reconstruction supervision and typically yield higher reconstruction quality; generation can also continue to a larger budget. Algo. 2 and Appendix A provide the full inference procedure and `BAR` specification.

5.2 Token Co-Training with `OAT`

In the token co-training role, `OAT` tokens supervise the VLM rather than being decoded into actions. Let c denote the image, language, and robot state context. Given a frozen tokenizer $\mathcal{T}$, each training action chunk $a_{1:H_a}$ defines a target sequence $T_{1:H_l} = \mathcal{T}(a_{1:H_a})$. The VLM predicts this sequence under teacher forcing, while the action expert learns from the same action chunk and a detached VLM key/value (K/V) cache. Their joint objective is (Driess et al., 2025; Fang et al., 2026)

$$\mathcal{L}_{\text{TC}} = \mathcal{L}_{\text{tok}}(T_{1:H_l} \mid c) + \lambda\, \mathcal{L}_{\text{flow}}\Big(a_{1:H_a}, \widetilde{a}^{\tau}_{1:H_a}, \tau; \text{sg}(\text{KV}_{\text{VLM}}(c))\Big), \tag{5.1}$$

where $\mathcal{L}_{\text{tok}}$ is the cross-entropy for predicting each T_i from c and the shifted ground-truth prefix $\langle \texttt{MASK} \rangle, T_{1:i-1}$. The leading mask supplies the prediction slot for T_1. The term $\widetilde{a}^{\tau}_{1:H_a}$ denotes the noised action chunk at flow time τ, and $\text{sg}(\cdot)$ denotes stop-gradient. The layer-wise cache $\text{KV}_{\text{VLM}}(c)$ is computed from the image, language, and robot state prefill only, excluding the teacher-forced action token positions.

The two losses follow separate gradient paths. The token loss provides action supervision to the VLM, whereas the flow loss trains the action expert without propagating through the detached cache. At inference, the VLM is prefilled once on c, but its action token logits are not sampled.

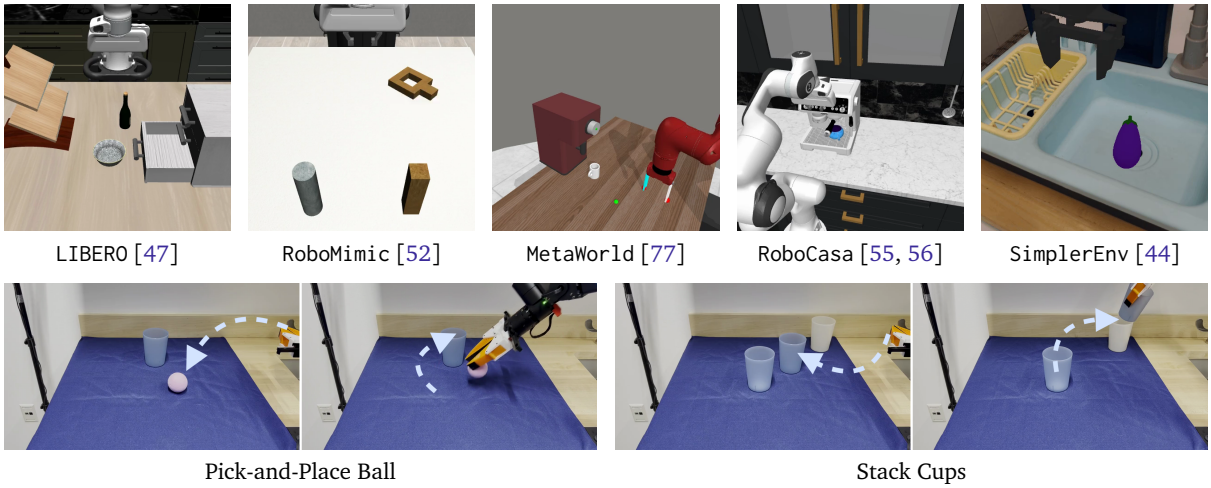


Figure 6: Evaluation environments. Top row: simulated manipulation benchmarks used for lightweight policies, VLM AR policies, and VLM TC policies. **Bottom row:** real-world tabletop tasks evaluated with a fixed-base ARX-5 arm and a single Logitech webcam; each filmstrip shows one representative rollout.

The flow-matching expert instead generates a continuous action chunk from the cached context for execution; no action tokens are generated or detokenized.

Because token co-training uses the full OAT sequence only as teacher-forced targets, its relevant OAT property is the prediction objective imposed on the VLM prefill representation. The one-token prefix objective trains the first OAT token to support reconstruction of the complete action chunk. Under token co-training, this is the first target in the teacher-forced sequence and is therefore predicted directly from c, without preceding action tokens. Its cross-entropy loss trains the prefill representation later consumed by the action expert against a chunk-summary target. We view this as plan-like supervision: the VLM must infer a summary of the complete action chunk from the image, language, and robot state context alone. By comparison, coordinate binning assigns the first target to a single action scalar, while learned latent baselines optimized only for full-sequence reconstruction do not explicitly train their first target to support full-chunk reconstruction. These alternatives therefore do not explicitly train the first VLM prediction against an analogous chunk-summary target. Fig. 9 evaluates this supervision against alternative action tokenizers; Appendix B provides further flow-matching and inference details.

6 Experiments

Experiments assess OAT at three levels of the action token interface. We begin at the tokenizer level in Sec. 6.2, testing whether OAT forms a compact, progressive action representation whose prefixes remain executable rather than only optimizing full-sequence reconstruction. We then move to closed-loop policy learning in Sec. 6.3, spanning lightweight policies and VLM-scale policies under autoregressive (AR) generation and token co-training (TC). Finally, Sec. 6.4 analyzes token ordering, action and token horizons, codebook capacity, and grouped generation to clarify the design choices behind OAT.

6.1 Experimental Setup

We evaluate OAT at two policy scales. The lightweight regime fixes the `Transformer` backbone and compares action representations across `LIBERO`-Long (Liu et al., 2023), `RoboMimic` (Mandlekar et al., 2022), `MetaWorld` (Yu et al., 2020), `RoboCasa` (Nasiriany et al., 2024), and two real-world

Setting	Benchmark	Tasks / suites	# Tasks	$H_a \times D_a$	Freq. (Hz)	Med. len.
Lightweight	LIBERO	Long suite	10	32×7	10	259
	RoboMimic	Lift; Square; Can	3	32×7	20	114
	MetaWorld	Box Close; Coffee Pull; Disassemble; Stick Pull	4	32×4	80	72
	RoboCasa	Close Drawer; Coffee Press Button; Turn Off Microwave; Turn Off Sink Faucet	4	32×12	20	184
	Real-world	Pick-and-Place Ball; Stack Cups	2	32×7	10	98
VLM	LIBERO	Long, Goal, Object, and Spatial suites	40	32×7	10	140
	RoboMimic	Lift; Square; Can; Tool Hang	4	32×7	20	130
	SimplerEnv	WidowX+Bridge series	4	8×7	5	37
	RoboCasa365	Close Toaster Oven Door; Open Drawer; Pick Place Drawer to Counter; Turn On Electric Kettle; Slide Dishwasher Rack	5	32×12	20	194

Table 2: Benchmark setup. Rows summarize the simulated and real-world settings used for policy evaluation. $H_a \times D_a$ gives the predicted action horizon and action dimension; Med. len. is the median episode length in environment steps. Task subsets differ by evaluation regime as listed. The lightweight regime fixes a Transformer policy, while the VLM regime uses the backbones in Table 3; within each regime, tokenizer comparisons keep the policy backbone fixed.

Backbone	Scale	Interface	Ctx. attn.	Act. attn.
Transformer	5M	Cross-attn.	Cross	Causal
PaliGemma2 [66]	3B	Prefix VLM	Full	Causal
Qwen3VL [4]	2B	Causal VLM	Causal	Causal

Table 3: Policy backbone interfaces. Rows define the backbone attention interfaces used in later comparisons. Ctx. attn. covers observation, language, and state tokens; Act. attn. covers generated action tokens.

tasks. The VLM-scale regime uses PaliGemma2 and Qwen3VL backbones under AR generation and TC supervision. The evaluation covers LIBERO, RoboMimic, and SimplerEnv (Li et al., 2025). We additionally use RoboCasa365 (Nasiriany et al., 2026).

Within each comparison block, methods share the observation interface, data split, rollout protocol, and policy backbone. The varied factors are the action representation and, for AR policies, the generation pattern; TC comparisons instead vary token supervision while holding the flow-matching expert architecture, objective, and training recipe fixed. Fig. 6 shows the environments, Table 2 summarizes benchmark coverage, and Tables 3 and 4 specify the policy interfaces and generation costs. Full rollout, tokenizer, policy, and optimization details are provided in Appendix C.

6.2 Rate–Distortion of Action Tokens

Fig. 7 measures reconstruction error from truncated token prefixes. For token budget k, we retain $T_{1:k}$, fill the ungenerated suffix with mask tokens, decode the partial sequence, and report its mean squared error against the original action chunk. This diagnostic quantifies how much action information each prefix budget preserves before the tokens are used as policy targets.

The baselines expose distinct rate–distortion operating points. Bin is nearly lossless but requires $H_a D_a$ tokens. FAST shortens the sequence through frequency-domain coding and BPE, while QueST and ACodec provide compact learned-latent operating points at their full token horizons. In contrast, OAT traces a family of operating points from one-token sketches to full-length reconstructions, allowing the same tokenizer to trade token budget for action fidelity.

Both OAT variants reduce reconstruction error smoothly as the token budget increases, and the

Scheme	Token structure	Generation	# Calls	Max block
`Bin`	Raw coordinate bins	stepwise D_a blocks	H_a	D_a
`FAST`	Frequency coefficients with BPE	token-wise BPE sequence	$\lvert T \rvert$	1
`QueST`	Learned temporal latents	token-wise latent sequence	H_l	1
`ACodec`	Joint latent block	one-shot latent block	1	H_l
$\texttt{OAT}_k^{\text{sing}}$	token-wise ordering	token-wise AR	k	1
$\texttt{OAT}_k^{\text{pow2}}$	power-of-two ordering	power-of-two `BAR`	$1+\lceil \log_2 k \rceil$	$\lceil k/2 \rceil$

Table 4: Action token budget and generation settings. Rows compare each tokenizer setting, generation pattern, and call cost. # Calls is the number of sequential policy calls for one action chunk, and Max block is the largest action token block generated in one policy call. For the `OAT` rows, both variants train at power-of-two reconstruction budgets, and k denotes one such budget. $\texttt{OAT}^{\text{pow2}}$ additionally uses power-of-two register groups and `BAR` endpoints. For `FAST`, $|T|$ denotes generated BPE length.

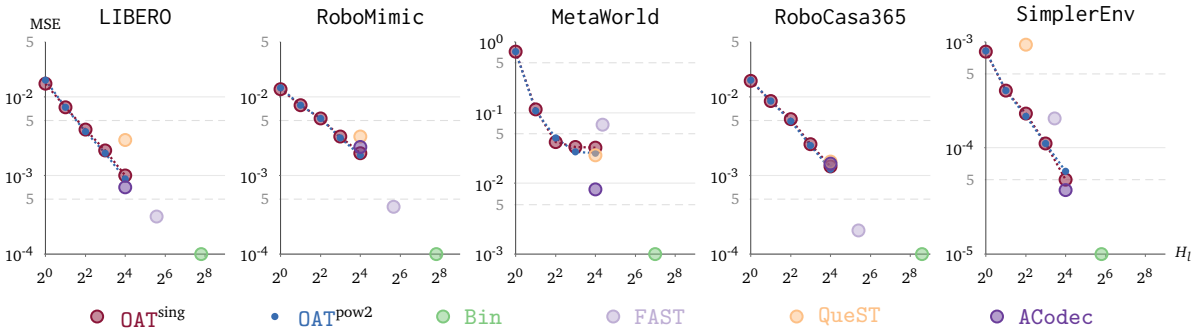


Figure 7: Rate–distortion curves for action tokenizers. Each panel plots raw reconstruction MSE against token budget on log–log axes; lower and further left is better. The `OAT` traces show mask-padded partial decodings under token-wise and power-of-two variants for $k \in \{1, 2, 4, 8, 16\}$, while `Bin`, `FAST`, `QueST`, and `ACodec` appear as fixed-budget operating points at their full token length, with `FAST` using its average BPE length. Near-zero `Bin` errors are drawn on the x-axis for visual consistency. Detailed token counts and MSE values are listed in Table 9.

power-of-two mask closely tracks the token-wise mask across budgets. Thus grouped register dependencies preserve progressive rate–distortion behavior without materially degrading reconstruction. Exact token counts and MSEs (10^{-3}) are listed in Appendix E.

Rate–distortion measures information retention, but it does not establish whether the resulting tokens support effective policy learning.

6.3 Policy Evaluation

We next ask whether tokenizer design translates into closed-loop policy performance. We evaluate this question across policy scales and action-token roles.

6.3.1 Lightweight Policies

Table 5 compares tokenizers in closed-loop control using the same small `Transformer` policy, isolating the action representation and generation pattern. We evaluate on `LIBERO-Long`, `RoboMimic`, `MetaWorld`, and `RoboCasa`, followed by the Pick-and-Place Ball and Stack Cups real-robot tasks in Fig. 6.

`OAT` performance improves with token budget: the shortest budgets are executable but coarse, whereas $\texttt{OAT}_8^{\text{sing}}$ gives the highest point estimate in every simulation and real-world group and the

Scheme	Simulation				Real-world		Avg. Rank
	LIBERO-Long	RoboMimic	MetaWorld	RoboCasa	P&P Ball	Stack Cups	
Bin	14.4	39.5	14.5	27.7	4/20	8/20	5.8
FAST	23.0	24.0	7.1	13.2	8/20	6/20	6.2
QueST	48.2	66.9	17.9	52.3	11/20	8/20	2.8
$\text{OAT}_{1}^{\text{sing}}$	11.7	50.8	11.3	47.7	7/20	3/20	6.0
$\text{OAT}_{2}^{\text{sing}}$	39.8	52.5	16.4	50.3	11/20	9/20	3.8
$\text{OAT}_{4}^{\text{sing}}$	46.4	65.3	19.5	51.7	13/20	12/20	2.5
$\text{OAT}_{8}^{\text{sing}}$	56.3	73.1	24.4	54.6	16/20	16/20	1.0

Table 5: Lightweight policy simulation and real-world results. Simulation entries are mean success rates in percent with a fixed Transformer policy; real-world entries are successful trials out of 20 independent rollouts. For this lightweight comparison, FAST uses strict decoding, so invalid FAST token sequences are rejected as described in Sec. 3. Avg. Rank is balanced over the four simulation benchmarks and two real-world tasks; lower is better.

best average rank. Both QueST and OAT use compact learned latents, but only OAT trains ordered prefixes to place control-relevant action information early; the matched ordering ablation in Fig. 10 tests this factor directly. These results provide closed-loop evidence that OAT prefixes improve policy learning and motivate evaluation at VLM scale.

At VLM scale, AR policies generate and detokenize action tokens, whereas TC policies use token losses to supervise the VLM while a flow-matching expert executes actions.

6.3.2 Autoregressive VLM Policies

Fig. 8 shows that OAT remains effective for VLM AR policies. Across both backbones, success generally improves with token budget; the best token-wise result is $\text{OAT}_{16}^{\text{sing}}$: 63.7 with PaliGemma2 and 56.8 with Qwen3VL.

BAR exposes the depth–accuracy tradeoff for the same ordered token family. At $k = 16$, the power-of-two variant matches token-wise generation with PaliGemma2 (63.8 vs. 63.7) using 5 rather than 16 policy calls; with Qwen3VL, token-wise conditioning is stronger (56.8 vs. 50.6). Full values appear in Appendix F.

Baseline comparisons confirm that tokenizer design remains consequential at VLM scale. Bin reconstructs almost exactly in Table 9, yet its long action suffix gives poor average success with both backbones. ACodec instead predicts all action tokens in one policy call, but varies across benchmarks and backbones. Useful action tokens must therefore be compact and decodable while remaining learnable by the policy interface.

6.3.3 Token Co-Training VLM Policies

Fig. 9 shows that tokenizer choice remains consequential under TC supervision. With PaliGemma2, OAT reaches 59.0 average success, tying QueST and ACodec while outperforming Bin and FAST; it leads on LIBERO and SimplerEnv, while QueST is stronger on RoboMimic (68.5 vs. 61.0). With Qwen3VL, OAT gives the highest average success at 62.5 and leads on RoboMimic and RoboCasa365; QueST remains stronger on SimplerEnv (33.0 vs. 27.5). Thus tokenizer design matters even when tokens supervise the VLM rather than define the executed action, although the best tokenizer can depend on the backbone and benchmark. This supports Sec. 5.2: the plan-like first-token target directly supervises the VLM prefill representation consumed by the expert.

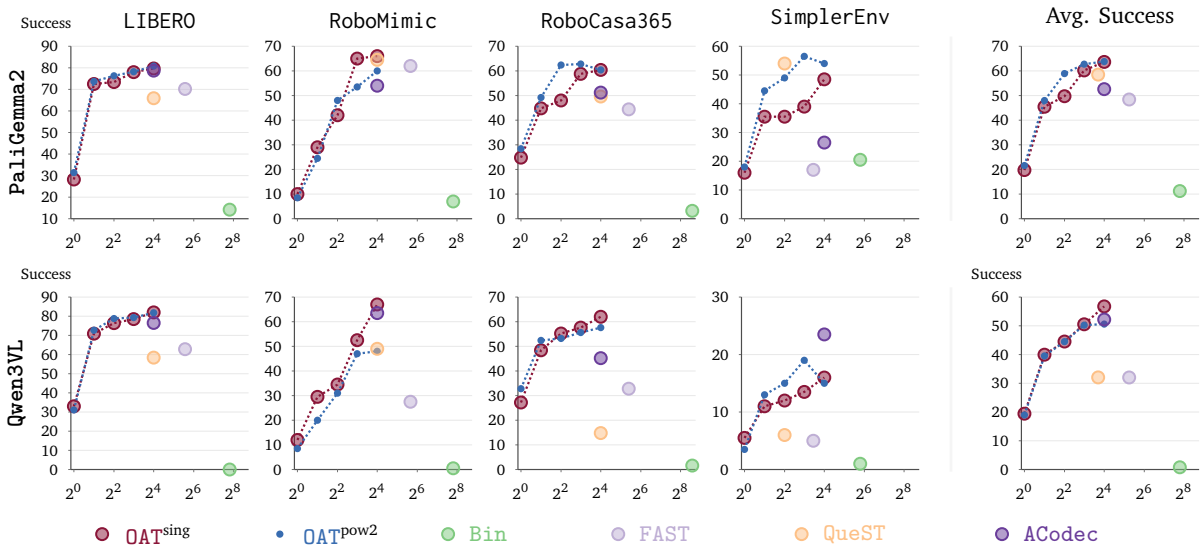


Figure 8: Closed-loop autoregressive VLM policy success rates. Each panel reports mean task success over 50 rollouts per task for one VLM backbone and benchmark group. The x-axis is token budget on a 2^t scale. Dotted curves trace OAT token budgets; fixed-budget baselines are plotted at their tokenizer lengths. Call counts are listed in Table 4. Rightmost panels report benchmark-balanced average success for each backbone. Numeric values are provided in Table 10.

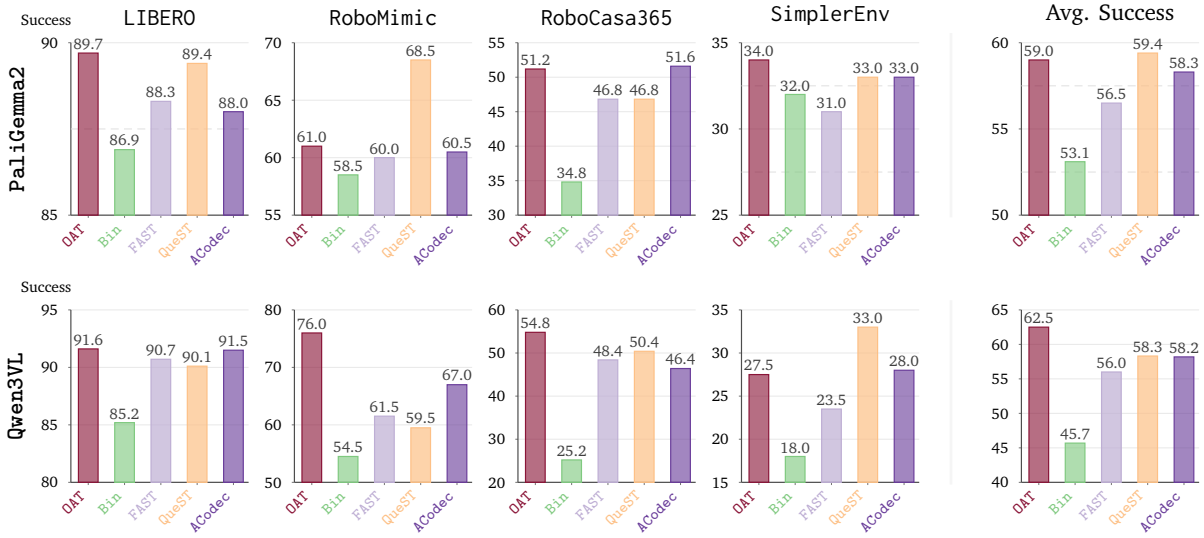


Figure 9: Closed-loop token co-training VLM policy success rates. Rows are VLM backbones trained with action token supervision and a stop-gradient boundary that blocks flow-matching expert losses from updating the VLM. Columns report mean task success over 50 evaluation episodes per task on LIBERO, RoboMimic, RoboCasa365, and SimplerEnv, plus the average with equal weight across the four benchmark panels. Within each row, bars compare the action token supervision used during training: OAT, Bin, FAST, QueST, and ACodec. At inference, these tokens are not decoded; the flow-matching expert produces actions from cached, detached VLM K/V context. Numeric labels give success in percent. Each panel uses its own vertical axis scale, shown by tick labels, so compare bar heights within each panel.

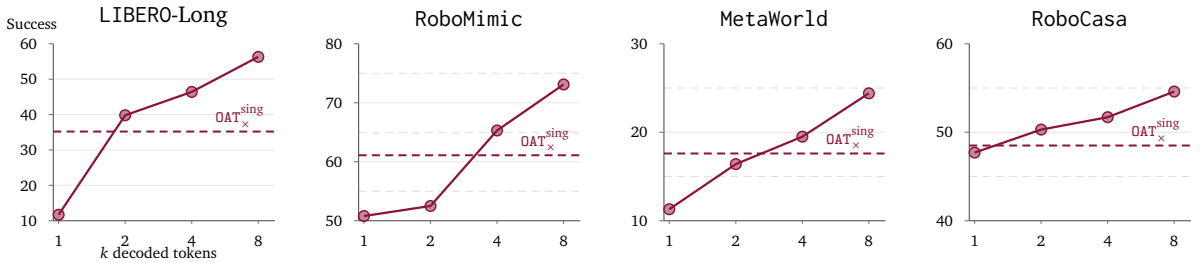


Figure 10: Token ordering ablation. Each panel reports one benchmark in the lightweight setting, with mean success over 50 rollouts per task when decoding the first k OAT$^{\text{sing}}$ tokens. Solid curves show ordered OAT$^{\text{sing}}$ budgets; dashed lines show OAT$^{\text{sing}}_{\times}$, which removes nested dropout during tokenizer training while keeping the tokenizer architecture and policy interface fixed.

6.4 Ablation and Analysis

Ablations examine the main design choices behind OAT: token ordering, action and token horizons, codebook capacity, and grouped generation. Unless otherwise specified, these studies use the lightweight `Transformer` setting, so each comparison keeps the policy backbone fixed and changes only the action representation.

6.4.1 Does Token Ordering Improve Policy Performance?

Fig. 10 evaluates the effect of token ordering. OAT$^{\text{sing}}_{\times}$ removes nested dropout, so short prefixes are no longer trained to reconstruct the action chunk. This separates the ordering objective from learnable registers and compact latent capacity.

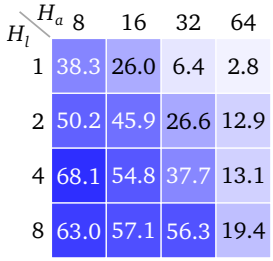
Removing ordering consistently degrades lightweight policy success. OAT$^{\text{sing}}_{\times}$ still uses the full latent horizon, so it is often stronger than the shortest OAT$^{\text{sing}}$ budgets, but it remains well below OAT$^{\text{sing}}_{8}$ and is often closer to OAT$^{\text{sing}}_{2}$ or OAT$^{\text{sing}}_{4}$. This supports the claim that compact latent tokens alone are insufficient for strong policies. Ordering is an important factor: next-token prediction benefits when high-impact action structure appears early and residual detail later.

6.4.2 How Do Action and Token Horizons Trade Off?

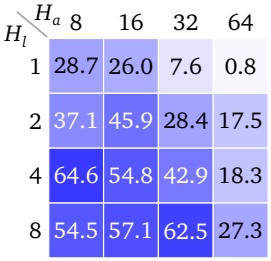
Fig. 11 studies two central factors in chunk-level action tokenization: the predicted action horizon H_a and the latent token horizon H_l. Larger H_a provides more future context to the policy, but also requires the tokenizer to compress a longer continuous trajectory. Larger H_l provides more register slots, but increases the action token suffix that the policy must model. We train models for $H_a \in \{8, 16, 32, 64\}$ and $H_l \in \{1, 2, 4, 8\}$ on LIBERO-Long and evaluate two execution protocols: execution after half the action horizon, which matches the protocol used elsewhere, and fixed 8-action execution, which holds execution frequency constant.

Executing half the action horizon exposes the compression side of the tradeoff. For a fixed H_l, success generally drops as H_a grows because the same number of tokens must represent a longer future chunk. Increasing H_l mitigates this drop, indicating that additional register slots are needed to preserve fine temporal structure. This supports using a larger latent horizon such as $H_l = 8$, while the action horizon must also account for execution frequency.

The fixed 8-action protocol separates prediction horizon from execution frequency. For a fixed H_l, a longer H_a can initially help because the policy predicts further into the future while executing the same number of actions per query. However, very long horizons again become difficult to compress, especially for $H_l \in \{1, 2\}$. In this setting, $H_a = 32$ with $H_l = 8$ is a strong compromise between lookahead and compression. The two heatmaps therefore point to the same design rule: action chunking and token capacity should be chosen jointly.



(a) Half-horizon execution.



(b) Fixed 8-action execution.

Figure 11: Action and token horizons. $\texttt{OAT}^{\text{sing}}_{H_l}$ success on LIBERO-Long with a fixed `Transformer` policy as action horizon H_a and the latent token horizon H_l varies. **(a)** queries again after executing $\frac{1}{2}H_a$ actions; **(b)** always executes 8 actions, isolating execution frequency. The panels expose compression and replanning tradeoffs. Values are success over 50 rollouts per task; darker cells are better.

FSQ levels	$\lvert\mathcal{V}\rvert$	LIBERO
$[8,6,5]$	240	29.2
$[8,8,8]$	512	53.5
$[8,5,5,5]$	1000	56.3
$[8,8,6,5]$	1920	54.6
$[7,5,5,5,5]$	4375	46.9

Table 6: Codebook capacity scaling. $\texttt{OAT}^{\text{sing}}$ success on LIBERO-Long with fixed `Transformer` ($H_a = 32$, $H_l = 8$) as FSQ levels vary [53]. Moderate vocabularies work best: too few codes can restrict the representation, while too many can make token targets harder to learn.

Scheme	Ordering	Generation	Calls	LIBERO
$\texttt{OAT}^{\text{sing}}_{16}$	token-wise	token-wise AR	16	79.7
$\texttt{OAT}^{\text{sing}}_{16}$ + post-hoc BAR	token-wise	power-of-two BAR	5	66.3
$\texttt{OAT}^{\text{pow2}}_{16}$	power-of-two	power-of-two BAR	5	80.8

Table 7: Grouped OAT generation on LIBERO. `PaliGemma2` success. Post-hoc BAR changes only endpoints; $\texttt{OAT}^{\text{pow2}}_{16}$ matches register ordering to blocks.

6.4.3 How Does Codebook Capacity Affect Policy Success?

Table 6 varies the FSQ levels while keeping the rest of the tokenizer fixed, isolating discrete vocabulary capacity from the number of latent tokens. The trend is non-monotonic. Success improves as the vocabulary increases from 240 to roughly 1000–2000 codes, consistent with the intuition that a small codebook is restrictive. Beyond that range, performance drops even though the tokenizer has more discrete capacity. One plausible explanation is that larger vocabularies spread supervision over more discrete targets, so the policy observes fewer examples per code. This reinforces the broader point from Sec. 3: effective action tokens must reconstruct actions and remain learnable policy targets.

6.4.4 Does Grouped Generation Require Matched OAT Ordering?

The final ablation tests whether OAT can be regrouped at inference without matching its register dependency structure to the generation blocks. Both tokenizers use the same reconstruction budgets. We apply power-of-two grouped generation post hoc to $\texttt{OAT}^{\text{sing}}_{16}$ by overriding only its generation endpoint list while retaining its causal register attention. We compare this condition with $\texttt{OAT}^{\text{pow2}}_{16}$, whose block-causal register groups match the generation blocks.

Table 7 separates faster generation from tokenizer compatibility. For $\texttt{OAT}^{\text{sing}}_{16}$, switching only the generation pattern reduces the number of calls from 16 to 5, but success drops from 79.7 to 66.3. $\texttt{OAT}^{\text{pow2}}_{16}$ reaches 80.8 with the same call count. Grouped generation therefore works best when the tokenizer's register dependency structure matches the generation blocks.

7 Related Work

We organize the most relevant work around visuomotor policy interfaces, action token representations, and block autoregressive generation.

7.1 Visuomotor Policy Interfaces

Action chunking predicts temporally coherent control segments and amortizes inference across multiple environment steps (Zhao et al., 2023; Zhang et al., 2025). Diffusion and flow-matching policies provide strong continuous action decoders for high-frequency control (Chi et al., 2025; Black et al., 2025,), while multitask systems extend this interface across task families, sensing modalities, and symbolic-continuous planning settings (Liu et al., 2026; Chen et al., 2025; Høeg et al., 2026; Liu et al., 2026). These methods establish the action chunk as an effective control unit, but usually leave it in continuous space. We study the complementary discrete interface, where the chunk is represented as a token sequence.

Large robot policies increasingly condition on language and often reuse VLM backbones for embodied control (Zitkovich et al., 2023; Driess et al., 2023; Ghosh et al., 2024; Kim et al., 2024; Li et al., 2024,; Qu et al., 2025; Wen et al., 2025; Black et al., 2025; NVIDIA et al., 2025), supported by large-scale datasets and generalist policy efforts (O'Neill et al., 2024; Walke et al., 2023; Khazatsky et al., 2024; Bousmalis et al., 2024). For policies that generate action tokens, tokenization determines output length, action validity, and next-token prediction difficulty, making it a central design axis (Zhong et al., 2025; Vuong et al., 2025; Wang et al., 2025). Discrete diffusion decoders retain a token interface while replacing fixed left-to-right generation with iterative parallel refinement (Liang et al., 2025).

Token co-training systems, sometimes referred to as knowledge insulation in other literature, instead use discrete action losses to update the VLM while a diffusion or flow-matching expert consumes detached VLM context and executes continuous actions (Black et al., 2025; Driess et al., 2025; Intelligence et al., 2026; Fang et al., 2026). Data-mixture studies likewise examine how vision-language and cross-embodiment data preserve VLM knowledge during robot training (Lin et al., 2026). Together, direct token generation and token co-training motivate action representations that are both predictable and useful as supervision.

7.2 Action Tokenization and Ordered Representations

Existing action tokenizers trade off compression, guaranteed decoding, and policy prediction difficulty. Per-dimension `Bin` is simple and totally decodable but produces sequences whose length grows with action dimension and chunk horizon (Brohan et al., 2023; Zitkovich et al., 2023; Kim et al., 2024). `FAST` compresses chunks with frequency-domain structure and BPE, introducing a low-to-high-frequency order together with variable-length decoding issues (Pertsch et al., 2025). Learned tokenizers use neural encoders and discrete bottlenecks for skill or latent action abstractions (Mete et al., 2024; Lee et al., 2024; Ye et al., 2025). These systems commonly draw on vector-quantization methods (van den Oord et al., 2017; Mentzer et al., 2024); recent VLA implementations include VQ-VLA and ActionCodec (Wang et al., 2025; Dong et al., 2026). Token surrogates also appear in in-context imitation learning and robotic sequence modeling (Palo and Johns, 2024; Fu et al., 2025; Bonatti et al., 2022). These methods establish action tokens as policy targets, but do not resolve which token properties make policy learning reliable.

Ordered representations provide an inductive bias for consuming or generating partial codes. Nested dropout and Matryoshka-style objectives retain information at multiple budgets (Rippel et al., 2014; Kusupati et al., 2022; Cai et al., 2025; Bachmann et al., 2025). Image and sequence models further show that representation design and autoregressive factorization change the modeling

problem faced by a downstream generator (Kolesnikov et al., 2022; Wen et al., 2025; Khemakhem et al., 2021; Im et al., 2025). This supports a broader view of latent-space generation in which representations should be selected for downstream predictability as well as reconstruction (Tschannen et al., 2018; Dieleman, 2025; Zhao et al., 2025; Kim et al., 2025). Action tokens add a control-specific requirement: retained prefixes should decode to executable action chunks rather than only embeddings.

7.3 Block Autoregressive Generation

Partially parallel generation reduces autoregressive latency by predicting, verifying, or refining multiple tokens per stage. Block-wise parallel decoding, speculative decoding, masked prediction, and non-autoregressive refinement instantiate this idea in language and sequence models (Stern et al., 2018; Leviathan et al., 2023; Ghazvininejad et al., 2019; Lee et al., 2018). Recent VLA tokenizers likewise target latency and prediction difficulty through fixed-size block prediction in action generation (Liu et al., 2026; Dong et al., 2026), while block diffusion interpolates between autoregressive and parallel generation through block-level denoising (Arriola et al., 2025). BAR provides a common schedule view of these action token generation patterns, with fixed-size block prediction as one point in the broader family.

8 Conclusion and Limitations

This paper studied action tokenization as a policy interface for visuomotor policy learning. We formalized three desiderata: high compression, total decodability, and ordered token structure. We then introduced OAT, a learned tokenizer whose mask-padded prefixes decode to executable action chunks. Early tokens capture coarse control, while later tokens refine residual detail, enabling flexible prefix-based generation in autoregressive policies.

Across tokenizer diagnostics, lightweight control, VLM-scale autoregressive and token co-training policies, and ablations, the results show that OAT is a strong action token interface. For autoregressive policies, OAT improves closed-loop control; BAR further formalizes token-wise, block-wise, one-shot, and power-of-two schedules, making the tradeoff between policy call depth and block prediction difficulty explicit. Under token co-training, tokenizer choice remains consequential even when action tokens are not decoded at inference. These results are consistent with the cross-entropy loss on the first OAT target imposing a plan-like, action-chunk-level objective on the VLM prefill representation consumed by the action expert.

The study is intentionally scoped to isolate tokenizer effects. We evaluate a fixed set of manipulation benchmarks, policy backbones, action horizons, and token budgets; broader embodiments, longer-horizon tasks, and larger real-world mixtures remain important tests of the same design principles. We also keep the flow-matching expert in token co-training policies fixed while varying tokenizers, leaving the joint design of expert architecture and token supervision for future work.

A natural next step is adaptive computation. Current policies choose token or block budgets before inference, but OAT prefixes can be decoded at any budget and BAR makes the cost of additional generated blocks explicit. Future policies could decide online when another token or block is worth the policy call, using token entropy, reconstruction uncertainty, or downstream value estimates (Graves, 2016; Banino et al., 2021; Dehghani et al., 2019; Elbayad et al., 2020). This would reserve deeper autoregressive refinement for precise or high-risk moments while keeping simple control decisions compact.

Acknowledgments

The computations in this paper were carried out in part on the FASRC cluster supported by the FAS Division of Science Research Computing Group at Harvard University, in part on cloud computing resources provided through the Lambda Research Grant Program, and in part on Delta system at National Center for Supercomputing Applications, Anvil supercomputer at Purdue University and Bridges-2 system at the Pittsburgh Supercomputing Center through allocation CIS260779 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

References

[1] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974. doi: 10.1109/T-C.1974.223784. 41

[2] Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=tyEyYT267x. 19

[3] 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. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 2241–2292. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/bachmann25a.html. 2, 4, 7, 18

[4] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631. 12

[5] Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=1EuxRTe0WN. 19

[6] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/black25a.html. 4, 18, 38

[7] 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 Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.010. 18, 37

[8] Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=UkR2zO5uww. 18

[9] Yochai Blau and Tomer Michaeli. Rethinking lossy compression: The rate-distortion-perception tradeoff. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 675–685. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/blau19a.html. 4

[10] Rogerio Bonatti, Sai Vemprala, Shuang Ma, Felipe Frujeri, Shuhang Chen, and Ashish Kapoor. Pact: Perception-action causal transformer for autoregressive robotics pre-training, 2022. URL https://arxiv.org/abs/2209.11133. 3, 18

[11] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Manon Devin, Alex X. Lee, Maria Bauza Villalonga, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, Antoine Laurens, Claudio Fantacci, Valentin Dalibard, Martina Zambelli, Murilo Fernandes Martins, Rugile Pevceviciute, Michiel Blokzijl, Misha Denil, Nathan Batchelor, Thomas Lampe, Emilio Parisotto, Konrad Zolna, Scott Reed, Sergio Gómez Colmenarejo, Jonathan Scholz, Abbas Abdolmaleki, Oliver Groth, Jean-Baptiste Regli, Oleg Sushkov, Thomas Rothörl, Jose Enrique Chen, Yusuf Aytar, David Barker, Joy Ortiz, Martin Riedmiller, Jost Tobias Springenberg, Raia Hadsell, Francesco Nori, and Nicolas Heess. Robocat: A self-improving generalist agent for robotic manipulation. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=vsCpILiWHu. 18

[12] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S Ryoo, Grecia Salazar, Pannag R Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan H Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.025. 2, 3, 5, 18, 39, 41

[13] Mu Cai, Jianwei Yang, Jianfeng Gao, and Yong Jae Lee. Matryoshka multimodal models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Learning Representations*, volume 2025, pages 46254–46272, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/722fcbc1a6667f2075d75ea79a1b3552-Abstract-Conference.html. 7, 18

[14] Haonan Chen, Jiaming Xu, Hongyu Chen, Kaiwen Hong, Binghao Huang, Chaoqi Liu, Jiayuan Mao, Yunzhu Li, Yilun Du, and Katherine Driggs-Campbell. Multi-modal manipulation via multi-modal policy consensus, 2025. URL https://arxiv.org/abs/2509.23468. 18

[15] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025. doi: 10.1177/02783649241273668. URL https://journals.sagepub.com/doi/10.1177/02783649241273668. 4, 18, 37, 39

[16] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2003354. 41

[17] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations*, volume 2024, pages 2632–2652, 2024. URL https://openreview.net/forum?id=2dnO3LLiJ1. 6

[18] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7. 19

[19] Sander Dieleman. Generative modelling in latent space, 2025. URL https://sander.ai/2025/04/15/latents.html. 19

[20] 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, 2026. URL https://arxiv.org/abs/2602.15397. 2, 3, 4, 5, 6, 9, 18, 19, 34, 37, 39, 42

[21] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-e: An embodied multimodal language model. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/driess23a.html. 18

[22] Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z. Ren, Homer Walke, Quan Vuong, Lucy Xiaoyang Shi, and Sergey Levine. Knowledge insulating vision-language-action models: Train fast, run fast, generalize better. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=cb0xbZ3APM. 4, 10, 18, 37, 38

[23] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://iclr.cc/virtual_2020/poster_SJg7KhVKPH.html. 19

[24] 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, 2026. URL https://arxiv.org/abs/2605.02881. 4, 10, 18

[25] Max Fu, Huang Huang, Gaurav Datta, Lawrence Yunliang Chen, Will Panitch, Fangchen Liu, Hui Li, and Ken Goldberg. ICRT: In-context imitation learning via next-token prediction. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5937–5944. IEEE, May 2025. doi: 10.1109/ICRA55743.2025.11128272. URL https://doi.org/10.1109/ICRA55743.2025.11128272. 3, 18

[26] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994. URL https://dl.acm.org/doi/10.5555/177910.177914. 2, 41

[27] Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6112–6121, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://aclanthology.org/D19-1633/. 19, 34

[28] Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Tan, Lawrence Chen, Quan Vuong, Ted Xiao, Pannag Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.090. 18

[29] Alex Graves. Adaptive computation time for recurrent neural networks, 2016. URL https://arxiv.org/abs/1603.08983. 19

[30] Sigmund H. Høeg, Aksel Vaaler, Chaoqi Liu, Olav Egeland, and Yilun Du. Hybrid diffusion for simultaneous symbolic and continuous planning. *IEEE Robotics and Automation Letters*, 11(4): 4489–4496, 2026. doi: 10.1109/LRA.2026.3664616. 18

[31] Daniel Jiwoong Im, Kevin Zhang, Nakul Verma, and Kyunghyun Cho. Deep autoregressive models as causal inference engines. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=uuREHPf2ll. 19

[32] Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, Vedant Choudhary, Foster Collins, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Maitrayee Dhaka, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachlan Groom, Haroun Habeeb, Hunter Hancock, Karol Hausman, Gashon Hussein, Victor Hwang, Brian Ichter, Connor Jacobsen, Szymon Jakubczak, Rowan Jen, Tim Jones, Gregg Kammerer, Ben Katz, Liyiming Ke, Mairbek Khadikov, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Brendon LeCount, Sergey Levine, Xinyu Li, Adrian Li-Bell, Vladislav Lialin, Zhonglin Liang, Wallace Lim, Yao Lu, Enyu Luo, Vishnu Mano, Nandan Marwaha, Aikys Mongush, Liam Murphy, Suraj Nair, Tyler Patterson, Karl Pertsch, Allen Z. Ren, Gavin Schelske, Charvi Sharma, Baifeng Shi, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Jiaming Tang, Jimmy Tanner, Shalom Tekeste, Marcel Torne, Kyle Vedder, Quan Vuong, Anna Walling, Haohuan Wang, Jason Wang, XuDong Wang, Chris Whalen, Samuel Whitmore, Blake Williams, Charles Xu, Sukwon Yoo, Lili Yu, Wuming Zhang, Zhuoyang Zhang, and Ury Zhilinsky. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities, 2026. URL https://arxiv.org/abs/2604.15483. 18, 38

[33] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Srirama, Lawrence Chen, Kirsty Ellis, Peter Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Ma, Patrick Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Hatch, Shan

Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Walke, Blake Wulfe, Ted Xiao, Jonathan Yang, Arefeh Yavary, Tony Zhao, Christopher Agia, Rohan Baijal, Mateo Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Foster, Jensen Gao, David Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Roy Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.120. 18

[34] Ilyes Khemakhem, Ricardo Monti, Robert Leech, and Aapo Hyvarinen. Causal autoregressive flows. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3520–3528. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/khemakhem21a.html. 19

[35] Jaeyeon Kim, Kulin Shah, Vasilis Kontonis, Sham M. Kakade, and Sitan Chen. Train for the worst, plan for the best: Understanding token ordering in masked diffusions. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 30749–30768. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/kim25ah.html. 19, 34

[36] 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. 2, 3, 18, 39, 41

[37] Alexander Kolesnikov, André Susano Pinto, Lucas Beyer, Xiaohua Zhai, Jeremiah Harmsen, and Neil Houlsby. Uvim: A unified modeling approach for vision with learned guiding codes. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 26295–26308. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/a86b7a9bf7647d6f9f9168d8167d9283-Paper-Conference.pdf. 19

[38] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, and Ali Farhadi. Matryoshka representation learning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30233–30249. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c32319f4868da7613d78af9993100e42-Paper-Conference.pdf. 7, 18

[39] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, Brussels, Belgium, October-November

2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. URL https://aclanthology.org/D18-1149/. 19, 34

[40] Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 26991–27008. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/lee24y.html. 2, 18, 42

[41] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html. 19

[42] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation, 2024. URL https://arxiv.org/abs/2411.19650. 18

[43] Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-language foundation models as effective robot imitators. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=lFYj0oibGR. 18

[44] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard, editors, *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pages 3705–3728. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/li25c.html. 11, 12

[45] Zhixuan Liang, Yizhuo Li, Tianshuo Yang, Chengyue Wu, Sitong Mao, Liuao Pei, Tian Nian, Shunbo Zhou, Xiaokang Yang, Jiangmiao Pang, Yao Mu, and Ping Luo. Discrete diffusion VLA: Bringing discrete diffusion to action decoding in vision-language-action policies, 2025. URL https://arxiv.org/abs/2508.20072. 18

[46] Fanqi Lin, Kushal Arora, Jean Mercat, Haruki Nishimura, Paarth Shah, Chen Xu, Mengchao Zhang, Mark Zolotas, Maya Angeles, Owen Pfannenstiehl, Andrew Beaulieu, and Jose Barreiros. A systematic study of data modalities and strategies for co-training large behavior models for robot manipulation, 2026. URL https://arxiv.org/abs/2602.01067. 18

[47] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 44776–44791. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html. 11

[48] Chaoqi Liu, Haonan Chen, Sigmund H. Høeg, Shaoxiong Yao, Yunzhu Li, Kris Hauser, and Yilun Du. Flexible multitask learning with factorized diffusion policy. *IEEE Robotics and Automation Letters*, 11(4):4697–4704, 2026. doi: 10.1109/LRA.2026.3664611. 18

[49] Chaoqi Liu, Xiaoshen Han, Jiawei Gao, Yue Zhao, Haonan Chen, and Yilun Du. Oat: Ordered action tokenization, 2026. URL https://arxiv.org/abs/2602.04215. 5

[50] Jiaming Liu, Hao Chen, Zhuoyang Liu, Pengju An, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, Chengkai Hou, Mengdi Zhao, KC alex Zhou, Pheng-Ann Heng, and Shanghang Zhang. HybridVLA: Collaborative diffusion and autoregression in a unified vision-language-action model. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=H1KDMNOKQn. 18

[51] 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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=k6nTUFoqeT. 3, 9, 19, 34

[52] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In Aleksandra Faust, David Hsu, and Gerhard Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 1678–1690. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/mandlekar22a.html. 11

[53] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8ishA3LxN8. 2, 4, 6, 17, 18, 39, 42

[54] Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. Quest: Self-supervised skill abstractions for learning continuous control. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 4062–4089. Curran Associates, Inc., 2024. doi: 10.52202/079017-0133. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/076c3e48fa502c660902105965fdd9f6-Paper-Conference.pdf. 2, 5, 6, 18, 39, 42

[55] 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*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.050. 11

[56] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tQJYKwc3n4. 11, 12

[57] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Jim Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan

Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734. 18

[58] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi Jim Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick Tree Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Martín-Martín, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vincent Vanhoucke, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebo-

tar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, May 2024. doi: 10.1109/ICRA57147.2024.10611477. URL https://doi.org/10.1109/ICRA57147.2024.10611477. 18

[59] Norman Di Palo and Edward Johns. Keypoint Action Tokens Enable In-Context Imitation Learning in Robotics. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024. doi: 10.15607/RSS.2024.XX.096. 18

[60] 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. 2, 5, 18, 38, 39, 41

[61] Delin Qu, Haoming Song, Qizhi Chen, Yuanqi Yao, Xinyi Ye, Jiayuan Gu, Zhigang Wang, Yan Ding, Bin Zhao, Dong Wang, and Xuelong Li. Spatialvla: Exploring spatial representations for visual-language-action models. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.011. 18

[62] Vivek Ramanujan, Kushal Tirumala, Armen Aghajanyan, Luke Zettlemoyer, and Ali Farhadi. When worse is better: Navigating the compression generation trade-off in visual tokenization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=o8hWyJIgAV. 42

[63] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1746–1754, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rippel14.html. 7, 18

[64] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162/. 2, 41

[65] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. 4, 7

[66] Andreas Steiner, André Susano Pinto, Michael Tschannen, Daniel Keysers, Xiao Wang, Yonatan Bitton, Alexey Gritsenko, Matthias Minderer, Anthony Sherbondy, Shangbang Long, Siyang Qin, Reeve Ingle, Emanuele Bugliarello, Sahar Kazemzadeh, Thomas Mesnard, Ibrahim Alabdulmohsin, Lucas Beyer, and Xiaohua Zhai. Paligemma 2: A family of versatile vlms for transfer, 2024. URL https://arxiv.org/abs/2412.03555. 12

[67] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html. 5, 19, 34

[68] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent advances in autoencoder-based representation learning, 2018. URL https://arxiv.org/abs/1812.05069. 4, 19

[69] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html. 2, 4, 18, 42

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html. 6

[71] An Dinh Vuong, Minh Nhat Vu, Dong An, and Ian Reid. Action tokenizer matters in in-context imitation learning. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13490–13496. IEEE, October 2025. doi: 10.1109/iros60139.2025.11246836. URL http://dx.doi.org/10.1109/IROS60139.2025.11246836. 18

[72] Homer Rich Walke, Kevin Black, Tony Z. Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, Abraham Lee, Kuan Fang, Chelsea Finn, and Sergey Levine. Bridgedata v2: A dataset for robot learning at scale. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 1723–1736. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/walke23a.html. 18

[73] Yating Wang, Haoyi Zhu, Mingyu Liu, Jiange Yang, Hao-Shu Fang, and Tong He. VQ-VLA: Improving vision-language-action models via scaling vector-quantized action tokenizers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11089–11099, October 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Wang_VQ-VLA_Improving_Vision-Language-Action_Models_via_Scaling_Vector-Quantized_Action_Tokenizers_ICCV_2025_paper.html. 18, 42

[74] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. Tinyvla: Toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025. doi: 10.1109/LRA.2025.3544909. URL https://doi.org/10.1109/LRA.2025.3544909. 18

[75] Xin Wen, Bingchen Zhao, Ismail Elezi, Jiankang Deng, and Xiaojuan Qi. "principal components" enable a new language of images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16641–16651, October 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Wen_Principal_Components_Enable_A_New_Language_of_Images_ICCV_2025_paper.html. 19

[76] Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Se June Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Latent action pretraining from videos. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VYOe2eBQeh. 18

[77] 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 *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1094–1100. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/yu20a.html. 11

[78] Thomas T. Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control, 2025. URL https://arxiv.org/abs/2507.09061. 4, 18, 39

[79] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023. doi: 10.15607/RSS.2023.XIX.016. 4, 18, 39

[80] Yue Zhao, Hanwen Jiang, Zhenlin Xu, Chutong Yang, Ehsan Adeli, and Philipp Krähenbühl. Spherical leech quantization for visual tokenization and generation. *arXiv preprint arXiv:2512.14697*, 2025. 4, 19

[81] Yifan Zhong, Fengshuo Bai, Shaofei Cai, Xuchuan Huang, Zhang Chen, Xiaowei Zhang, Yuanfei Wang, Shaoyang Guo, Tianrui Guan, Ka Nam Lui, Zhiquan Qi, Yitao Liang, Yuanpei Chen, and Yaodong Yang. A survey on vision-language-action models: An action tokenization perspective, 2025. URL https://arxiv.org/abs/2507.01925. 18

[82] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html. 2, 3, 5, 18, 39, 41

Contents

A Block-Wise Autoregressive Training and Inference

This appendix completes the specification of block-wise autoregression (BAR) introduced in Sec. 5.1 and its integration with OAT. It details block patterns, block-shifted teacher forcing, inference, and tokenizer compatibility. Within autoregressive (AR) generation, BAR covers token-wise AR, one-shot parallel decoding (Dong et al., 2026), prior fixed-size block prediction (Liu et al., 2026; Dong et al., 2026), and intermediate schedules, thereby exposing the tradeoff between sequential policy depth and within-block prediction difficulty. Related depth-reduction methods include block-wise parallel decoding, masked prediction, and non-autoregressive generation (Stern et al., 2018; Ghazvininejad et al., 2019; Lee et al., 2018).

A.1 Block Patterns

For a fixed-length action token sequence $T_{1:H_l}$, let $\mathbf{b} = (b_0, b_1, \ldots, b_S)$ denote the endpoint list, with $0 = b_0 < b_1 < \cdots < b_S = H_l$. At stage s, BAR generates the block $G_s = T_{b_{s-1}+1:b_s}$, whose size is

$$g_s = |G_s| = b_s - b_{s-1}.$$

After this stage, the realized prefix is $T_{1:b_s}$. For indexing, we use the empty-prefix convention $G_0 = T_{1:0} = \varnothing$ and set $g_0 = 0$. The endpoint list therefore determines both the number of sequential policy calls S and the number of action tokens predicted in parallel at each stage.

As illustrated in Fig. 5, the BAR family ranges from token-wise autoregression to one-shot parallel decoding, with fixed-size and variable-size blocks between those endpoints. This formulation is consistent with recent evidence that token ordering and decoding order can strongly affect partially parallel generative models (Kim et al., 2025).

For the shifted-slot implementation below, we restrict BAR schedules to nondecreasing generated block sizes,

$$g_1 \le g_2 \le \cdots \le g_S.$$

Early stages keep small blocks, so early positions receive more serial conditioning; later stages save policy calls by predicting larger blocks. In particular, we use the positive endpoints

$$(b_1, b_2, \ldots, b_S) = (1, 2, 4, 8, 16, \ldots, 2^{S-1}).$$

These endpoints induce the block sizes

$$(g_1, g_2, \ldots, g_S) = (1, 1, 2, 4, 8, \ldots, 2^{S-2}).$$

For $H_l = 2^{S-1}$, this schedule reduces the sequential policy depth from $O(H_l)$ to $O(\log H_l)$.

Power-of-two budgets use the fewest stages under a balanced growth constraint. Starting from $b_1 = 1$, require each later generated block to be no larger than the realized prefix. For $s \ge 2$,

$$\begin{aligned} g_s &= b_s - b_{s-1} \le b_{s-1}, \\ b_s &= b_{s-1} + g_s \le 2b_{s-1}. \end{aligned}$$

It follows that S stages reach at most $b_S \le 2^{S-1}$ tokens. Reaching a length-K prefix therefore requires at least $1 + \lceil \log_2 K \rceil$ stages. When K is a power of two, doubling the prefix length at each stage meets this bound exactly.

A.2 Block-Shifted Teacher Forcing

BAR trains each generation stage with the same shifted context structure used at inference. For stage s, define the block-shifted input sequence

$$X^{(s)} = T_{1:b_{s-1}} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}}.$$

The final g_s slots of $X^{(s)}$ are the logit-read slots. When $s > 1$, these slots contain the previous block context followed by padding masks:

$$G_{s-1} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}}.$$

Nondecreasing block sizes ensure that this shifted window can carry the previous block while adding only new mask slots as the block width grows. The policy reads logits from these slots and supervises them against the current block G_s. Let π denote the policy, o its observation context, and $p^{(s)}_{\pi,j}(\cdot \mid X^{(s)}, o)$ denote the categorical distribution read from the j-th final logit-read slot. The loss is

$$\mathcal{L}_{\text{BAR}} = -\frac{1}{S} \sum_{s=1}^{S} \frac{1}{g_s} \sum_{j=1}^{g_s} \log p^{(s)}_{\pi,j}\big(T_{b_{s-1}+j} \mid X^{(s)}, o\big).$$

We therefore average within each block and weight all generation stages equally. This block-shifted objective produces all g_s logits in one policy forward pass while blocking access to ground-truth tokens from the current block. Each logit-read slot sees only realized prefix tokens or masks and is supervised against its target token in G_s; a block-causal attention mask still allows mask-slot states to interact.

A.3 Block-Wise Inference

Inference mirrors the same block structure and shifted slots. Stage 1 feeds g_1 masks and predicts $\widehat{G}_1$. For $s > 1$, the policy uses

$$\widehat{X}^{(s)} = \widehat{T}_{1:b_{s-1}} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}},$$

whose final g_s logit-read slots are

$$\widehat{G}_{s-1} \oplus \langle \texttt{MASK} \rangle^{g_s - g_{s-1}}.$$

The previous block symbols in these slots are shifted context, not newly appended tokens. The policy reads the g_s predictions from the final slots and appends only the predicted block $\widehat{G}_s$ to the prefix. After the final stage, the complete action token sequence is

$$\widehat{T}_{1:H_l} = \widehat{G}_1 \oplus \widehat{G}_2 \oplus \cdots \oplus \widehat{G}_S.$$

The completed sequence can be detokenized by any tokenizer that supports full-sequence decoding. If the tokenizer supports partial decoding, then after stage s the prefix $\widehat{T}_{1:b_s}$ can also be decoded as a lower-budget action chunk. Prefixes at trained reconstruction budgets receive direct decoder supervision and typically reconstruct more accurately than intermediate prefixes.

A.4 Reconstruction Budgets and Generation Endpoints

Tokenizer reconstruction budgets and BAR generation endpoints serve distinct roles. Both evaluated variants train the decoder at reconstruction budgets $\mathcal{K} = \{1, 2, 4, 8, \ldots, H_l\}$. The BAR endpoint list $\mathbf{b}$ specifies the blocks used to reach a target budget K and may contain intermediate endpoints that are not decoder reconstruction points.

For $\texttt{OAT}^{\text{sing}}$, ordinary causal register attention orders individual token positions, and BAR uses singleton blocks with endpoints $(1, 2, \ldots, K)$. Reaching a length-K prefix therefore requires K sequential policy calls. For $\texttt{OAT}^{\text{pow2}}$, the block-causal register groups and BAR endpoints follow the power-of-two reconstruction budgets. With $H_l = 16$, endpoints $(1, 2, 4, 8, 16)$ induce block sizes $(1, 1, 2, 4, 8)$ and require five calls to generate the complete sequence. For either variant, a generated prefix $\widehat{T}_{1:K}$ can be suffix-padded to length H_l with learned mask tokens and passed to the decoder. Inference can then stop and execute the decoded action chunk or continue toward a larger budget. Reconstruction budgets in $\mathcal{K}$ are directly optimized and generally provide stronger reconstruction quality, as summarized in Algo. 2.

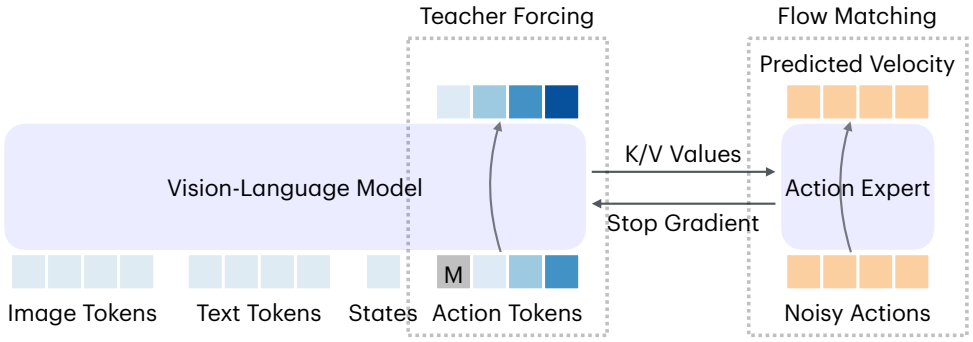


Figure 12: TC computation graph. The VLM predicts targets produced by a frozen `OAT` tokenizer under teacher forcing, and the resulting token loss trains the VLM. Gray "M" denotes the leading mask used to predict the first target. In parallel, the flow-matching expert receives noisy actions and a detached K/V cache from the image, language, and state prefill; the cache excludes the teacher-forced action token positions. At inference, action token logits are not sampled, and the expert generates a continuous action chunk from the cached context.

B Token Co-Training for Vision-Language-Action Control

This appendix expands the token co-training (TC) interface in Sec. 5.2 for vision-language-action (VLA) control. TC combines token supervision for the vision-language model (VLM) with a continuous flow-matching expert. Continuous action generative models are effective for control (Chi et al., 2025; Black et al., 2025), but allowing the corresponding continuous action loss to update a pretrained VLM can degrade its knowledge (Driess et al., 2025). The stop-gradient boundary in Eq. (5.1) isolates the VLM from the flow loss while preserving its context as input to the expert (Driess et al., 2025). Fig. 12 summarizes the resulting computation graph.

Teacher-forced token supervision. Let c denote the image, language, and robot state context. For a continuous action chunk $a_{1:H_a}$, the frozen tokenizer supplies targets $T_{1:H_l} = \mathcal{T}(a_{1:H_a})$. The token loss in Eq. (5.1) is the cross-entropy for predicting each T_i from c and the shifted action token prefix $\langle \texttt{MASK} \rangle, T_{1:i-1}$. The leading mask supplies the prediction slot for T_1, and this branch updates the VLM through action token prediction.

Detached context and flow matching. The expert receives the layer-wise key/value (K/V) cache $\mathrm{KV}_{\mathrm{VLM}}(c)$ from the image, language, and state prefill. This cache excludes K/V values from the teacher-forced action token positions (Dong et al., 2026). Image, language, and state therefore condition the expert through the detached cache rather than as separate raw inputs. Given a normalized action chunk a, Gaussian noise ϵ, and flow time τ, we use

$$a_\tau = \tau\epsilon + (1-\tau)a,$$
$$v = \epsilon - a.$$

Thus the noised action $\tilde{a}^{\tau}_{1:H_a}$ in Eq. (5.1) is a_τ. The expert takes the detached K/V cache, a_τ, a flow-time embedding, and an action padding mask as inputs. Its flow loss is the mean squared error (MSE) between the predicted velocity and v. We use all VLM layers by default, weight both token and flow losses by 1.0, and apply stop-gradient only at the VLM context passed to the expert.

Inference. For each action chunk, the VLM is prefilled once on c and its logits are discarded. Starting from Gaussian noise at $\tau = 1$, the flow-matching expert uses the cached VLM context to

integrate the learned velocity backward to $\tau = 0$ with 10 uniform Euler steps. Executed actions therefore come from the expert rather than detokenization.

In principle, any tokenizer that maps continuous action chunks to discrete symbols can provide TC targets. Prior systems often use Frequency-space Action Sequence Tokenization (`FAST`) (Driess et al., 2025; Black et al., 2025; Intelligence et al., 2026; Pertsch et al., 2025). Within each backbone, our comparison holds the VLM, action expert, cache interface, flow objective, loss weights, and inference solver fixed; only the tokenizer and resulting supervision targets change.

C Experimental Protocol and Implementation

This appendix reports the rollout protocol, tokenizer configurations, policy interfaces, and optimization recipes used in Sec. 6. Full launch-level configuration is provided in the released code.

C.1 Evaluation Protocol

Unless otherwise specified, simulated tasks use 50 evaluation episodes per task and report mean success rate; real-world tasks use 20 independent rollouts and report completed trials. Within each environment, all methods use the same benchmark split, rollout horizon, and success criterion. Policies predict contiguous action chunks with benchmark-specific action dimension D_a. All benchmarks use action horizon $H_a = 32$, except `SimplerEnv`, which uses $H_a = 8$. At test time, only the first half of each predicted chunk is executed before the policy is queried again (Zhao et al., 2023; Chi et al., 2025; Zhang et al., 2025). Because success rates are finite rollout estimates, we interpret very small gaps as practical ties and emphasize consistent trends and benchmark-level differences rather than rank changes from small margins.

C.2 Tokenizer Configurations

The comparison covers the tokenizer families analyzed in Sec. 3. `Bin` discretizes each action dimension into $N = 256$ uniform bins, producing $H_a D_a$ scalar tokens following the action discretization used by Brohan et al. (2023), Zitkovich et al. (2023), and Kim et al. (2024). `FAST` (Pertsch et al., 2025) uses its universal tokenizer with vocabulary size 2048. `QueST` (Mete et al., 2024) compresses action chunks with a temporal convolution of downsample factor 2 before learned tokenization. `ACodec` (Dong et al., 2026) uses learnable registers that cross-attend to the action chunk without attending to one another, forming a one-shot parallel decoding endpoint. `OAT` summarizes action chunks with learnable registers and ordered partial codes; we evaluate the token-wise and power-of-two attention masks in Sec. 4.4.

For `FAST`, an arbitrary byte-pair encoding (BPE) token sequence sampled by an autoregressive (AR) policy may decode to a coefficient stream whose length differs from the fixed topology required by the inverse transform. In simulated rollouts, we use nonstrict decoding unless otherwise specified: the coefficient stream is padded or truncated to the expected length before the inverse discrete cosine transform (DCT). This keeps every sampled sequence executable but may shift frequency-coefficient positions. For physical rollouts, we instead use strict decoding: an invalid sequence is rejected and the policy is queried again.

For fair comparison among learned chunk tokenizers, `QueST`, `ACodec`, and the `OAT` variants use the same encoder and decoder capacity: 6 Transformer layers, 8 attention heads, and model dimension 256. `QueST` and `OAT` use finite scalar quantization (FSQ) with levels $[8, 8, 6, 5]$, corresponding to $8 \times 8 \times 6 \times 5 = 1920 \approx 2048$ discrete codes (Mentzer et al., 2024). For `ACodec`, we follow the official implementation and use vector quantization with latent dimension 512 and codebook size 2048.

C.3 Policy Implementations

The lightweight AR policy uses the `Transformer` backbone: image and robot state observations enter through cross-attention, and the model predicts action token logits directly. Vision-language model (VLM) policies instead place observation, language, and state tokens in the language model context and attach an action token suffix. To make action codes valid VLM outputs, we extend each

VLM tokenizer with the special token set

$$\mathcal{S}_{\text{act}} = \{\texttt{<|action\_}i\texttt{|>} : i \in [|\mathcal{V}_a|]\} \cup \{\texttt{<|action\_mask|>}\}.$$

Each `<|action_i|>` maps to one discrete action code, and `<|action_mask|>` supplies the mask symbol for block-wise autoregression (`BAR`). Prefix VLM backbones such as `PaliGemma2` use full attention over context tokens and causal attention over action suffix tokens, while causal VLM backbones such as `Qwen3VL` use causal attention over both.

AR and token co-training (TC) policies differ in how this suffix is used. In VLM AR policies, the action suffix uses the block-shifted masks from Sec. 5.1 and is detokenized into continuous actions. In VLM TC policies, action token cross-entropy updates the VLM, but the token logits are not sampled at inference. A flow-matching expert executes actions from detached layer-wise VLM key/value (K/V) context, noisy actions, and flow-time embeddings; the flow loss updates only this expert. We use velocity flow-matching with 10 Euler steps.

C.4 Optimization Recipes

Table 8 lists the optimization recipe used by each model family. Learned latent tokenizers use one recipe; all discrete AR policies, including `BAR` patterns, use a single policy optimization recipe independent of generation pattern; and TC policies use a separate recipe for their joint token and flow objective. This presentation keeps the appendix focused on comparability: within a family, differences in closed-loop success should be read against a fixed optimizer, learning rate schedule, warmup, weight decay, gradient clipping, and batch size. In Table 8, Opt., LR, sched., Min, WD, and Clip denote optimizer, learning rate, schedule, minimum, weight decay, and gradient clipping norm, respectively.

Component	Opt.	LR	Batch	LR sched.	Warmup	Min LR ratio	WD	Clip
Latent tokenizers	AdamW	5e-5	512	constant	200	–	0	1.0
AR policies	AdamW	1e-4	16	cosine	500	0.1	1e-6	1.0
TC policies	AdamW	5e-5	32	constant	10k	–	1e-6	1.0

Table 8: Training recipes. Optimization settings for learned latent tokenizers, AR policies, and TC policies. The AR row covers token-wise, fixed-size, one-shot, and variable block patterns. Opt. is the optimizer, LR sched. is the learning rate schedule, WD is weight decay, and clip is the gradient clipping norm.

D Action Tokenizer Baselines and Their Tradeoffs

D.1 Per-Dimension Binning Is Total but Long

Per-dimension `Bin` is the standard total decoding baseline for autoregressive (AR) robot policies (Brohan et al., 2023; Zitkovich et al., 2023; Kim et al., 2024). Each scalar action coordinate is normalized to a fixed range, commonly $[-1, 1]$, divided into N uniform bins, and mapped to the corresponding bin index. For an action chunk of shape $H_a \times D_a$, `Bin` produces the serialization

$$\mathcal{T}(a_{1:H_a}) = [T_{1,1}, \ldots, T_{1,D_a}, T_{2,1}, \ldots, T_{H_a,D_a}].$$

Each coordinate token satisfies $T_{i,j} \in [N]$. If every scalar is emitted as its own token, the resulting token horizon is $H_l = H_a D_a$.

`Bin` is reliable because its detokenizer is simple and total. Every bin index maps back to a scalar action value, so every sampled token sequence of the expected length maps to an action chunk. It also has near-perfect reconstruction as the number of bins grows, up to quantization error. Because the D_a coordinates at one time step can be predicted jointly, `Bin` is compatible with simple block-wise generation.

The drawback is rate and ordering. Common action chunks can require hundreds of tokens, increasing training cost and leaving at least H_a sequential prediction steps under the natural block pattern. The token order is also a manual serialization over dimensions and time: a prefix may contain a few coordinates of the first action step while saying nothing about the rest of the trajectory. Thus `Bin` is reliable as a decoder and supports coordinate blocks, but remains poor as a compact, predictable action representation: it satisfies P.2 and simple block-wise generation, but fails to satisfy P.1 and P.3. The next natural attempt is to compress action chunks while preserving a structured order.

D.2 Frequency-Domain Tokens Are Ordered but Not Total

Frequency-domain tokenizers address the rate and ordering weaknesses of per-dimension binning by representing action trajectories through spectral coefficients (Cooley and Tukey, 1965). Frequency-space Action Sequence Tokenization (`FAST`) is one representative example: it applies the discrete cosine transform (DCT), quantizes the resulting spectral coefficients, and then applies byte-pair encoding (BPE) (Ahmed et al., 1974; Gage, 1994; Sennrich et al., 2016; Pertsch et al., 2025). The resulting order is compatible with AR policies because low-frequency components appear before high-frequency components. Early tokens tend to describe coarse motion structure, while later tokens encode higher-frequency detail. This gives `FAST` a form of AR-friendly ordering and high information density.

The limitation is structural decodability. A robot action chunk requires a fixed coefficient topology before it can be reshaped and transformed back into $H_a \times D_a$ control. After deterministic pruning, quantization, or flattening, the detokenizer expects a coefficient stream of fixed length N, and the positions in this stream are not interchangeable: each index corresponds to a particular frequency basis, temporal component, and action dimension.

BPE breaks this fixed-topology assumption because tokens expand to variable-length coefficient sequences. Let $e(T_i)$ be the coefficient subsequence obtained by expanding token T_i. For a generated sequence $T_{1:L}$, the recovered coefficient stream has length

$$|e(T_1) \oplus e(T_2) \oplus \cdots \oplus e(T_L)| = \sum_{i=1}^{L} |e(T_i)|.$$

Valid decoding requires this length to equal N. Equivalently, the natural domain of the FAST detokenizer is

$$\mathcal{D}_{\text{FAST}} = \{T_{1:L} : \sum_{i=1}^{L} |e(T_i)| = N\},$$

which is a strict subset of the token sequences an AR policy can emit. A next-token policy is not inherently constrained to stay inside $\mathcal{D}_{\text{FAST}}$, so a generated sequence can expand to too few or too many coefficients. In that case, the inverse reshape and inverse frequency transform are mathematically undefined.

This creates a mismatch with fixed-position policy interfaces. Padding or truncation can keep the length valid, but later symbols occupy shifted coefficient slots. Rejection or constrained decoding instead changes what the policy may sample. Thus FAST is compact and ordered, but not total over unconstrained policy outputs. Its variable-length expansion also destabilizes fixed token budgets and block-wise autoregression (BAR) schedules, motivating learned latents with a fixed token horizon and a decoder defined over the full latent space.

D.3 Learned Latents Are Compact but Not Necessarily Predictable

Learned latent tokenizers address compression by learning a neural bottleneck for action chunks. Methods such as Quantized Skill Transformer (QueST) and ActionCodec (ACodec) map an action chunk into a latent sequence of shape $H_l \times D_l$, quantize that latent representation with vector quantization or finite scalar quantization (FSQ), and decode the quantized latents back into continuous actions (Lee et al., 2024; van den Oord et al., 2017; Mentzer et al., 2024; Mete et al., 2024; Wang et al., 2025; Dong et al., 2026). The latent horizon H_l and latent dimension D_l are hyperparameters, often chosen to be much smaller than the raw action dimension $H_a D_a$. For example, an action chunk with horizon $H_a = 32$ can be represented by a latent sequence with $H_l = 8$ tokens.

These tokenizers are attractive from a rate–distortion viewpoint. They can be much more compact than Bin, and their learned decoders are total over the discrete latent space: any valid code index at each latent position can be embedded and decoded into a continuous action chunk. This satisfies P.1 and P.2 under the policy's supported token vocabulary. However, compactness and total decodability do not by themselves make the tokens learnable policy targets (Ramanujan et al., 2025). For sequential latent tokenizers such as QueST, a latent sequence optimized mainly for endpoint reconstruction need not put important motion information early. The policy may then have to predict latent indices in an order that was not designed for next-token learning.

ACodec provides the prior fixed-size block prediction baseline in our comparison. Its latent groups are designed for joint prediction and joint decoding, so the policy emits a fixed block of action tokens per generation step (Dong et al., 2026). This reduces serial policy depth without exposing an AR-friendly prefix order. In our taxonomy, the tradeoff is block modeling difficulty: the policy must infer the tokens in each fixed block together. The remaining gap is an ordered latent code that stays compact and total while giving next-token prediction a more learnable conditional structure.

E Rate–Distortion Across Action Token Budgets

Table 9 gives the numeric values behind the rate–distortion curves in Fig. 7. Full-budget baselines appear as one operating point per benchmark and include per-dimension `Bin`, Frequency-space Action Sequence Tokenization (`FAST`), Quantized Skill Transformer (`QueST`), and ActionCodec (`ACodec`). `OAT` reports partial decodings at $k \in \{1, 2, 4, 8, 16\}$ under token-wise and power-of-two variants. For `FAST`, the token count is the average byte-pair encoding (BPE) length. The table reports the number of tokens (#Tok) and mean squared error (MSE); MSE is scaled by 10^{-3} to keep the table compact; the main-paper figure plots the corresponding raw MSE values on log axes. The table highlights the diagnostic role of rate–distortion: the power-of-two variant closely tracks the token-wise variant across token budgets, so the closed-loop differences should be read as differences in policy prediction difficulty and generation pattern, not as reconstruction failures.

Scheme	LIBERO		RoboMimic		MetaWorld		RoboCasa365		SimplerEnv	
	#Tok	$\text{MSE}_{10^{-3}}$	#Tok	$\text{MSE}_{10^{-3}}$	#Tok	$\text{MSE}_{10^{-3}}$	#Tok	$\text{MSE}_{10^{-3}}$	#Tok	$\text{MSE}_{10^{-3}}$
`Bin`	224	0.0	224	0.0	128	1.2	384	0.0	56	0.00
`FAST`	47.5	0.3	50.9	0.4	20.5	67.9	42.2	0.2	11.0	0.19
`QueST`	16	2.8	16	3.1	16	24.9	16	1.5	4	0.94
`ACodec`	16	0.7	16	2.3	16	8.2	16	1.4	16	0.04
$\texttt{OAT}_1^{\text{sing}}$	1	14.7	1	12.4	1	731.4	1	16.0	1	0.81
$\texttt{OAT}_2^{\text{sing}}$	2	7.3	2	7.8	2	110.0	2	8.9	2	0.35
$\texttt{OAT}_4^{\text{sing}}$	4	3.8	4	5.3	4	38.1	4	5.2	4	0.21
$\texttt{OAT}_8^{\text{sing}}$	8	2.1	8	3.1	8	32.7	8	2.5	8	0.11
$\texttt{OAT}_{16}^{\text{sing}}$	16	1.0	16	1.9	16	32.1	16	1.3	16	0.05
$\texttt{OAT}_1^{\text{pow2}}$	1	16.2	1	12.9	1	734.6	1	16.2	1	0.82
$\texttt{OAT}_2^{\text{pow2}}$	2	7.4	2	7.8	2	107.1	2	8.9	2	0.35
$\texttt{OAT}_4^{\text{pow2}}$	4	3.6	4	5.3	4	44.4	4	4.9	4	0.20
$\texttt{OAT}_8^{\text{pow2}}$	8	1.9	8	3.0	8	27.7	8	2.4	8	0.11
$\texttt{OAT}_{16}^{\text{pow2}}$	16	0.9	16	1.8	16	26.4	16	1.2	16	0.06

Table 9: Rate–distortion of action tokenizers. #Tok is the number of discrete action tokens used for one action chunk; for `FAST`, it is the average BPE length. MSE is measured after detokenization in raw action space and scaled by 10^{-3}; lower is better. Baseline tokenizers are evaluated at their full generated length, while $\texttt{OAT}_k^{\text{sing}}$ and $\texttt{OAT}_k^{\text{pow2}}$ decode only the first k ordered tokens to show token-wise and power-of-two partial reconstruction.

F Autoregressive Action Generation with Vision-Language Models

Table 10 gives the tabular values behind the vision-language model (VLM) policy success plots with autoregressive (AR) action token generation in Fig. 8. The table compares OAT with per-dimension Bin, Frequency-space Action Sequence Tokenization (FAST), Quantized Skill Transformer (QueST), and ActionCodec (ACodec). For FAST, symbolic token and policy call counts depend on the generated byte-pair encoding (BPE) length. In the table, # Tokens and # Calls denote the token count and sequential policy call count, while Avg. denotes average. The table separates the two VLM backbones so that tokenizer effects are compared within a fixed policy model, then reports success across the four benchmark groups and two benchmark-balanced summaries. The LIBERO column averages the Long, Goal, Object, and Spatial suites, matching the aggregation used in the main-paper figure.

PaliGemma2								
Scheme	# Tokens	# Calls	LIBERO	RoboMimic	RoboCasa365	SimplerEnv	Avg. Success	Avg. Rank
Bin	$H_a D_a$	H_a	14.2	7.0	3.2	20.5	11.2	13.3
FAST	$\|T\|$	$\|T\|$	70.2	62.0	44.4	17.0	48.4	9.5
QueST	H_l	H_l	65.9	64.5	49.6	54.0	58.5	5.9
ACodec	H_l	1	78.7	54.0	51.2	26.5	52.6	6.3
$\text{OAT}^{\text{sing}}_{1}$	1	1	28.2	10.0	24.8	16.0	19.8	13.0
$\text{OAT}^{\text{sing}}_{2}$	2	2	72.5	29.0	44.8	35.5	45.5	9.4
$\text{OAT}^{\text{sing}}_{4}$	4	4	73.4	42.0	48.0	35.5	49.7	8.6
$\text{OAT}^{\text{sing}}_{8}$	8	8	78.0	65.0	58.8	39.0	60.2	4.8
$\text{OAT}^{\text{sing}}_{16}$	16	16	79.7	66.0	60.4	48.5	63.7	2.9
$\text{OAT}^{\text{pow2}}_{1}$	1	1	31.4	8.5	28.4	18.0	21.6	12.3
$\text{OAT}^{\text{pow2}}_{2}$	2	2	73.7	24.5	49.2	44.5	48.0	8.0
$\text{OAT}^{\text{pow2}}_{4}$	4	3	76.3	48.0	62.4	49.0	58.9	5.0
$\text{OAT}^{\text{pow2}}_{8}$	8	4	78.2	53.5	62.8	56.5	62.8	3.3
$\text{OAT}^{\text{pow2}}_{16}$	16	5	80.8	60.0	60.4	54.0	63.8	3.0

Qwen3VL								
Scheme	# Tokens	# Calls	LIBERO	RoboMimic	RoboCasa365	SimplerEnv	Avg. Success	Avg. Rank
Bin	$H_a D_a$	H_a	0.0	0.5	1.6	1.0	0.8	14.0
FAST	$\|T\|$	$\|T\|$	62.8	27.5	32.8	5.0	32.0	10.6
QueST	H_l	H_l	58.4	49.0	14.8	6.0	32.1	9.5
ACodec	H_l	1	76.5	63.5	45.2	23.5	52.2	4.5
$\text{OAT}^{\text{sing}}_{1}$	1	1	33.0	12.0	27.2	5.5	19.4	11.8
$\text{OAT}^{\text{sing}}_{2}$	2	2	70.9	29.5	48.4	11.0	40.0	8.8
$\text{OAT}^{\text{sing}}_{4}$	4	4	76.4	34.5	55.2	12.0	44.5	6.8
$\text{OAT}^{\text{sing}}_{8}$	8	8	78.5	52.5	57.6	13.5	50.5	4.1
$\text{OAT}^{\text{sing}}_{16}$	16	16	82.0	67.0	62.0	16.0	56.8	1.5
$\text{OAT}^{\text{pow2}}_{1}$	1	1	31.1	8.5	32.8	3.5	19.0	12.4
$\text{OAT}^{\text{pow2}}_{2}$	2	2	72.7	20.0	52.4	13.0	39.5	8.3
$\text{OAT}^{\text{pow2}}_{4}$	4	3	78.8	31.0	53.2	15.0	44.5	5.6
$\text{OAT}^{\text{pow2}}_{8}$	8	4	79.3	47.0	55.6	19.0	50.2	3.8
$\text{OAT}^{\text{pow2}}_{16}$	16	5	81.8	48.0	57.6	15.0	50.6	3.5

Table 10: Closed-loop VLM AR policy success rates. Rows are grouped by VLM backbone, action token scheme, action token budget, and sequential policy call count. Entries are mean task success rates over 50 rollouts per task. *Avg. Success* gives equal weight to LIBERO, RoboMimic, RoboCasa365, and SimplerEnv; *Avg. Rank* applies the same aggregation across benchmarks to per-benchmark ranks, where lower is better. For baseline tokenizers, # Tokens and # Calls are symbolic because token counts can depend on benchmark action dimensions or generated BPE length; for OAT^{sing} and OAT^{pow2}, they report the evaluated token budget and the number of policy calls required by the generation pattern.