NAT: Scaling Causal Neural Action Tokenizers Across Embodiments

Ahad Jawaid, Alexander Swerdlow, and Yu Xiang

***Abstract*—Action tokenizers let robot policies treat control as autoregressive next-token prediction. As these policies are trained on larger and more diverse datasets, they will need a universal tokenizer that can encode actions from many embodiments and data sources, including ones it has not seen. We study action tokenization at scale, training on 49k hours of action data from real-world robots, handheld grippers, simulation, and humans. We introduce Neural Action Tokenizer (NAT), a causal tokenizer whose decoder lets actions be executed as their tokens are generated. Across four simulation benchmarks, NAT outperforms FAST and prior neural action tokenizers when tokenizing unseen robot embodiments, and in vision-language-action models it outperforms FAST+ across four benchmarks and on a real robot when acting through its tokens. NAT tokens also close the performance gap to a flow-matching action head on some benchmarks while lowering the inference latency for first actions. Our experiments also find that robot data may not be necessary to learn an effective action space: a NAT trained only on human hand motion performs close to tokenizers trained on robot data.**

***Index Terms*—Robot learning, action tokenization, autoregressive policies, vision-language-action models, robot manipulation**

I. Introduction

Large language models [1] have benefited from a simple scalable training recipe: convert text into a shared vocabulary of tokens, then train a single autoregressive model on a large corpus of tokens. A similar recipe may help scale robotic learning. Robot data, however, comes from many embodiments, often with different action spaces, control frequencies, and ranges of motion, and is collected in different ways: teleoperation on real robots, handheld grippers such as UMI [2], simulation, and recordings of human hands. A foundation robot policy that uses this recipe will likely need a tokenizer that can encode large and diverse sets of actions, ideally including those from embodiments it has not seen.

The action tokenizer therefore plays an important role in training autoregressive robot policies, and its design can substantially affect policy performance, training efficiency, and inference speed [3]. Early autoregressive policies discretized each action dimension into uniform bins [4], [5]. Binning can be applied to most robots, but it tends to produce long token sequences that slow down inference. FAST [3] compresses action chunks with a frequency-domain transform and byte pair encoding, yielding shorter sequences and faster training, and the released FAST+ [3] is trained on 1M robot trajectories such that it can work across embodiments. A growing line of work instead learns the tokenizer: a neural encoder, a quantizer [6], [7], and a decoder that reconstructs the action chunk [8], [9], [10], [11], [12]. These tokenizers can represent actions with fewer tokens than FAST [3], but most are trained on comparatively small datasets, so how they behave on the scale and diversity of data a foundation policy would require remains largely unexplored.

In this work, we study action tokenization at scale. We assemble a corpus of 49k hours of action data from 156 datasets spanning real-world robots, universal manipulation interfaces (UMIs) [2], simulated robots, and human hand motion, and use it to train and evaluate action tokenizers. We also introduce the Neural Action Tokenizer (NAT)[1], a causal neural tokenizer that encodes action chunks into short token sequences using finite scalar quantization (FSQ) [7] at multiple temporal scales. Its causal decoder supports streaming: actions can be decoded as their tokens are generated, so the robot can begin executing while the policy is still predicting the rest of the chunk, and the policy can condition its next chunk on the actions still executing. We then pretrain vision-language-action models (VLAs) with NAT to test whether these results carry over to large pretrained policies, and compare acting through NAT tokens with acting through a flow-matching action head [13], [14].

Our contributions are:

- **State-of-the-art neural action tokenization.** NAT outperforms FAST [3] and prior neural action tokenizers (OAT [10], ActionCodec [12], and NAC [11]) in mean success on unseen robot embodiments.
- **A universal streaming action tokenizer.** We train NAT on 49k hours of actions from real-world robots, handheld grippers, simulation, and humans. Its causal decoder supports immediate execution, and its time-ordered tokens let a policy condition its next chunk on the actions still executing without retraining.
- **Human data for action tokenizers.** Our experiments suggest that robot data may not be necessary to train an action tokenizer: a NAT trained only on human hand motion is nearly as effective as one trained on all sources.
- **Pretraining vision-language-action models with NAT tokens.** We show that NAT tokens can help pretrain vision-language-action models: further training a pretrained vision-language model on 1.8k hours of robot data through NAT tokens, before fine-tuning, raises mean success on four simulation benchmarks over fine-tuning the vision-language model directly.
- **Flow action heads and action tokens.** Acting through NAT tokens produces first actions sooner than acting

A. Swerdlow is with Stanford University. Y. Xiang is with The University of Texas at Dallas.

[1]Website: https://www.ahadjawaid.com/nat. Tokenizer: https://huggingface.co/ahad-j/nat-tokenizer-h32-l4-c6400.

through a flow-matching action head, and on some benchmarks matches its success: on LIBERO [15], the two are within the 95% intervals.

II. Related Work

A. Autoregressive Robot Policies

Robot foundation models are commonly built by fine-tuning a pretrained vision-language model on robot data [14], [16], [17], [18]. Since these backbones are autoregressive transformers [19], actions can be tokenized and predicted with next-token prediction; early work showed that even simple binning transfers web-scale knowledge to robot control [4], [5]. Other models, such as π_0 [14] and GR00T [17], instead decode actions with a flow-matching or diffusion head which showed improvements in inference latency. FAST [3] showed that mid-training on discrete action tokens improves training efficiency and policy performance, and later work adopts this for models with a continuous action expert [16], [20], [18].

B. Action Tokenizers

Binning each action dimension [4], [5] is simple but produces long token sequences that can make real-time inference impractical. FAST [3] uses handcrafted frequency-based compression to produce much shorter sequences, improving training efficiency and policy performance, and its variant FAST+ [3], fitted on 1M real robot action sequences, tokenizes new action spaces zero-shot. Learned tokenizers compress actions further with vector-quantized autoencoders [8], and have been extended to multiple embodiments [9], aligned with language [12], ordered so that partial token sequences can be decoded [10], and scaled to 2B action frames with vision-language alignment [21]. Most neural tokenizers have been trained on comparatively smaller datasets, so it remains largely unexplored how learned tokenizers behave when trained on large, heterogeneous action data spanning real robots, UMIs [2], simulation, and humans.

C. Real-Time Execution

Action chunking [22], [23] lets a policy predict several future actions per inference, but the robot must wait for each new chunk, and switching between independently sampled chunks can cause discontinuities. Many VLAs predict chunks with a diffusion or flow-matching action head [14], [17], which generates every action of a chunk in parallel over a fixed number of denoising steps; knowledge insulation [20] trains such a head beside discrete action tokens, and consistency distillation [24] reduces the number of denoising steps. Other work hides inference latency during execution: asynchronous inference [25] predicts the next chunk while the current one runs, bidirectional decoding [26] selects among sampled chunks for coherence with the previous one, and real-time chunking [27] inpaints the actions still executing into a flow policy's sampling so that consecutive chunks join smoothly. Autoregressive policies, in contrast, usually decode a chunk only after all of its tokens are generated [3]. NAT's time-ordered tokens and causal decoder give them both properties: actions are executed as their tokens are generated, and executing actions are fixed as a token prefix.

III. Preliminaries

Action chunking. We consider policies that, given an observation o (camera images, proprioception, and a task), predict a chunk of H future actions $a_{1:H} \in \mathbb{R}^{H \times D}$, where D is the action dimension. During execution, the robot executes the first $s \leq H$ actions of each chunk before querying the policy again. We refer to s as the execution horizon [27].

Action tokenization. Autoregressive policies that predict discrete tokens, the setting we study, require continuous action chunks to first be converted into a token sequence. A tokenizer $\mathcal{T}$ maps an action chunk to a sequence of L tokens, $\mathcal{T}(a_{1:H}) = c_{1:L}$ with $c_i \in \mathcal{V}, i = 1, \ldots, L$, where $\mathcal{V}$ is the tokenizer's vocabulary, and a detokenizer $\mathcal{T}^{-1}$ maps a token sequence back to an action chunk, $\mathcal{T}^{-1}(c_{1:L}) = \hat{a}_{1:H}$. The policy only predicts in the token space, while the robot executes the continuous action reconstructed by $\mathcal{T}^{-1}$.

Existing tokenizers differ in how they construct this mapping. *Binning* [4], [5] normalizes each action dimension to $[-1, 1]$ and assigns each value to one of N uniform bins, giving one token per dimension per timestep ($L = HD$). FAST [3] applies the discrete cosine transform [28] to each action dimension, rounds the resulting coefficients, and compresses them with byte-pair encoding [29]. *Neural tokenizers* [8], [9], [10], [11], [12] instead learn the mapping: an encoder compresses the chunk into a short sequence of latent vectors, a quantizer [6], [7] maps each latent to an entry of a discrete codebook, and a decoder reconstructs the chunk.

Streaming. For most tokenizers, the detokenizer $\mathcal{T}^{-1}$ needs the complete token sequence before it can decode any action, so the robot must wait for the policy to generate all L tokens. We call a tokenizer *streaming* if every prefix of the token sequence already determines a prefix of the action chunk, and those actions do not change as further tokens are generated. With a streaming tokenizer, the robot can begin executing a chunk while the policy is still generating it.

IV. Neural Action Tokenizer

We introduce the Neural Action Tokenizer (NAT), a neural tokenizer whose decoder enables streaming.

A. Architecture

NAT builds on the Neural Action Codec (NAC) [11], an action tokenizer adapted from neural audio codecs [30], [31]. We start from the variant of NAC trained without adversarial losses and build on it with three changes, each guided by an ablation (Section V-D). First, we replace NAC's multi-scale residual vector quantization (RVQ) with a multi-scale variant of finite scalar quantization (FSQ) [7], which substantially improved downstream policy performance. Second, we replace NAC's inverse short-time Fourier transform (ISTFT) output head with a linear layer, which performed comparably with a simpler design. Third, we make the encoder and decoder causal, which is what enables streaming. This caused only a small change in policy performance relative to a non-causal NAT. Figure 1 shows the resulting model, which consists of an encoder E_ϕ, a multi-scale FSQ quantizer, and a decoder D_ψ.

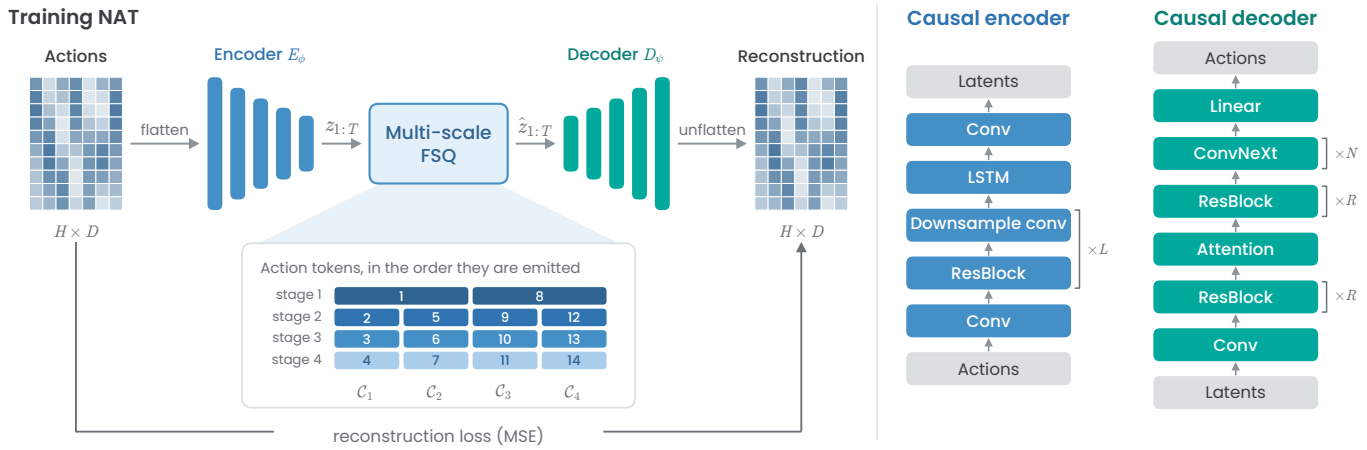


Fig. 1. **NAT architecture.** *Left:* an $H \times D$ action chunk is flattened, encoded into latents $z_{1:T}$, quantized by multi-scale FSQ into coarse-to-fine tokens, and decoded back; NAT is trained end to end with a mean-squared reconstruction loss. The inset shows the 14 tokens of one chunk of the universal NAT ($T = 4$ latents, stage spans $[2, 1, 1, 1]$), numbered in the order they are emitted; $\mathcal{C}_t$ holds the tokens of latent t. *Right:* the causal encoder and decoder layer stacks, read bottom to top; L, R, and N denote repeated blocks (Appendix B).

Encoder. As in NAC [11], we flatten the action chunk time-major into a one-dimensional signal $w = [a_{1,1}, \ldots, a_{1,D}, a_{2,1}, \ldots, a_{H,D}] \in \mathbb{R}^{HD}$, so that each action occupies a contiguous segment. The encoder is a SEANet network [32], [30]: a stack of strided one-dimensional convolutions with residual blocks, followed by an LSTM as in EnCodec [31]. It downsamples w into T latent vectors $z_{1:T} = E_\phi(w)$, each summarizing $k = H/T$ consecutive actions. To make the encoder causal, we pad convolutions only on the left and run the LSTM forward, so latent z_t depends only on the actions $a_{1:tk}$.

Multi-scale FSQ. FSQ [7] quantizes a vector without a learned codebook by bounding each of its d channels and rounding it to one of ℓ_j levels,

$$\mathrm{FSQ}(x)_j = \mathrm{round}\big(h_j \tanh(x_j + \beta_j) - o_j\big), \qquad j = 1, \ldots, d, \tag{1}$$

where $h_j = (\ell_j - 1)/2$; for even ℓ_j, $o_j = 1/2$ and $\beta_j = \mathrm{artanh}(o_j/h_j)$ give ℓ_j levels (else $o_j = \beta_j = 0$). Each rounded vector is one of $V = \prod_j \ell_j$ codes, and its index is the token. During training, gradients are passed through the rounding unchanged (the straight-through estimator).

We apply FSQ in M stages, following residual quantization [6]: each stage quantizes the part of z that the previous stages have not yet captured, called the residual. Following SNAC [33], each stage also works at its own time resolution: stage m averages every p_m consecutive latents into one vector before quantizing, so it produces one token for every p_m latents. Starting from $r_1 = z_{1:T}$, stage m averages the residual, quantizes it, and subtracts the result:

$$\hat{q}_m = W_{\text{out}}^{(m)}\, \mathrm{FSQ}\big(W_{\text{in}}^{(m)}\, \mathrm{Avg}_{p_m}(r_m)\big), \tag{2}$$

$$r_{m+1} = r_m - \mathrm{Repeat}_{p_m}(\hat{q}_m), \tag{3}$$

where Avg_p averages each window of p latents, Repeat_p copies each vector p times to return to one vector per latent, and $W_{\text{in}}^{(m)}, W_{\text{out}}^{(m)}$ are learned linear projections. The quantized latents are the sum of all stages,

$$\hat{z}_{1:T} = \mathrm{MFSQ}(z_{1:T}) = \sum_{m=1}^{M} \mathrm{Repeat}_{p_m}(\hat{q}_m), \tag{4}$$

and we write $c_i^{(m)}$ for the i-th token of stage m.

Decoder. The decoder follows NAC's Vocos-style backbone [34], a stack of convolutional, self-attention, and ConvNeXt [35] blocks operating on the quantized latents. We make it causal in three ways: convolutions are padded only on the left, each latent attends only to itself and a fixed window of preceding latents, and normalization statistics are computed separately for each latent rather than across time. A linear head then maps the decoder output at latent t to the k actions that latent represents. As a result, the reconstructed actions $\hat{a}_{(t-1)k+1:tk}$ depend only on $\hat{z}_{1:t}$.

Training objective. NAT is trained end to end to reconstruct its input. Writing the reconstruction as $\hat{w} = D_\psi\big(\mathrm{MFSQ}(E_\phi(w))\big)$, we minimize the mean squared error

$$\mathcal{L}_{\text{NAT}}(\phi, \psi) = \mathbb{E}_{a_{1:H} \sim \mathcal{D}}\Big[\tfrac{1}{HD}\|\hat{w} - w\|_2^2\Big] \tag{5}$$

over a dataset $\mathcal{D}$ of action chunks.

B. Tokenization and Detokenization

Tokenization. To tokenize a chunk, NAT encodes it and quantizes it with multi-scale FSQ, which gives a set of tokens for every stage. We then interleave the stages into a single sequence in time order, so that tokens describing the beginning of the chunk come first. Concretely, we walk through the latents $t = 1, \ldots, T$ and, at each one, emit every token that latent needs and that has not been emitted yet, from the coarsest stage to the finest. Calling these tokens $\mathcal{C}_t$, the token sequence is

$$\mathcal{T}(a_{1:H}) = [\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_T]. \tag{6}$$

For example, with four stages where the first produces one token per two latents and the rest one token per latent, odd latents contribute four tokens (one per stage) and even latents

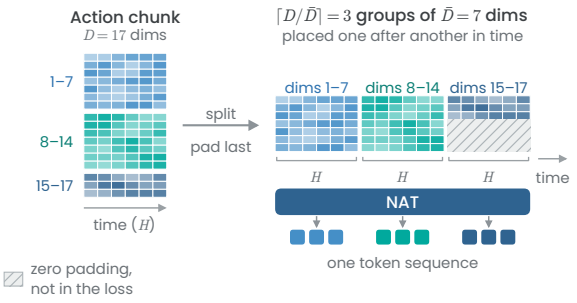


Fig. 2. **Varying action dimensions.** The D action dimensions are split into $\lceil D/\bar{D} \rceil$ groups of $\bar{D}$ slots ($\bar{D} = 7$ here and in our experiments), one per arm where the dataset defines arms and consecutive dimensions otherwise, and the last group is zero-padded. The groups are placed one after another in time, NAT encodes each as an $H \times \bar{D}$ chunk, and their tokens form one sequence. Padded slots are excluded from the reconstruction loss.

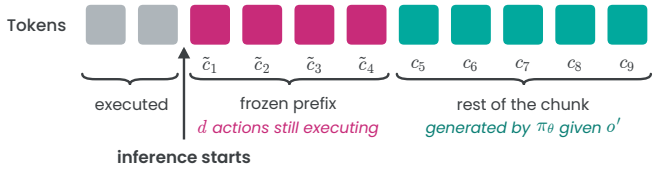


Fig. 3. **Real-time chunking with action tokens.** Gray tokens encode actions already executed. When inference starts, the next d actions will execute before the new chunk is ready, so NAT encodes these actions as the start of the new chunk and freezes the resulting first P tokens ($\tilde{c}_{1:P}$). The policy generates only the remaining tokens $c_{P+1:L}$, given the new observation o'. Because the decoder is causal, the frozen prefix decodes to the actions being executed.

contribute three, since their first-stage token was already emitted with the previous latent. Since recovering a latent requires one token from every stage, the actions of latent t can be decoded only after all tokens in $\mathcal{C}_t$ have been generated. Streaming therefore advances in steps of $|\mathcal{C}_t|$ tokens rather than one token at a time: in this example, the first actions are decoded after four tokens, and each later latent's actions after three or four more.

Varying action dimensions. Robots differ in their number of action dimensions, while NAT encodes chunks of a fixed width $\bar{D}$. We therefore split each action vector into groups of at most $\bar{D}$ dimensions, one per arm where the dataset defines arms and consecutive dimensions otherwise, and zero-pad each group to $\bar{D}$. Each group is tokenized as its own $H \times \bar{D}$ chunk, and the tokens of all groups are concatenated (Figure 2). Padded dimensions are excluded from the loss (Appendix A).

Detokenization. To detokenize, each token is mapped back to its FSQ code, the codes are projected and summed across stages to recover $\hat{z}_{1:T}$, and the decoder produces the action chunk. This ordering is what makes NAT streaming: the first t groups $\mathcal{C}_{1:t}$ contain every token needed to compute $\hat{z}_{1:t}$, and because the decoder is causal, $\hat{z}_{1:t}$ is all it needs to produce the actions of latent t.

C. Autoregressive Policy

Following NAC [11], the policy π_θ is a transformer decoder [19] that generates NAT tokens autoregressively while cross-attending to features of the observation. Tokens from different FSQ stages share the same code indices but carry different information, so we give each stage its own range of V token ids, giving a vocabulary of MV ids plus a beginning-of-sequence token. *NAT is trained first and kept frozen while the policy is trained.* Given a demonstration $(o, a_{1:H})$ with token sequence $c_{1:L} = \mathcal{T}(a_{1:H})$, the policy is trained with the standard next-token prediction loss, averaged over the chunk's tokens,

$$\mathcal{L}_\pi(\theta) = -\,\mathbb{E}_{(o,a_{1:H})\sim\mathcal{D}}\Big[\frac{1}{L}\sum_{i=1}^{L}\log \pi_\theta\big(c_i \mid o, c_{<i}\big)\Big]. \quad (7)$$

Streaming decoding. At inference, the policy generates tokens one at a time. As soon as the tokens $\mathcal{C}_t$ of a latent are complete, NAT decodes that latent's k actions and passes them to the robot, while the policy continues generating the remaining tokens (Algorithm 1). The first actions are therefore available after $|\mathcal{C}_1|$ tokens rather than after all L. After s actions, the policy is queried with a new observation.

Algorithm 1 Streaming NAT policy inference

Require: Observation o; policy π_θ; NAT decoder D_ψ; execution horizon s

$c \leftarrow [\,\texttt{BOS}\,]$
for $t = 1, \ldots, T$ **do**
  for each token position i in $\mathcal{C}_t$ **do**
    Sample $c_i \sim \pi_\theta(\cdot \mid o, c_{<i})$ and append it to c
  Recover $\hat{z}_{1:t}$ from c and decode the actions of latent t with D_ψ
  Send these k actions to the robot for execution
  if $tk \geq s$ **then break** ▷ *the rest of the chunk will not be executed*

Real-time chunking with tokens. While the policy predicts the next chunk, the robot keeps executing the current one, so the next chunk must continue from the actions that run in the meantime. Because NAT's tokens are ordered in time and its decoder is causal, this can be done through the tokens alone, in three steps (Figure 3). (1) Let $\tilde{a}_{1:d}$ be the last d actions of the current chunk that will still execute, with d a multiple of the k actions per latent. NAT encodes them as the start of a new chunk, padded by repeating the last action, and their first P tokens, $\tilde{c}_{1:P}$, form a fixed prefix.(2) Given the new observation o', the policy starts from this prefix and generates only the remaining tokens, $c_i \sim \pi_\theta(\cdot \mid o', c_{<i})$ for $i > P$. (3) Because the decoder is causal, the first d actions of the new chunk depend only on the prefix and approximately reproduce $\tilde{a}_{1:d}$, while the later actions are predicted to continue them. The robot therefore switches to the new chunk after $\tilde{a}_{1:d}$ without less of a jump. This gives the main benefits of real-time chunking (RTC) [27], which inpaints the executing actions into a flow policy's sampling, without any change to training or sampling, since conditioning on earlier tokens is what an autoregressive policy already does. In our NAT VLAs, RTC lowers the mean absolute difference between the last executed action of a chunk and the first of the next (Table XVII).

TABLE I
COMPARISON WITH PRIOR ACTION TOKENIZERS

Method	LIBERO-Long [15]	Meta-World [36]	RoboMimic [37]	RoboCasa [38]	Average
NAT (ours)	$\mathbf{74.4 \pm 3.8}$	38.2 ± 6.6	38.6 ± 2.5	$\mathbf{36.8 \pm 3.3}$	$\mathbf{47.0 \pm 1.5}$
FAST [3]	68.6 ± 0.7	$\mathbf{39.4 \pm 1.4}$	35.1 ± 3.4	30.8 ± 7.9	43.4 ± 1.7
OAT [10]	70.3 ± 2.9	9.2 ± 3.1	$\mathbf{40.0 \pm 3.7}$	26.8 ± 4.7	36.6 ± 2.3
ActionCodec [12]	61.8 ± 3.8	32.2 ± 2.3	39.1 ± 1.6	30.5 ± 4.5	40.9 ± 1.6
NAC [11]	57.2 ± 1.3	7.9 ± 2.5	26.1 ± 4.2	25.0 ± 2.3	29.0 ± 1.2
Diffusion Policy [23]	57.5 ± 4.6	31.8 ± 4.3	29.2 ± 3.5	25.2 ± 5.0	35.9 ± 2.6

Success (%) of one cross-embodiment policy per tokenizer, trained jointly on the four benchmarks.

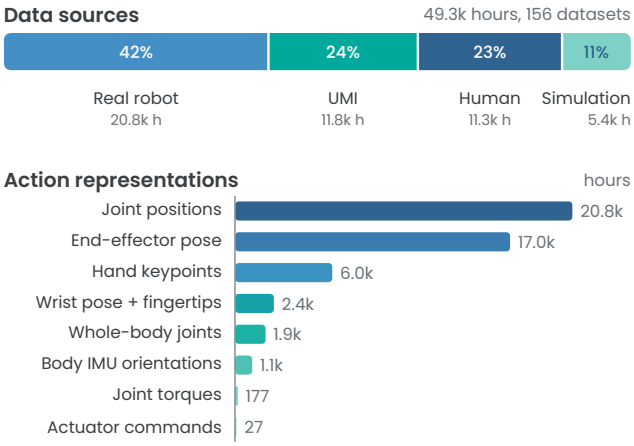


Fig. 4. **Action corpus.** Hours by source (top) and action representation (bottom).

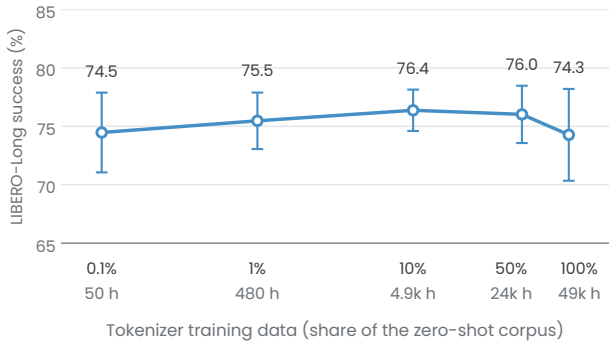


Fig. 5. **NAT is data-efficient.** LIBERO-Long success of a fixed policy as NAT is trained on growing shares of the zero-shot corpus with the same number of updates. Error bars are 95% confidence intervals.

V. EXPERIMENTS

A. Experimental Setup

Action corpus. The corpus has 49k hours (5.5B action frames) from 156 datasets: real-robot teleoperation, simulation, handheld grippers (UMI), and human hands (Figure 4). Each dataset is normalized per dimension to $[-1, 1]$; chunks span $H = 32$ steps in groups of 7 dimensions (Appendix A).

Zero-shot corpus. The corpus excludes the evaluation robots (Franka Panda and Sawyer), so the tokenizers are not trained on them, except in Tables IV and XIII. Unless stated otherwise, tokenizers are trained on a fixed 1% subset.

Tokenizers. We compare NAT with FAST [3], OAT [10], ActionCodec [12], and NAC [11]. All use the same data, a 51,200-token vocabulary, and 28 tokens per chunk (FAST averages 29.8 tokens); the neural tokenizers have 63M parameters.

Policies. Each frozen tokenizer drives the same autoregressive policy (Section IV-C; 162M or 280M parameters, Table XI), which predicts 32-step chunks and executes 16. A diffusion policy [23] with the same backbone serves as a non-autoregressive reference (Appendix B).

Benchmarks. We evaluate on LIBERO-Long [15], Meta-World [36], RoboMimic [37], and RoboCasa [38] with their standard protocols (Appendix C), reporting mean success and 95% confidence intervals over 4 seeds.

B. How Does NAT Compare with Prior Tokenizers?

Prior tokenizers. With one cross-embodiment policy per tokenizer (Table I), NAT has the highest mean success (47.0%), ahead of FAST (43.4%), the other neural tokenizers (29.0–40.9%), and the diffusion policy (35.9%). NAT is best on LIBERO-Long and RoboCasa, FAST on Meta-World, and OAT on RoboMimic. OAT and NAC solve under 10% of Meta-World, whose actions fill a small part of the normalized range and so magnify reconstruction error (Appendix C). With a LIBERO-only policy (Table II), NAT reaches 77.9% (FAST: 66.2%).

Tokens and generation rate. On LIBERO, which runs at 20 Hz, representing one second of action takes 9 tokens with the universal NAT, 24 with FAST+ [3] (2.8× more), and 140 with per-dimension binning (16× more).

C. What Data Does an Action Tokenizer Need?

These experiments train tokenizers on subsets of the zero-shot corpus and evaluate on LIBERO-Long [15], with a LIBERO-only policy for Tables II and III and a policy trained on the four LIBERO suites for Figure 5 (Table XI).

Human data alone. Trained on human hands only (Table II, *Human*), NAT keeps 74.4% and FAST, whose BPE is not learned, is unchanged, while OAT, ActionCodec, and NAC lose 18–33 points. NAT's reconstruction error on LIBERO grows by 22%, versus 27% for OAT and 40% for ActionCodec. A human-only NAT is also close to the all-data NAT with a cross-embodiment policy (Appendix E).

Data mix. Every subset of source types stays within 4.1 points of the full mix and above FAST (Table III). Removing human-hand data causes the largest drop (3.1 points).

TABLE II
TOKENIZERS TRAINED ON ALL DATA OR ON HUMAN DATA ONLY

	Recon. error		Success (%)	
Method	All	Human	All	Human
NAT (ours)	1.44	1.76	$\mathbf{77.9 \pm 1.8}$	$\mathbf{74.4 \pm 2.2}$
FAST [3]	0.95	0.95	66.2 ± 2.1	67.0 ± 3.5
OAT [10]	3.10	3.93	61.5 ± 1.8	38.6 ± 1.3
ActionCodec [12]	2.47	3.45	58.4 ± 1.2	40.9 ± 2.8
NAC [11]	2.56	4.04	57.8 ± 2.8	25.1 ± 1.9

One LIBERO-only policy per tokenizer, evaluated on LIBERO-Long. *All*: tokenizer trained on all source types; *Human*: on human-hand data only. Recon. error: mean absolute error ($\times 10^{-2}$) on LIBERO-10 arm actions.

TABLE III
DATA MIXTURE ABLATION

Source type	Only this type	This type removed
Real robot	77.8 ± 4.0	75.9 ± 3.2
Sim robot	–	77.0 ± 5.1
UMI [2]	76.0 ± 2.4	77.2 ± 3.6
Human hands	74.4 ± 2.2	74.8 ± 2.5
UMI and human hands	–	73.8 ± 2.7

NAT trained on one source type only or with source types removed, in the setup of Table II; with all source types it reaches $77.9 \pm 1.8\%$. Dashes: not run.

TABLE IV
ARCHITECTURE ABLATIONS ON LIBERO-LONG

Component	NAT	Success	Alternative	Success
Quantizer†	MS-FSQ	69.3 ± 2.2	MS-RVQ	60.5 ± 1.2
Output head	Linear	75.8 ± 3.6	ISTFT	75.6 ± 0.5
Causality	Causal	75.8 ± 3.6	Non-causal	77.5 ± 3.1
Scales (tokens)	1 (8)	2 (12)	**4 (28)**	8 (60)
Success	12.9 ± 0.8	56.3 ± 4.3	75.8 ± 3.6	74.9 ± 1.6

Success (%) of NAT's choice and an alternative for each component. †Run on an earlier reference (Appendix D).

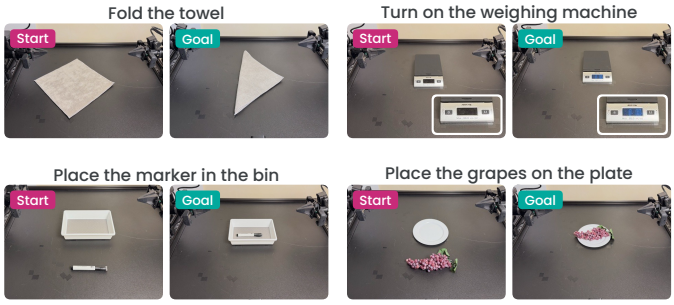


Fig. 6. **Real-world tasks** on the Trossen AI Solo arm: start and goal states.

Data amount. With the policy fixed, training NAT on 0.1% to 100% of the corpus (50 to 49k hours, same number of updates) keeps success between 74.3% and 76.4% (Figure 5).

D. Which Architectural Choices Matter?

Table IV ablates NAT's changes to NAC [11] and the number of scales. Here and in Table XIII only, the 1% subset comes from an earlier corpus that includes the evaluation robots.

Scales and quantization. Success rises from 12.9% with one scale to 75.8% with four, with no gain from eight. Replacing FSQ with NAC's RVQ costs 8.8 points. We suspect two causes, not isolated here: RVQ's learned codebooks are prone to collapse, leaving codes unused [7]; and FSQ's fixed grid makes codes that share most channel levels decode to similar latents, so a policy's confusions between them cost less, while RVQ's codes have no such structure.

Output head and causality. The ISTFT and linear heads perform similarly (75.6% vs. 75.8%), so we use the linear head. A non-causal NAT scores 1.7 points higher, within the 95% intervals. Appendix D gives further sweeps.

E. Does NAT Help Vision-Language-Action Models?

Universal NAT. Our VLAs use a *universal NAT* with 4 latents (14 tokens) per chunk and 6,400 codes per scale, trained for one pass over the full zero-shot corpus (910M windows). Appendix D varies the token and codebook budgets.

We pretrain two VLAs built on LFM2-VL-450M [39], one with our universal NAT (*NAT VLA*) and one with FAST+ [3] (*FAST+ VLA*), otherwise identical. Both train on 4.8B tokens of MolmoAct2's seven-robot data mixture [18] (1.8k hours), which includes DROID [40], BC-Z [41], RT-1 [42], and Bridge-Data V2 [43] (Appendix F). On held-out data, NAT's action error under greedy decoding (temperature 0) is 16% lower (Appendix F). We fine-tune each jointly on four simulation benchmarks and on a real robot.

Multiple benchmarks. We fine-tune each backbone jointly on the four benchmarks of Table I, using all four LIBERO suites. Acting through tokens with the same pretraining and fine-tuning, NAT leads on all four and raises mean success from 37.4% to 61.4% (Table V); on LIBERO it reaches 90.8% against 62.0% for FAST+, while on Meta-World the intervals overlap. Without robot pretraining, the NAT VLA reaches 54.9%, still above the pretrained FAST+ VLA; robot pretraining through NAT tokens adds 6.5 points (4.8 on LIBERO), with non-overlapping intervals on Meta-World and RoboMimic.

Real robot. We fine-tune both VLAs on 481 teleoperated demonstrations on a Trossen AI Solo arm. The universal NAT's corpus includes our own Trossen AI recordings. The four tasks are folding a towel, turning on a scale, putting a marker in a bin, and placing grapes on a plate (Figure 6). Each policy runs 10 trials per task from the demonstrations' median start pose, each scored by an operator (Appendix F). NAT succeeds in 55% of trials and FAST+ in 35% (Table VII). NAT leads on three of the four tasks. FAST+ outperforms NAT on placing grapes (70% vs. 20%), although NAT's reconstruction error on the grape demonstrations is small (MSE 0.0005) and its policy's greedily sampled actions have lower reconstruction error than FAST+'s (MSE 0.021 vs. 0.038). In failed trials NAT VLA did not appear to lower the gripper far enough and closed above the grapes; because each chunk's 16 actions execute open loop, such an offset may go uncorrected.

TABLE V
ONE VLA FINE-TUNED ON FOUR BENCHMARKS

Tokenizer	Acts via	LIBERO	Meta-World	RoboMimic	RoboCasa	Mean
NAT (ours)	Tokens	$\mathbf{90.8 \pm 2.6}$	$\mathbf{22.8 \pm 2.2}$	$\mathbf{81.2 \pm 2.4}$	$\mathbf{50.9 \pm 3.3}$	**61.4**
FAST+	Tokens	62.0 ± 4.2	20.9 ± 2.4	29.1 ± 2.8	37.6 ± 3.2	37.4
NAT, no robot pretraining	Tokens	86.0 ± 2.8	15.6 ± 1.6	72.5 ± 2.6	45.4 ± 3.1	54.9
NAT (ours)	Flow expert	$\mathbf{89.2 \pm 2.8}$	$\mathbf{45.7 \pm 2.5}$	$\mathbf{89.8 \pm 2.0}$	$\mathbf{69.5 \pm 3.0}$	**73.6**
FAST+	Flow expert	89.0 ± 2.5	43.9 ± 2.5	84.7 ± 2.3	67.9 ± 3.0	71.4

Success (%) at 50k steps with 95% bootstrap intervals, acting through action tokens or a flow-matching expert. All rows use robot-pretrained backbones except *no robot pretraining*, which is fine-tuned from the base LFM2-VL-450M with the same recipe. Bold: best within each group.

TABLE VI
ACTION LATENCY (MS) ON AN RTX 4090

				Action tokens		Flow head
Model	Tokenizer	Size	Per token	First actions	Full chunk	Full chunk
NAT VLA (ours)	NAT	0.45B	2.0 ± 0.0	$\mathbf{19.0 \pm 0.5}^*$	$39.5 \pm 0.6^*$	42.0 ± 0.9
FAST+ VLA	FAST+	0.45B	2.2 ± 0.0	97.5 ± 37.6	97.5 ± 37.6	$\mathbf{46.5 \pm 12.6}$
GR00T N1.7 [17]	NAT†	3B	4.1 ± 0.0	$\mathbf{44.7 \pm 1.0}$	85.2 ± 1.0	56.6 ± 1.2
$\pi_{0.5}$ [16]	NAT†	3B	6.5 ± 0.0	$\mathbf{69.0 \pm 1.7}$	134.5 ± 1.7	72.6 ± 0.2
MolmoAct2 [18]	NAT†	5B	12.6 ± 0.0	$\mathbf{100.8 \pm 0.3}$	226.4 ± 0.4	132.4 ± 1.3

Batch size 1, CUDA graphs; mean $\pm$ s.d. over 50 frames for our models and 30 runs for the others. *First actions*: 8 actions from 4 NAT tokens; *full chunk*: 14 NAT tokens. Bold: faster route to the first actions. *Includes detokenization. †Latency analysis only: these models are not trained with NAT; we time 4 and 14 decoding steps.

TABLE VII
VLA SUCCESS ON A REAL TROSSEN AI SOLO ARM

	Fold towel	Turn on scale	Marker in bin	Grapes on plate	Mean
NAT (ours)	**90**	**70**	**40**	20	**55**
FAST+	50	20	0	**70**	35

Success (%) over 10 trials per task. The tasks are shown in Figure 6.

F. How Much Does a Flow Action Head Help?

Many VLAs add a flow-matching action head with the aim of reducing latency and discretization error, so we evaluate the performance of one on top of our VLA. Following the MolmoAct2 recipe [18], we add a flow-matching action head [13] trained with knowledge insulation [20]: the head attends to the language model's activations, but its gradients do not reach the language model, so the backbone keeps training on action tokens. Each pretrained model is first mid-trained with the head and then fine-tuned with tokens and head together (Appendix F).

Latency. FAST+ uses 6 to 73 tokens per chunk (38.8 on average) and decodes only after the whole chunk. NAT uses 14, and its first 4 decode to 8 actions, which take 267 ms to execute at 30 Hz; the rest of the chunk, and the next one (Section IV-C), are generated while the robot moves. Table VI times this on NAT VLA (50 samples). It also times the same decoding steps on the language models of larger VLAs, as a latency analysis only. NAT VLA has its first actions after 19.0 ms, 5.1$\times$ sooner than FAST+ VLA and 2.2$\times$ sooner than its own flow expert. On these 3–5B models, generating NAT's first 4 tokens would take 45–101 ms, comparable to or faster than their action heads, and a full chunk 85–226 ms, under the 267 ms the first actions take to execute.

Success and cost. We fine-tune each backbone with the flow head on the four benchmarks of Table V. These runs add the mid-training stage and use MolmoAct2's post-training learning rates, so they differ from the token runs in more than the action head. Through the expert, NAT VLA reaches 73.6% mean success and FAST+ VLA 71.4% (Table V); the intervals overlap except on RoboMimic. The expert raises NAT's mean by 12.2 points, mostly on Meta-World and RoboCasa, and FAST+'s by 34.0. On LIBERO, NAT tokens and the expert are within the 95% intervals (90.8% vs. 89.2%). The expert adds 77.7M parameters, a mid-training stage, and a denoising process at inference, while NAT's first actions arrive sooner than a chunk from our flow head (19.0 vs. 42.0 ms) and, in our latency analysis, than the heads of GR00T N1.7 and MolmoAct2 (Table VI), and its tokens keep the streaming and real-time chunking of Section IV-C.

VI. DISCUSSION

Our results support four takeaways. First, a single neural action tokenizer can serve robots it has never seen, outperforming FAST and prior neural tokenizers without training on the evaluation robots. Second, an action tokenizer may not need robot data: human hand motion alone yields a tokenizer close to one trained on all sources, so large-scale human data could be used to train action tokenizers. Third, NAT tokens are an effective action representation for vision-language-action models, both for robot pretraining and for acting, where they outperform FAST+. Fourth, action tokens can be faster than flow action heads, and on some domains acting through tokens alone closes the performance gap to a flow head, though a gap remains on others.

Limitations and future work. Acting through NAT tokens still trails a flow-matching head on some benchmarks; further tuning and design choices, such as the token budget, decoding,

and fine-tuning recipe, may close this gap fully. NAT encodes every chunk in a fixed number of tokens; a future study could set the number of tokens per chunk dynamically. NAT is also trained on a single chunk length, so tokenizing chunks of varying length remains to be explored. Our data-scaling experiment scales the tokenizer's training data while keeping the policy's LIBERO data fixed; a future study could explore the benefits of scaling the tokenizer corpus together with the policy's training corpus.

APPENDIX A
ACTION CORPUS

The corpus has 156 datasets (49.3k hours, 5.5B action frames) from four source types (Figure 4). Each dataset is normalized per dimension to $[-1, 1]$ with min–max statistics from its training recordings, and its actions are split into groups of at most 7 dimensions (Section IV-B). Padded dimensions and frames past the end of an episode are excluded from the loss. Dataset k is sampled with probability proportional to N_k^τ, where N_k is its number of frames and τ a temperature. Validation and test sets (0.1% each, from held-out recordings) are excluded from training, and no dataset contains the evaluation robots (Franka Panda and Sawyer). Table VIII lists the collections in each source type. All collections were public when accessed, except our own Trossen AI recordings (5.3 hours) and two proprietary sets of human hand recordings: 1,956 hours from MicroAGI and 72 hours from DeepReach AI, both 21-joint hand poses from head-mounted capture.

APPENDIX B
HYPERPARAMETERS

Tables IX–XI list the tokenizer architecture, the tokenizer training runs, and the policies.

TABLE IX
NAT ARCHITECTURE

	Standard	Universal
Tokenization		
Chunk	$H = 32$ steps $\times$ 7 dimensions	
Latents per chunk T	8 (4 steps each)	4 (8 steps each)
Latent dimension	512	
Quantizer	Multi-scale FSQ, $M = 4$ scales	
Scale spans	$[2, 1, 1, 1]$ latents	
FSQ codes per scale	12,800	6,400
Tokens per chunk	28	14
Encoder (0.9M parameters)		
Downsampling, $L = 2$	strides 4, 7	strides 8, 7
Recurrent layer	2-layer causal LSTM	
Decoder (62M parameters)		
Width	768	
ResBlocks around attention	$R = 2$ before and 2 after	
Causal attention window	8 latents	
ConvNeXt blocks	$N = 12$	
Output values per latent	28	56

Standard: the NAT of Tables I–III. Universal: the NAT of the VLAs (Section V-E). Codes per scale for the other runs are in Table X. L, R, and N correspond to Figure 1.

TABLE X
TOKENIZER TRAINING

	Standard	Data scaling	Ablations
Used in	Tables I–III	Figure 5	Tables IV, XIII
Data	1% of recordings	0.1–100%	1% of recordings*
Sampling temp.	$\tau = 0.5$	$\tau = 1$	$\tau = 0.5$
FSQ codes per scale	12,800	512	512 or 2,048
Batch size	1,024	1,024	256
Steps	30,000	165,947	30,000
Warm-up steps	9,000	49,784	9,000

*Of a larger corpus that also includes the evaluation robots.

Baselines. OAT, ActionCodec, and NAC keep their original configurations but are matched to NAT's data, 28 tokens per chunk, 51,200-token vocabulary, and 62–63M parameters. FAST's 51,200-token BPE vocabulary is fitted on the same corpus.

APPENDIX C
EVALUATION PROTOCOLS

All benchmarks follow their reference protocols with 4 evaluation seeds (0–3) and 50 trials per task per seed. **LIBERO-Long**: 10 tasks with the official initial states. **Meta-World**: box-close, coffee-pull, disassemble, and stick-pull (800 scripted demonstrations). **RoboMimic**: Lift, Can, Square, and ToolHang (800 proficient-human demonstrations). **RoboCasa**: CloseDrawer, TurnOffSinkFaucet, CoffeePressButton, and TurnOffMicrowave (216 human demonstrations). *Mean* averages the four benchmarks within each seed and reports the mean and 95% confidence interval over seeds.

Meta-World's actions are normalized by their training limits, which squeezes the executed commands into about a fifth of the tokenizer's range and magnifies reconstruction error. The reconstruction error in Table II is the round-trip mean absolute error on LIBERO-10 chunks, averaged over position and rotation dimensions in normalized units.

APPENDIX D
DESIGN SWEEPS

Token budget and codebook size. Table XIV varies the number of latents per chunk, which sets the number of tokens, and the number of codes per scale. Halving the tokens from 28 to 14 does not reduce success here (74.4% vs. 75.8%), but it does in the 512-code sweep of Table XIII (76.6% vs. 73.3%), so the effect may depend on the codebook size or data. The 14-token NAT reaches 78.0% when trained on the full corpus (the universal NAT also uses 14 tokens, but 6,400 codes and a longer training run). Halving again to 7 tokens drops success to 55.1%, as LIBERO arm reconstruction error rises from 0.025 to 0.040. Codebook size has little effect from 512 to 12,800 codes in our runs (Table XIII; 74.2% with 6,400 and 75.8% with 12,800 here), but 256 codes reduce success to 64.5%.

These ablations train each tokenizer on the zero-shot corpus for 30,000 steps of 1,024 windows, then one 162M cross-embodiment policy, evaluated on LIBERO-Long over 4 seeds.

Universal NAT. The universal NAT trains for 222,289 steps of 4,096 windows, with learning rate 2×10^{-4} and cosine decay.

TABLE VIII
ACTION CORPUS BY SOURCE TYPE

Source type	Datasets	Hours	Collections
Real robot	86	20,821	Open X-Embodiment [44], BridgeData V2 [43], BC-Z [41], QT-Opt [45], Language-Table [46], RH20T [47], AgiBot World [48], RoboMIND [49], RoboMIND 2.0 [50], RoboCOIN [51], OpenNeo [52], ABC-130k [53], AIRoA MoMa [54], Galaxea Open-World [55], MolmoAct2 [18], WEB [56], OpenArms [57], HIW-500 [58], GR00T teleoperation [17], ActionNet [59], Humanoid Everyday [60], Unitree G1/Z1 [61], ETRI G1 [62], LeRobot community [63], Open-H [64], TriFinger RRC [65], DexWild [66], HRDexDB [67], TDex [68]/TAVI [69], TWIST2 [70], ReFlex [71], EgoVerse [72], EgoSteer [73], PH^{2}D [74], Trossen AI (ours)*
Simulation	18	5,408	InternData [75], BEHAVIOR-1K [76], GraspXL [77], GR00T simulation [17], HOP [78], COMET [79], GenieSim 3 [80], AgiBot World 2026 [81], DexTrack [82], LEAP-Lift [83], TriFinger RRC [65], Open-H [64], PH^{2}D [74]
UMI	7	11,755	GenRobot [84], OpenNeo [52], HY-Embodied [85], HiFi-UMI [86], FastUMI-100K [87]
Human	45	11,347	EgoVerse [72], EgoSteer [73], OpenEgo [88], VITRA [89], MicroAGI*, DeepReach AI*, EgoDemo [90], Assembly101 [91], PH^{2}D [74], OMG [92], Chingmu MoCap [93], Tracker POV [94]
Total	156	49,331	

Dataset counts treat each embodiment or recording variant of a collection separately. *Proprietary.

TABLE XI
POLICIES

	LIBERO-only	Cross-embodiment	LIBERO suites
Used in	Tables II, III	Table I	Fig. 5, Tables IV, XIII
Training data	LIBERO-10	Four benchmarks	Four LIBERO suites
Parameters	280M	162M	162M
Width / layers / heads	1,024 / 16 / 16	768 / 16 / 12	768 / 16 / 12
Steps	25,000	50,000	25,000
Batch size	256	256	256
Shared by all policies			
Observations	Last 2 steps; ResNet-18 with spatial softmax per camera, linear proprioception, frozen BERT instruction		
Execution	32-step chunk decoded greedily, first 16 steps executed		
Vocabulary	51,200 tokens for compared tokenizers; else $M\times$ codes (Table X)		

The human-only cross-embodiment policy of Table XV uses the LIBERO-only architecture (280M) for 25,000 steps.

Architecture ablations. Tables IV and XIII train tokenizers on 1% of an earlier corpus that includes the evaluation robots (30,000 steps, batch 256), then a 162M policy on the four LIBERO suites (25,000 steps), evaluated on LIBERO-Long over 4 seeds. Each block changes one setting of the reference NAT, which used 2,048 codes per scale (512 for the scale-layout and frequency sweeps).

The quantizer comparison used an earlier reference that delays the coarsest token until both latents it covers are seen, so that no token depends on future actions. Without this delay, the coarsest token can depend on up to 4 future steps when tokenizing a ground-truth chunk, but not when decoding generated tokens. The RVQ variant uses NAC's commitment loss.

APPENDIX E
HUMAN-ONLY NAT ON MULTIPLE BENCHMARKS

Table XV repeats the multi-benchmark comparison with a NAT trained only on human-hand data. Both NATs drive the same cross-embodiment policy, which uses the LIBERO-only architecture for 25,000 steps (Table XI), so success is lower than in Table I. The human-only NAT is within 0.8 points of the all-data NAT on every benchmark (mean 38.0% vs. 38.5%).

TABLE XV
NAT TRAINED ON HUMAN-HAND DATA ONLY

Benchmark	All source types	Human hands only
LIBERO-Long	53.0 ± 4.1	52.9 ± 2.0
Meta-World	32.4 ± 1.4	31.8 ± 2.9
RoboMimic	34.0 ± 2.5	33.2 ± 2.6
RoboCasa	34.6 ± 3.5	34.0 ± 6.2
Mean	38.5 ± 1.9	38.0 ± 1.8

Success (%) in the multi-benchmark setting of Table I.

APPENDIX F
VISION-LANGUAGE-ACTION MODELS

Pretraining. NAT VLA and FAST+ VLA are LFM2-VL-450M models pretrained with MolmoAct2's trainer and data pipeline [18] on its seven-robot mixture (Table XVI): 1,843 hours in 333,729 episodes, of which 16,184 are held out. The two runs differ only in the tokenizer. Both train for 95,315 steps on batches of 12 packed 4,200-token sequences (4.8B tokens), with learning rates 5×10^{-5} (language model) and 2.5×10^{-5} (vision encoder and connector). Actions are 32-step chunks at each dataset's native rate, normalized to $[-1, 1]$, and both tokenizers share 25,600 added action-token embeddings.

TABLE XII
PRETRAINED VLA BACKBONES ON HELD-OUT ROBOT DATA

		NAT (ours)			FAST+		
Robot	Steps	PPL	Greedy	Sampled	PPL	Greedy	Sampled
YAM (bimanual)	30	27.8	**0.0250**	0.0985	7.5	0.0314	**0.0419**
SO-100/101	30	70.9	**0.0067**	0.0307	11.1	0.0078	**0.0071**
MolmoAct Franka	10	91.5	**0.0551**	**0.1163**	10.7	0.0784	0.1236
DROID	15	11.3	**0.0479**	**0.1000**	6.9	0.0579	0.1087
BC-Z	10	274.8	0.1280	0.2222	25.7	**0.1216**	**0.1277**
RT-1	3	94.9	**0.1009**	0.2088	8.1	0.1284	**0.1107**
Bridge	5	28.6	0.1075	**0.1702**	3.5	**0.1068**	0.2357
Mix-weighted		31.7	**0.0337**	0.0867	8.4	0.0400	**0.0622**

Greedy/*Sampled*: action MSE under greedy and temperature-1 decoding over MolmoAct2's horizon (*Steps*); lower is better.

TABLE XIII
NAT DESIGN SWEEPS ON LIBERO-LONG

Sweep	Setting	Success (%)
Codes per scale	512	76.6 ± 1.4
	1,024	75.2 ± 2.3
	2,048	75.8 ± 3.6
	4,096	76.5 ± 2.8
	8,192	75.0 ± 4.2
Scale layout (512 codes)	$[2, 1, 1, 1]$, 28 tokens	76.6 ± 1.4
	$[2, 1, 1]$, 20 tokens	68.2 ± 3.8
	$[2, 1, 1, 1]$, 4 latents, 14 tokens	73.3 ± 0.9
	Pyramid $[4, 2, 1]$, 14 tokens	65.1 ± 3.0
	Pyramid $[8, 4, 2, 1]$, 15 tokens	61.1 ± 3.0
	Pyramid $[16, \ldots, 1]$, 31 tokens	70.2 ± 1.8
	Pyramid $[32, \ldots, 1]$, 63 tokens	67.6 ± 3.9
Policy size	162M	75.8 ± 3.6
	21M	67.1 ± 3.4
	40M	72.8 ± 2.0
	81M	75.5 ± 1.8
Tokenizer size	63M	75.8 ± 3.6
	5M	69.8 ± 2.5
	124M	75.8 ± 2.5
	243M	77.2 ± 2.6
Action frequency (512 codes)	Native rates	76.6 ± 1.4
	Resampled to 20 Hz	74.0 ± 2.3
	Resampled to 30 Hz	70.6 ± 1.7

Each block varies one setting; scales are spans over a chunk's latents.

TABLE XIV
TOKEN BUDGET AND CODEBOOK SIZE ON LIBERO-LONG

Tokens	Latents	Codes	Success (%)
28	8	12,800	74.4
14	4	12,800	**75.8**
14	4	6,400	74.2
7	2	12,800	55.1
28	8	256	64.5

NAT trained on 1% of the zero-shot corpus.
Tokens per 32×7 chunk; codes per scale.

NAT VLA uses the universal NAT (14 tokens per group of up to 7 dimensions); FAST+ VLA uses the released FAST+ tokenizer on the whole chunk. Both are trained with MolmoAct2's language-model loss rather than Eq. (7): the loss covers each sample's answer, an action-start token, the action tokens, an action-end token, and an end-of-sequence token; each sample's token losses are weighted by $1/\sqrt{N}$ for its N answer tokens, with the weights normalized over the batch; and a z-loss [95] with coefficient 10^{-4} is added.

TABLE XVI
VLA PRETRAINING MIXTURE

Robot	Rate	Episodes	Held out	Hours
YAM (bimanual) [18]	30%	32,246	2,386	704.1
SO-100/101 [63]	30%	36,637	1,085	177.1
MolmoAct Franka [96]	2.5%	10,488	11	25.2
DROID [40]	30%	74,604	2,651	328.9
BC-Z [41]	2.5%	39,350	2,938	152.0
RT-1 [42]	2.5%	87,212	3,985	350.6
Bridge [43]	2.5%	53,192	3,128	105.2
Total		333,729	16,184	1,843.0

Rate: sampling rate in the mixture. Episodes and hours include the held-out episodes.

Held-out evaluation. Both models are evaluated at their final checkpoint on the same 128 held-out frames per robot (Table XII). We report perplexity per action token and the action mean squared error under greedy and temperature-1 decoding, in normalized units over MolmoAct2's horizon. The mix-weighted error weights robots by their sampling rates.

With greedy decoding, NAT has lower action error on five of seven robots and in mix-weighted error. With temperature-1 sampling, FAST+ is lower on four robots and in mix-weighted error, so sampling raises NAT's error more than FAST+'s.

Real-world evaluation. Both models are fine-tuned for 10,000 steps (batch 64) on 481 teleoperated demonstrations on a Trossen AI Solo arm, recorded at 30 Hz with top and wrist cameras and absolute joint targets for six joints and a gripper. Both decode greedily, predict 32-step chunks, and execute 16 at 30 Hz. The next chunk is predicted while the last 6 actions of the current one run. For safety, a chunk with a jump above 0.3 rad between consecutive actions is not executed. Each policy ran 10 trials per task, each up to 33 s from the demonstrations' median start pose and scored by an operator.

Latency. Table VI times every model on one RTX 4090 in bfloat16 at batch size 1. Our models are timed on 50 distinct LIBERO frames; each baseline uses its released checkpoint and example input, timed 30 times after 3 warm-up runs. At batch size 1, eager PyTorch is dominated by kernel launches, so every model's decoding and action head are replayed as

CUDA graphs, which produce the same outputs as eager execution. The baselines are GR00T N1.7 (its 16-layer language model, with Qwen3-VL-2B in place of the gated backbone of the same architecture), $\pi_{0.5}$ (`pi05_libero`), and MolmoAct2 (`MolmoAct2-LIBERO`). Prompts are 237 tokens for our model, 469 for GR00T N1.7, 497 for MolmoAct2, and 968 for $\pi_{0.5}$. Baseline token timings exclude NAT's detokenization (about 3 ms). FAST+ VLA uses 6 to 73 tokens per chunk, so its latency varies (median 85.3 ms, 90th percentile 146.8 ms).

Multiple environments. Each pretrained model is fine-tuned on action tokens only for 50,000 steps at batch 64, with learning rates 5×10^{-5} (language model) and 2.5×10^{-5} (vision encoder and connector), on an equal mixture of LIBERO (MolmoAct2's four-suite dataset), Meta-World, RoboMimic, and RoboCasa demonstrations. Policies predict 32-step chunks, decode greedily, and each simulator executes the first 10 actions of every chunk. All results are at step 50,000. LIBERO uses 400 episodes (10 tasks per suite, 10 initial states each), RoboMimic and RoboCasa 800 each, and Meta-World 1,000 (5 seeds), with 95% bootstrap intervals. The *no robot pretraining* row uses the same recipe from the released LFM2-VL-450M, with new action-token embeddings.

Flow expert. For the flow-expert rows of Table V, each pretrained model is first mid-trained for 23,829 steps with MolmoAct2's knowledge-insulation recipe [20], which adds a 77.7M-parameter flow-matching expert [13] (width 448, one block per language-model layer) whose gradients do not reach the language model. It is then fine-tuned on the same clipped mixture for 50,000 steps at batch 64 with MolmoAct2's post-training recipe: learning rates 10^{-5} for the language model, 5×10^{-6} for the vision encoder and connector, and 5×10^{-5} for the expert, with 8 flow times per chunk and the action tokens trained beside the expert. It is evaluated through the expert with 10 Euler steps, under the same protocols as the token runs.

Real-time chunking. Table XVII compares, for the NAT VLA, the difference between consecutive executed actions at chunk switches with and without RTC.

TABLE XVII
DIFFERENCE BETWEEN CONSECUTIVE ACTIONS

	Across a chunk switch		Within
NAT VLA	With RTC	Without RTC	one chunk
Fine-tuned	0.065 ± 0.007	0.126 ± 0.010	0.031 ± 0.003
Pretrained	0.0096 ± 0.0007	0.046 ± 0.009	0.0024 ± 0.0002

$|a_t - a_{t-1}|$ averaged over arm dimensions (in $[-1, 1]$), steps, and benchmarks; $\pm$: 95% bootstrap interval. *Across a chunk switch*: from the last executed action of a chunk to the first of the next; *within one chunk*: two consecutive actions of the same chunk. Fine-tuned: LIBERO, RoboMimic, Meta-World; pretrained: held-out Bridge, RT-1, BC-Z, SO-100/101.

ACKNOWLEDGMENT

We thank MicroAGI and DeepReach AI for providing access to their human hand-motion data, and Chaoqi Liu for helpful early conversations.

REFERENCES

[1] S. Minaee *et al.*, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.
[2] C. Chi *et al.*, "Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots," in *Proc. Robot.: Sci. Syst. (RSS)*, 2024.
[3] K. Pertsch *et al.*, "FAST: Efficient action tokenization for vision-language-action models," in *Proc. Robot.: Sci. Syst. (RSS)*, 2025.
[4] B. Zitkovich *et al.*, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 229, 2023, pp. 2165–2183.
[5] M. J. Kim *et al.*, "OpenVLA: An open-source vision-language-action model," in *Proc. Conf. Robot Learn. (CoRL)*, 2024.
[6] D. Lee, C. Kim, S. Kim, M. Cho, and W.-S. Han, "Autoregressive image generation using residual quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 11 513–11 522.
[7] F. Mentzer, D. Minnen, E. Agustsson, and M. Tschannen, "Finite scalar quantization: VQ-VAE made simple," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024, pp. 51 772–51 783.
[8] Y. Wang, H. Zhu, M. Liu, J. Yang, H.-S. Fang, and T. He, "VQ-VLA: Improving vision-language-action models via scaling vector-quantized action tokenizers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2025, pp. 11 089–11 099.
[9] Y. Liu *et al.*, "FASTer: Toward powerful and efficient autoregressive vision–language–action models with learnable action tokenizer and block-wise decoding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026, arXiv:2512.04952.
[10] C. Liu, X. Han, J. Gao, Y. Zhao, H. Chen, and Y. Du, "OAT: Ordered action tokenization," in *Proc. Robot.: Sci. Syst. (RSS)*, 2026.
[11] A. Jawaid and Y. Xiang, "NAC: Neural action codec for vision-language-action models," in *Proc. Conf. Robot Learn. (CoRL)*, 2026.
[12] Z. Dong *et al.*, "ActionCodec: What makes for good action tokenizers," *arXiv preprint arXiv:2602.15397*, 2026.
[13] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.
[14] K. Black *et al.*, "π_0: A vision-language-action flow model for general robot control," in *Proc. Robot.: Sci. Syst. (RSS)*, 2025.
[15] B. Liu *et al.*, "LIBERO: Benchmarking knowledge transfer for lifelong robot learning," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 36, 2023, pp. 44 776–44 791.
[16] K. Black *et al.*, "$\pi_{0.5}$: A vision-language-action model with open-world generalization," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 305, 2025, pp. 17–40.
[17] NVIDIA, "GR00T N1: An open foundation model for generalist humanoid robots," *arXiv preprint arXiv:2503.14734*, 2025.
[18] H. Fang *et al.*, "MolmoAct2: Action reasoning models for real-world deployment," arXiv preprint arXiv:2605.02881, 2026.
[19] A. Vaswani *et al.*, "Attention is all you need," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.
[20] D. Driess *et al.*, "Knowledge insulating vision-language-action models: Train fast, run fast, generalize better," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 38, 2025, pp. 102 867–102 888.
[21] M. Kang *et al.*, "X-Tokenizer: A multimodal action tokenizer for vision-language-action pretraining," arXiv preprint arXiv:2606.14752, 2026.
[22] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," in *Proc. Robot.: Sci. Syst. (RSS)*, 2023.
[23] C. Chi *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," *Int. J. Robot. Res.*, vol. 44, no. 10-11, pp. 1684–1704, 2025, arXiv:2303.04137.
[24] A. Prasad, K. Lin, J. Wu, L. Zhou, and J. Bohg, "Consistency policy: Accelerated visuomotor policies via consistency distillation," in *Proc. Robot.: Sci. Syst. (RSS)*, 2024.
[25] M. Shukor *et al.*, "SmolVLA: A vision-language-action model for affordable and efficient robotics," *arXiv preprint arXiv:2506.01844*, 2025.
[26] Y. Liu, J. I. Hamid, A. Xie, Y. Lee, M. Du, and C. Finn, "Bidirectional decoding: Improving action chunking via guided test-time sampling," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.
[27] K. Black, M. Galliker, and S. Levine, "Real-time execution of action chunking flow policies," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 38, 2025, pp. 33 383–33 407.
[28] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," *IEEE Trans. Comput.*, vol. C-23, no. 1, pp. 90–93, 1974.
[29] P. Gage, "A new algorithm for data compression," *C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994.
[30] N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi, "SoundStream: An end-to-end neural audio codec," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 30, pp. 495–507, 2022.
[31] A. Défossez, J. Copet, G. Synnaeve, and Y. Adi, "High fidelity neural audio compression," *Trans. Mach. Learn. Res.*, 2023, arXiv:2210.13438.

[32] M. Tagliasacchi, Y. Li, K. Misiunas, and D. Roblek, "SEANet: A multi-modal speech enhancement network," in *Proc. Interspeech*, 2020, pp. 1126–1130.

[33] H. Siuzdak, F. Grötschla, and L. A. Lanzendörfer, "SNAC: Multi-scale neural audio codec," in *Audio Imagination: NeurIPS 2024 Workshop on AI-Driven Speech, Music, and Sound Generation*, 2024.

[34] H. Siuzdak, "Vocos: Closing the gap between time-domain and fourier-based neural vocoders for high-quality audio synthesis," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2024.

[35] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022, pp. 11 976–11 986.

[36] T. Yu *et al.*, "Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Proc. Conf. Robot Learn. (CoRL)*, 2020, pp. 1094–1100.

[37] A. Mandlekar *et al.*, "What matters in learning from offline human demonstrations for robot manipulation," in *Proc. Conf. Robot Learn. (CoRL)*, 2022, pp. 1678–1690.

[38] S. Nasiriany *et al.*, "RoboCasa: Large-scale simulation of household tasks for generalist robots," in *Proc. Robot.: Sci. Syst. (RSS)*, 2024.

[39] Liquid AI, "LFM2 technical report," *arXiv preprint arXiv:2511.23404*, 2025.

[40] A. Khazatsky *et al.*, "DROID: A large-scale in-the-wild robot manipulation dataset," in *Proc. Robot.: Sci. Syst. (RSS)*, 2024.

[41] E. Jang *et al.*, "BC-Z: Zero-shot task generalization with robotic imitation learning," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 164, 2022, pp. 991–1002.

[42] A. Brohan *et al.*, "RT-1: Robotics transformer for real-world control at scale," in *Proc. Robot.: Sci. Syst. (RSS)*, 2023.

[43] H. R. Walke *et al.*, "BridgeData V2: A dataset for robot learning at scale," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 229, 2023, pp. 1723–1736.

[44] A. O'Neill *et al.*, "Open X-Embodiment: Robotic learning datasets and RT-X models," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024, pp. 6892–6903.

[45] D. Kalashnikov *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 87, 2018, pp. 651–673.

[46] C. Lynch *et al.*, "Interactive language: Talking to robots in real time," *IEEE Robot. Autom. Lett.*, pp. 1–8, 2023.

[47] H.-S. Fang *et al.*, "RH20T: A comprehensive robotic dataset for learning diverse skills in one-shot," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024, pp. 653–660.

[48] AgiBot-World-Contributors *et al.*, "AgiBot World Colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2025.

[49] K. Wu *et al.*, "RoboMIND: Benchmark on multi-embodiment intelligence normative data for robot manipulation," in *Proc. Robot.: Sci. Syst. (RSS)*, 2025.

[50] C. Hou *et al.*, "RoboMIND 2.0: A multimodal, bimanual mobile manipulation dataset for generalizable embodied intelligence," arXiv preprint arXiv:2512.24653, 2025.

[51] S. Wu *et al.*, "RoboCOIN: An open-sourced bimanual robotic data collection for integrated manipulation," arXiv preprint arXiv:2511.17441, 2025.

[52] NeoteAI Team and Fudan TEAI Team, "n_0-foundation: Towards the age of tactile intelligence," arXiv preprint arXiv:2608.29601, 2026, data: https://huggingface.co/datasets/NeoteAIEmbodied/OpenNeoData.

[53] A. Allshire *et al.*, "Scalable behavior cloning with open data, training, and evaluation," arXiv preprint arXiv:2606.27375, 2026.

[54] R. Takanami *et al.*, "AIRoA MoMa dataset: A large-scale hierarchical dataset for mobile manipulation," arXiv preprint arXiv:2509.25032, 2025.

[55] T. Jiang *et al.*, "Galaxea Open-World Dataset and G0 dual-system VLA model," arXiv preprint arXiv:2509.00576, 2025.

[56] WorldEngineAI, "WEB-Dataset," https://huggingface.co/datasets/WorldEngineAI/WEB-Dataset, 2026, accessed 2026-10.

[57] Hugging Face LeRobot team, "OpenArms bimanual folding dataset (LeRobot format)," https://huggingface.co/datasets/lerobot-data-collection/level12_rac_2_2026-02-07, 2026, accessed 2026-10.

[58] BitRobot Foundation, "HIW-500: Humanoids In-the-Wild dataset," https://huggingface.co/datasets/BitRobot/HIW-500, 2026, accessed 2026-10.

[59] Fourier, "Fourier ActionNet dataset," https://action-net.org/, 2025, accessed 2026-10.

[60] Z. Zhao *et al.*, "Humanoid everyday: A comprehensive robotic dataset for open-world humanoid manipulation," arXiv preprint arXiv:2510.08807, 2025.

[61] Unitree Robotics, "Unitree robotics open datasets (G1/Z1)," https://huggingface.co/unitreerobotics, 2026, accessed 2026-10.

[62] ETRI, "G1 inspire-hand humanoid dataset (ETRI v1)," https://huggingface.co/datasets/etri-robotai-platform-dev/g1_inspire_hand_2025_v3, 2026, accessed 2026-10; repository no longer public.

[63] R. Cadene *et al.*, "LeRobot: An open-source library for end-to-end robot learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2026.

[64] Open-H-Embodiment Consortium *et al.*, "Open-H-Embodiment: A large-scale dataset for enabling foundation models in medical robotics," arXiv preprint arXiv:2604.21017, 2026.

[65] N. Gürtler *et al.*, "Benchmarking offline reinforcement learning on real-robot hardware," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[66] T. Tao, M. K. Srirama, J. J. Liu, K. Shaw, and D. Pathak, "DexWild: Dexterous human interactions for in-the-wild robot policies," in *Proc. Robot.: Sci. Syst. (RSS)*, 2025.

[67] J. Lim *et al.*, "HRDexDB: A 4D dexterous grasping dataset across human and multiple robot embodiments," arXiv preprint arXiv:2604.14944, 2026.

[68] I. Guzey, B. Evans, S. Chintala, and L. Pinto, "Dexterity from touch: Self-supervised pre-training of tactile representations with robotic play," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 229, 2023, pp. 3142–3166.

[69] I. Guzey, Y. Dai, B. Evans, S. Chintala, and L. Pinto, "See to touch: Learning tactile dexterity through visual incentives," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2024, pp. 13 825–13 832.

[70] Y. Ze *et al.*, "TWIST2: Scalable, portable, and holistic humanoid data collection system," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2026, pp. 15 900–15 907.

[71] R. Iyengar, V. R. Osorio, P. Bhattachan, A. Ragobar, and B. Tripp, "A dataset of 40K naturalistic 6-degree-of-freedom robotic grasp demonstrations," arXiv preprint arXiv:1812.11683, 2018.

[72] R. Punamiya *et al.*, "EgoVerse: An egocentric human dataset for robot learning from around the world," in *Proc. Robot.: Sci. Syst. (RSS)*, 2026.

[73] Y. Zhong *et al.*, "EgoSteer: An open-source full-stack system towards steerable dexterous manipulation from egocentric videos," arXiv preprint arXiv:2607.09701, 2026.

[74] R.-Z. Qiu *et al.*, "Humanoid policy $\sim$ human policy," in *Proc. Conf. Robot Learn. (CoRL)*, ser. PMLR, vol. 305, 2025, pp. 2888–2906.

[75] Y. Tian *et al.*, "InternData-A1: Pioneering high-fidelity synthetic data for pre-training generalist policy," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2026, pp. 976–985.

[76] C. Li *et al.*, "BEHAVIOR-1K: A human-centered, embodied AI benchmark with 1,000 everyday activities and realistic simulation," arXiv preprint arXiv:2403.09227, 2024.

[77] H. Zhang, S. Christen, Z. Fan, O. Hilliges, and J. Song, "GraspXL: Generating grasping motions for diverse objects at scale," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2024.

[78] H. G. Singh *et al.*, "Hand-object interaction pretraining from videos," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2025, pp. 3352–3360.

[79] J. Bai *et al.*, "Openpi Comet: Competition solution for 2025 BEHAVIOR Challenge," arXiv preprint arXiv:2512.10071, 2025, data: https://huggingface.co/datasets/delinqu/comet-1.5k.

[80] C. Yin *et al.*, "Genie Sim 3.0: A high-fidelity comprehensive simulation platform for humanoid robot," arXiv preprint arXiv:2601.02078, 2026.

[81] AgiBot, "AgiBot World 2026," https://huggingface.co/datasets/agibot-world/AgiBotWorld2026, 2026, accessed 2026-10.

[82] X. Liu, J. Adalibieke, Q. Han, Y. Qin, and L. Yi, "DexTrack: Towards generalizable neural tracking control for dexterous manipulation from human references," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2025.

[83] leo01110111, "bc_with_hands: LEAP hand cube-lift demonstrations," https://github.com/leo01110111/bc_with_hands, 2026, accessed 2026-10.

[84] GenRobot, "10Kh RealOmni-Open dataset," https://huggingface.co/datasets/genrobot2025/10Kh-RealOmin-OpenData, 2025, accessed 2026-10.

[85] H. Zhang *et al.*, "Hy-Embodied-0.5-VLA: From vision-language-action models to a real-world robot learning stack," arXiv preprint arXiv:2606.14409, 2026.

[86] Simple AI *et al.*, "HiFi-UMI: Learning deployable manipulation policies from high-fidelity UMI data alone," arXiv preprint arXiv:2607.25895, 2026.

[87] K. Liu *et al.*, "FastUMI-100K: Advancing data-driven robotic manipulation with a large-scale UMI-style dataset," arXiv preprint arXiv:2510.08022, 2025.

[88] A. Jawaid and Y. Xiang, "OpenEgo: A large-scale multimodal egocentric dataset for dexterous manipulation," arXiv preprint arXiv:2509.05513, 2025.

[89] Q. Li *et al.*, "Scalable vision-language-action model pretraining for robotic manipulation with real-life human activity videos," arXiv preprint arXiv:2510.21571, 2025.
[90] LightwheelAI, "EgoDemo: A sample of EgoSuite-Open100K," https://huggingface.co/datasets/LightwheelAI/EgoDemo, 2026, accessed 2026-10.
[91] F. Sener *et al.*, "Assembly101: A large-scale multi-view video dataset for understanding procedural activities," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2022.
[92] S. Huang *et al.*, "OMG: Omni-modal motion generation for generalist humanoid control," arXiv preprint arXiv:2606.10340, 2026, data: https://huggingface.co/datasets/THU-MARS/OMG-Data.
[93] ChingMu, "ChingMu 1000-hour embodied motion dataset (MotionDecode)," https://chingmudata.github.io/MotionDecode/, 2026, accessed 2026-10.
[94] Eidon AI, "Tracker POV (IMU)," https://huggingface.co/datasets/eidon-ai/tracker-pov-imu, 2026, accessed 2026-10.
[95] A. Chowdhery, S. Narang, J. Devlin *et al.*, "PaLM: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 240, pp. 1–113, 2023.
[96] J. Lee *et al.*, "MolmoAct: Action reasoning models that can reason in space," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2026, pp. 2585–2592.