TokenPB: Enabling VLA Action Self-Verification via VLM Token Probability Bucketing

Chen Zhao[1,2*], **Zhuoran Wang**[1,2], **Haoyang Li**[1,2*], **Shifeng Bao**[1,2*], **Guanlin Li**[1,2*], **Youhe Feng**[1,2*], **Yang Li**[1,2], **Jie Tang**[3], **Jing Zhang**[1,2†],
[1] Renmin University of China, Beijing 100872, China
[2] Key Laboratory of Data Engineering and KnowledgeEngineering, MOE, China
[3] Tsinghua University
2021201645@ruc.edu.cn, zhang-jing@ruc.edu.cn
github.com/RUCKBReasoning/TokenPB

Abstract

Test-time sampling can improve Vision-Language-Action (VLA) policies, but only if the model can efficiently select a good action from multiple candidates. Prior work often relies on separately trained external verifiers, adding extra models and inference overhead. We propose **Token Probability Bucketing (TokenPB)**, a self-verification approach that discretizes continuous action chunks into bucket tokens and co-trains a single VLA checkpoint with a joint objective: a flow-matching loss for action generation and an auxiliary bucket loss $\mathcal{L}_{\text{bucket}}$ that teaches the VLM backbone to model observation-conditioned distributions over bucket-token sequences. At inference, TokenPB samples M candidates, scores each via teacher-forced evaluation under the learned distribution, and selects the best—enabling efficient batched scoring with observation-prefix KV-cache reuse. We provide theory connecting the training objective to a monotonic ranking property and bounds on expected selection regret that separate candidate-set effects from calibration. Across multiple backbones and benchmarks, TokenPB yields consistent success-rate improvements over single-sample inference from the same checkpoint (+2.7 in simulation, +17.1 in real-world, +16.4 under out-of-distribution shifts), and further improves over external-verifier baselines under the same sampling budget.

1 Introduction

Vision-Language-Action (VLA) policies must translate rich perceptual-language context into continuous, high-dimensional robot actions. In manipulation, the same observation can admit multiple plausible low-level action modes, while small deviations in a single executed action chunk can cause irreversible failures. This motivates *test-time inference with sampling-and-selection*: generate multiple candidate action chunks and execute the most promising one. In practice, however, sampling is only useful if the policy can efficiently and reliably score candidates without introducing additional models or large inference overhead.

A common approach is to train a separate external verifier to rank sampled actions. While effective, external verifiers add training complexity, require additional data or supervision choices, and often incur a forward pass per candidate at inference time. More fundamentally, action quality is defined in a continuous space, whereas the VLM backbone natively produces probabilities over discrete tokens. This mismatch makes it difficult to build a verifier that is both accurate and computationally efficient.

We propose Token Probability Bucketing (TokenPB), a self-verification framework that turns continuous-action verification into a token-probability estimation problem. TokenPB discretizes each

*Done as intern at Zhipu AI.
†Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

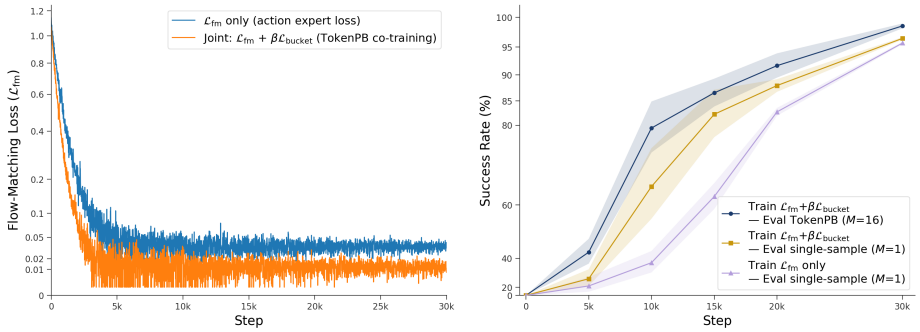

Figure 1: **Bucket co-training improves optimization and success. Left:** adding $\mathcal{L}_{\text{bucket}}$ alongside the flow-matching loss $\mathcal{L}_{\text{fm}}$ accelerates convergence (lower $\mathcal{L}_{\text{fm}}$ at the same step). **Right:** at every checkpoint, joint training ($\mathcal{L}_{\text{fm}} + \beta\,\mathcal{L}_{\text{bucket}}$) yields higher downstream success rates than the $\mathcal{L}_{\text{fm}}$-only baseline.

continuous action chunk with a fixed, deterministic tokenizer into a compact sequence of bucket tokens. To endow the VLA with self-verification ability, we co-train the VLM backbone with two complementary signals within a single checkpoint: a flow-matching loss $\mathcal{L}_{\text{fm}}$ drives a diffusion-style action expert to generate continuous action chunks from VLM context features, while an auxiliary bucket loss $\mathcal{L}_{\text{bucket}}$ teaches the same backbone to model observation-conditioned distributions over bucket-token sequences. At inference time, we sample M candidate action chunks by varying the injected noise, tokenize each candidate into bucket tokens, and score them via teacher-forced evaluation under the learned distribution. The highest-scoring candidate is selected for execution. Because all candidates share the same observation prefix, scoring can be implemented with batched decoding and observation-prefix KV-cache reuse, making test-time sampling practical.

A natural concern with joint training is that the auxiliary bucket objective might interfere with the primary action generation task. Figure 1 shows that the opposite is true: the bucket loss *accelerates* flow-matching convergence (left panel) and, at every training checkpoint, the jointly trained model achieves higher downstream success rates than a baseline trained only with the flow-matching action expert loss $\mathcal{L}_{\text{fm}}$ (right panel). This synergy arises because the two objectives share the VLM backbone, and $\mathcal{L}_{\text{bucket}}$ provides a complementary regularization signal that sharpens action-relevant representations beneficial to both generation and scoring.

We further provide theoretical support for the bucket score as a candidate ranker. We establish a *scoring monotonicity* property: as the learned distribution converges toward the true expert-induced distribution, the ranking produced by the bucket score becomes consistent with the true quality ordering among candidates. Building on this, we derive bounds on expected selection regret that separate two effects: (i) a candidate-set term capturing how the number and diversity of sampled candidates influence selection difficulty, and (ii) a calibration term measuring how well the learned distribution approximates the true one. This decomposition clarifies when test-time sampling is most beneficial and why improving calibration directly improves selection.

Our contributions are threefold: (1) we introduce TokenPB, which enables efficient test-time inference with sampling-and-selection inside a single VLA checkpoint by combining $\mathcal{L}_{\text{fm}}$ with an auxiliary $\mathcal{L}_{\text{bucket}}$ objective and performing batched, prefix-cached scoring; (2) we provide monotonicity guarantees and regret bounds that disentangle candidate-set effects from calibration; and (3) we demonstrate consistent improvements across multiple backbones and benchmarks, including significant gains from test-time inference over single-sample inference and further improvements over external-verifier baselines under the same sampling budget.

2 Related Work

External action verification. Prior work often improves VLA reliability by generating multiple candidate actions at test time and using an auxiliary scorer beyond the base generator for selection. RoboMonkey [17] trains a dedicated VLM-based verifier to score and select among candidates, while

RoVer [6] learns a process reward model that provides process-level quality estimates for ranking. V-GPS [23] instead uses an offline RL-learned value function as a plug-in critic to re-rank actions from a frozen generalist policy at deployment. AsyncVLA [12] is related but different: it trains an additional confidence rater to estimate token-level confidence and triggers selective self-correction via regeneration, rather than sampling-and-ranking action candidates. Overall, these approaches typically require training and deploying additional scoring modules (verifier, reward/value model, or confidence rater) alongside the base VLA. In contrast, TokenPB co-trains the verifier within the VLM backbone itself, enabling test-time selection without a separately deployed external scorer.

Verifier-free selection. MG-Select [11] scores candidates using an autoregressive VLA's own masking distribution. Its scoring mechanism is tied to action-token generation, whereas TokenPB separates continuous action generation by the action expert from teacher-forced token scoring by the VLM. This allows the two stages to scale independently and permits reuse of the sampler's observation-prefix KV cache.

Connection to offline RL and Q-functions. A Q-function estimates expected cumulative future reward, whereas TokenPB learns the observation-conditioned demonstration distribution $P^*(c \mid s)$ and uses it to rank a finite set of sampled candidates. Value-based approaches such as V-GPS require a learned critic; representative offline RL algorithms include CQL [16], TD3+BC [8], and IQL [15]. TokenPB instead uses the same demonstration pairs as behavior cloning, without reward annotations, preference labels, or Bellman backups. Co-training the sampler and verifier on shared data focuses verification on the sampler's candidates, although it does not guarantee that all candidates are in distribution. Its scores express conditional demonstration preference and need not identify the globally optimal action. Appendix I.2 compares these learning targets and evaluates the learned Q-functions under a shared-sampler protocol.

3 Preliminaries

Problem Statement. Let s_t denote the observation state at time step t, and let $Q(s_t)$ denote the task completion progress at state s_t. We define the single-step utility of an action a as:

$$R(a) = Q(s_{t+1} \mid a, s_t) - Q(s_t).$$

Since $Q(s_t)$ does not depend on a, maximizing $R(a)$ is equivalent to maximizing $Q(s_{t+1} \mid a, s_t)$; we use the latter for brevity. Suppose a VLA policy generates M candidate actions via test-time sampling, yielding the candidate set $\mathcal{A}_{s_t} = \{a^{(1)}, a^{(2)}, \ldots, a^{(M)}\}$. We denote the oracle-best candidate as:

$$a^* = \arg\max_{a^{(m)} \in \mathcal{A}_{s_t}} Q(s_{t+1} \mid a^{(m)}, s_t).$$

Note that a^* is constrained to the candidate set and may differ from the global optimum over the full action space. The objective of the VLM-based verifier is to select the candidate $\hat{a}^*$ that best approximates a^*. We define the *verifier effectiveness* as the utility difference between the verifier's selection and the oracle-best candidate:

$$\Delta_{\text{verify}}(s_t) = Q(s_{t+1} \mid \hat{a}^*, s_t) - Q(s_{t+1} \mid a^*, s_t),$$

where $\Delta_{\text{verify}} \leq 0$ by definition and $\Delta_{\text{verify}} = 0$ corresponds to perfect verification. For convenience, Section 4.1 bounds $|\Delta_{\text{verify}}(s_t)|$, which equals $Q(s_{t+1} \mid a^*, s_t) - Q(s_{t+1} \mid \hat{a}^*, s_t)$ under our definition.

Flow-Matching Action Generation. Modern VLA architectures adopt flow matching [19] to generate continuous actions from the VLM hidden representation. Given the VLM context feature f_c extracted from the current observation s_t, a lightweight diffusion action expert learns a conditional velocity field $v_\theta(x_\tau, \tau \mid f_c)$ that transports a standard Gaussian $x_0 \sim \mathcal{N}(0, I)$ toward the expert action distribution. Concretely, we sample an expert demonstration action chunk $a^{\text{demo}} \sim \mathcal{D}$ and consider the straight-line interpolation path $x_\tau = (1-\tau)x_0 + \tau a^{\text{demo}}$, with $\tau \in [0, 1]$, and train v_θ by matching the constant target velocity along this path:

$$\mathcal{L}_{\text{fm}} = \mathbb{E}_{\tau, x_0, a^{\text{demo}}}\left[\|v_\theta(x_\tau, \tau \mid f_c) - (a^{\text{demo}} - x_0)\|^2\right]. \tag{1}$$

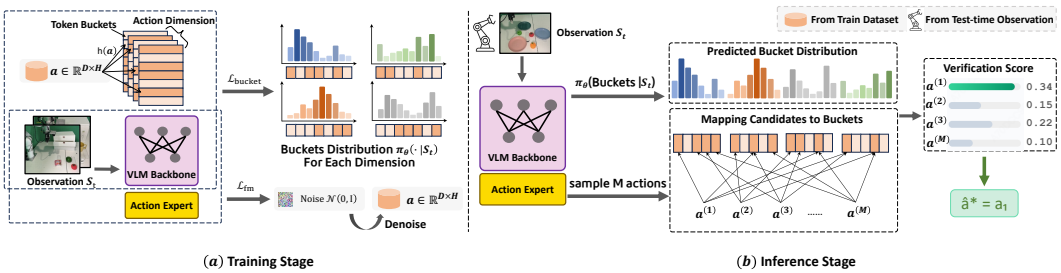

Figure 2: **Overview of the TokenPB framework.** Left: the VLM backbone learns a mapping from observation states to action token bucket distributions. Right: at test time, the VLA generates multiple candidate action chunks via the diffusion action expert, then scores each candidate by its alignment with the action token bucket probability distribution.

At inference time, an action chunk is produced by integrating v_θ from $\tau{=}0$ to $\tau{=}1$ using a few-step Euler solver. Because the action expert is decoupled from the heavy VLM backbone and shares only the hidden feature f_c, sampling multiple candidates incurs negligible overhead compared to a single VLM forward pass.

4 Method

TokenPB follows a *sample-and-verify* paradigm (Figure 2): the diffusion action expert samples multiple candidate action chunks, and the VLM backbone scores each candidate through an observation-conditioned distribution over discretized action buckets. Below we describe how actions are discretized (Section 4.1), how candidates are sampled, scored, and co-trained (Section 4.2), and how the resulting verifier's selection error is bounded (Section 4.3).

4.1 Action Token Bucketing

We discretize a continuous action chunk into a sequence of discrete tokens via a fixed, deterministic tokenizer h as the mapping function. For an action chunk $a \in \mathbb{R}^{D\times H}$ (action dimension D, chunk horizon H), the tokenizer outputs a token sequence

$$c = h(a) \in \{1, \dots, C\}^L,$$

where C is the token vocabulary size and L is the resulting sequence length[1]. The mapping h is non-learned and admits a deterministic decoder g that reconstructs a continuous proxy action $\tilde{a} = g(c)$. Due to quantization in the tokenization process, multiple continuous actions can map to the same token sequence; thus reconstruction is only defined up to the tokenizer's quantization resolution.

Discretization and selection. Tokenization introduces a quantization gap between a continuous candidate and its decoded proxy action. Let $a^* = \arg\max_{a^{(m)}\in\mathcal{A}_{s_t}} Q(s_{t+1} \mid a^{(m)}, s_t)$ be the oracle-best candidate in the sampled set, and let $\tilde{a}^* = g(h(a^*))$ be the decoded proxy action of its token sequence. The total verification gap decomposes as:

$$|\Delta_{\text{verify}}| \le \underbrace{\left|Q(s_{t+1} \mid \tilde{a}^*, s_t) - Q(s_{t+1} \mid \hat{a}^*, s_t)\right|}_{\text{selection error}} + \underbrace{\left|Q(s_{t+1} \mid a^*, s_t) - Q(s_{t+1} \mid \tilde{a}^*, s_t)\right|}_{\text{discretization error}}.$$

The discretization error depends solely on the bucket granularity and is independent of the verifier. We observe diminishing returns beyond 1024 buckets, suggesting discretization is no longer the dominant bottleneck (Figure 5(right) in Section 5.4). We therefore focus on bounding the selection error in Section 4.3.

[1] In our implementation, h is instantiated by Textual FAST (Appendix G), which transforms action chunks into compact discrete tokens and then encodes them using the VLM tokenizer; prediction-head reuse and score normalization are detailed in Appendix G.4.

4.2 Candidate Sampling and Verification

Sample. As depicted in Figure 2, conditioned on the VLM context feature $f_c = f_\theta(s_t)$, the diffusion action expert generates M candidate action chunks by varying the injected noise, yielding $\mathcal{A}_{s_t} = \{a^{(1)}, \ldots, a^{(M)}\}$. Because the diffusion action expert is lightweight, the sampling overhead is negligible relative to a VLM forward pass.

Verify. The VLM backbone defines a conditional distribution over *action-token sequences* given the current observation, denoted $\pi_\theta(\cdot \mid s_t)$. For a candidate $a^{(m)}$, we map it into $c^{(m)} = h(a^{(m)}) = (c_1^{(m)}, \ldots, c_L^{(m)})$ and assign a score under π_θ:

$$\text{score}(a^{(m)}) \;=\; \frac{1}{L} \log \pi_\theta\big(c^{(m)} \mid s_t\big) \;=\; \frac{1}{L} \sum_{\ell=1}^{L} \log \pi_\theta\big(c_\ell^{(m)} \mid s_t, c_{<\ell}^{(m)}\big).$$

We then select

$$\hat{a}^* = \arg\max_{a^{(m)} \in \mathcal{A}_{s_t}} \text{score}(a^{(m)}).$$

This verification stage requires no external model: $\pi_\theta(\cdot \mid s_t)$ is computed by the same VLM backbone used for action generation. Moreover, because all candidates share the same observation prefix, we encode the prefix once and reuse its transformer KV cache. For each candidate $a^{(m)}$, we score only the candidate-specific token continuation in a single teacher-forced pass conditioned on the cached prefix, batching across all m. This yields a small additional overhead in practice (Section 5.4), without requiring a separate verifier or an expensive per-candidate recomputation of the shared prefix.

Co-training objective. To learn the bucket-scoring distribution $\pi_\theta(\cdot \mid s)$, we jointly optimize the flow-matching action generation loss $\mathcal{L}_{\text{fm}}$ and an auxiliary *token-bucket cross-entropy* on demonstration bucket-token sequences:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{fm}} + \beta \cdot \mathcal{L}_{\text{bucket}}, \qquad \mathcal{L}_{\text{bucket}} = -\mathbb{E}_{(s_t, a^{\text{demo}}) \sim \mathcal{D}} \Big[\frac{1}{L^{\text{demo}}} \log \pi_\theta\big(h(a^{\text{demo}}) \mid s_t\big) \Big],$$

where L^{demo} is the token sequence length produced by h on the demonstration action chunk. As training converges, $\pi_\theta(\cdot \mid s)$ approaches the demonstration-induced conditional distribution over token sequences.

4.3 Selection Error Bound

We now bound the verifier's expected sub-optimality. The central question is: how much regret does the verifier incur by ranking candidates via the bucket score rather than true action quality?

The connection between the two rests on a natural assumption: candidates whose bucket representations are more frequent under expert demonstrations tend to have higher task utility. The analysis is conditional on this alignment between demonstration likelihood and task utility.

Since $\mathcal{L}_{\text{bucket}}$ is strictly proper, $\pi_\theta(\cdot \mid s_t) \to P^*(\cdot \mid s_t)$ as training proceeds, where $P^*(\cdot \mid s_t)$ denotes the (unknown) ground-truth distribution of token sequences induced by the expert demonstrations in $\mathcal{D}$ conditioned on state s_t. For our verifier, only the token sequences that appear among the sampled candidates matter. We therefore quantify the residual mismatch via a candidate-set calibration error in log space:

$$\varepsilon_{\text{cal}} := \mathbb{E}_{s_t} \Big[\max_{m \in \{1, \ldots, M\}} \big| \log \pi_\theta(h(a^{(m)}) \mid s_t) - \log P^*(h(a^{(m)}) \mid s_t) \big| \Big]. \tag{2}$$

Here $P^*(h(a^{(m)}) \mid s_t)$ and $\pi_\theta(h(a^{(m)}) \mid s_t)$ denote probabilities of the entire token sequence produced by h. Once ε_{cal} is sufficiently small, π_θ preserves the true quality ordering among candidates—a property we call *scoring monotonicity*:

$$P^*(h(a^{(m)}) \mid s_t) > P^*(h(a^{(m')}) \mid s_t) \implies \pi_\theta(h(a^{(m)}) \mid s_t) > \pi_\theta(h(a^{(m')}) \mid s_t). \tag{3}$$

The formal statements, assumptions, and proofs are given in Appendices B and C.

Table 1: **Comparison with Single-Sample vs. Test-Time Inference.** All models sharing the same backbone are co-trained with both bucket-token and flow-matching objectives using a single checkpoint; configurations differ only at inference time. "Model-GR00T": single sample (M=1) without verification. "Model-GR00T+TokenPB": multi-candidate verification (M=16). For Real-World, "Seen" denotes in-distribution tasks and "Unseen" denotes held-out instructions and object configurations not seen during training. Detailed training settings are provided in Table 8.

Benchmark	**Model**	**Success Rate (%)**					
LIBERO		**Spatial**	**Object**	**Goal**	**Long**	**Avg.**	
	Qwen2.5-VL-GR00T	94.6	97.2	93.8	90.2	93.9	
	Qwen2.5-VL-GR00T+TokenPB	96.6	98.8	97.0	94.6	96.8	
	InternVL3.5-GR00T	95.8	96.0	94.2	92.8	94.7	
	InternVL3.5-GR00T+TokenPB	97.6	**99.4**	97.2	96.8	97.8	
	$\pi_{0.5}$	97.8	98.4	96.6	93.0	96.5	
	$\pi_{0.5}$**+TokenPB**	**98.4**	**99.4**	**98.2**	**97.6**	**98.4**	
CALVIN ABC→D		**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Avg. Len.**
	Qwen2.5-VL-GR00T	90.8	81.2	74.9	68.7	59.5	3.751
	Qwen2.5-VL-GR00T+TokenPB	93.1	87.4	81.6	75.9	71.3	4.093
	$\pi_{0.5}$	93.5	86.7	79.6	75.3	71.1	4.071
	$\pi_{0.5}$**+TokenPB**	**95.4**	**90.6**	**85.3**	**79.6**	**75.8**	**4.267**
RoboTwin 2.0		**Block Rank RGB**	**Open Laptop**	**Adjust Bottle**	**.....**	**Avg.**	
		Easy / Hard	*Easy / Hard*	*Easy / Hard*		*Easy / Hard*	
	Qwen2.5-VL-GR00T	38.0 / 11.0	39.0 / 18.0	96.0 / 16.0	...	30.1 / 6.7	
	Qwen2.5-VL-GR00T+TokenPB	**52.0 / 24.0**	**55.0 / 27.0**	**97.0 / 26.0**	...	**39.3 / 10.2**	
Real-World		**Push Blocks**	**Clean Table**	**Pick & Place**	**Hang Cups**	**Avg.**	
		Seen / Unseen	*Seen / Unseen*	*Seen / Unseen*	*Seen / Unseen*	*Seen / Unseen*	
	Qwen2.5-VL-GR00T	56.2 / 5.0	52.5 / 8.0	45.5 / 28.0	43.7 / 32.0	49.5 / 18.3	
	Qwen2.5-VL-GR00T+TokenPB	75.4 / **45.0**	**72.5** / 24.0	75.0 / 36.0	62.5 / 40.0	71.4 / 36.3	
	InternVL3.5-GR00T	54.1 / 10.0	56.7 / 8.0	55.0 / 16.0	68.7 / 40.0	58.6 / 18.5	
	InternVL3.5-GR00T+TokenPB	72.7 / 40.0	66.7 / 16.0	**83.3** / 32.0	**81.3 / 56.0**	76.1 / 36.0	
	$\pi_{0.5}$	74.0 / 25.0	61.5 / 16.0	71.3 / **40.0**	55.3 / 40.0	65.6 / 30.3	
	$\pi_{0.5}$**+TokenPB**	**81.7** / 40.0	70.7 / **48.0**	79.5 / **40.0**	78.3 / 48.0	**77.6 / 44.0**	

Expected selection error bound. For analysis, we model the deterministic $\arg\max$ over score$(a^{(m)})$ as a Gibbs selector with effective sharpness λ—a standard surrogate that recovers the argmax in the limit $\lambda \to \infty$. The expected selection regret then satisfies:

$$\mathbb{E}\big[Q(s_{t+1} \mid a^*, s_t) - Q(s_{t+1} \mid \hat{a}^*, s_t)\big] \;\le\; \underbrace{\frac{\log M}{\lambda}}_{\text{candidate diversity}} \;+\; \underbrace{O(\varepsilon_{\text{cal}})}_{\text{calibration gap}} . \tag{4}$$

The first term reflects the intrinsic cost of choosing among M candidates with a soft scorer; the second vanishes as calibration improves. The full derivation is provided in Appendix D.2.

5 Experiments

We evaluate TokenPB across simulated and real-world manipulation benchmarks, addressing four research questions that map one-to-one to the subsections below:

Q1: Does TokenPB consistently improve over single-sample inference across benchmarks, backbones, and distribution shifts? (Section 5.1)

Q2: How does TokenPB compare to external verification methods in both task performance and computational cost? (Section 5.2)

Q3: How does the verification overhead scale with the candidate set size M? (Section 5.3)

Q4: Where do the gains come from—co-training, test-time verification, or both—and can the bucket score reliably distinguish good actions from bad ones? (Section 5.4)

Setup. We evaluate two VLM backbones (Qwen2.5-VL-3B, InternVL3.5-2B) paired with a GR00T-N1-style flow-matching action expert [1], and one VLA ($\pi_{0.5}$) following the same co-training protocol. All models are co-trained in a single checkpoint with a joint objective combining (i) the flow-matching loss $\mathcal{L}_{\text{fm}}$ for continuous action generation and (ii) an auxiliary bucket-token objective $\mathcal{L}_{\text{bucket}}$ for

candidate scoring. At inference time, MODEL-GR00T denotes the resulting backbone+expert policy run with single-sample inference (M=1) and no selection, while MODEL-GR00T+TOKENPB enables test-time sampling-and-selection by scoring M candidates (default M=16) using the learned bucket score. Simulation benchmarks include LIBERO [20], CALVIN [22], and RoboTwin2.0 [4]; real-world evaluation covers four bimanual tasks (Appendix E.4, F).

5.1 Comparison with Single-Sample vs. Test-Time Inference

Table 1 tests the central premise of TokenPB: from the *same co-trained checkpoint*, switching from single-sample inference (M=1) to test-time sampling-and-selection (with TokenPB scoring, default M=16) should consistently improve task success without any additional training. This is confirmed across every benchmark–backbone combination. On LIBERO the average gain is +2.7 points (up to +3.1 on InternVL3.5); on CALVIN ABC→D, +0.34 in average length on Qwen2.5-VL and +0.20 on $\pi_{0.5}$; on RoboTwin2.0, +9.2 on Easy and +3.5 on Hard; and in real-world environments, +21.9 / +17.5 / +12.0 on the three backbones (average +17.1). The gains also generalize beyond the training distribution: on a held-out Real-World Unseen setting with novel instructions and object configurations, TokenPB yields an average +16.4-point gain—comparable to the in-distribution improvement—indicating that the verifier captures task-relevant action semantics rather than memorizing training-time patterns (statistical details in Appendix E.5).

5.2 Comparison with Baseline Verification Methods

A central claim of TokenPB is that it enables effective test-time sampling-and-selection *without* relying on a separately trained external verifier, while achieving competitive or superior task performance. We validate this claim along two axes: *task performance* and *computational cost*, comparing against both external-verifier-based selection (e.g., RoboMonkey) and internal scoring baselines that still incur per-candidate scoring compute (e.g., MG-Select).

Task performance. We compare TokenPB against RoboMonkey [17], MG-Select [11], AsyncVLA [12] and RoVer [6] on their respective benchmarks. TokenPB outperforms MG-Select on LIBERO (96.8% vs. 93.1%) and achieves the highest success rate on SimplerEnv-WidowX (0.76), as shown in Table 2. It also yields a larger gain than RoVer on CALVIN (+0.20 vs. +0.11 average length, Table 3)—notably, TokenPB achieves these results without requiring any separately trained verifier.

Computational cost. Figure 3 compares the extra inference time as M varies from 1 to 64. RoboMonkey and MG-Select grow roughly linearly with M (>4 s and >1.5 s at M=64), because their scoring introduces *per-candidate* compute: RoboMonkey runs a separately trained verifier on each candidate action, while MG-Select performs an additional *masked* scoring forward pass for each candidate under the VLA's own masking distribution. In both cases, batching can improve throughput but does not eliminate the $O(M)$ *total* scoring compute, since each candidate still requires its own scoring computation (and MG-Select cannot score all candidates in one teacher-forced continuation pass conditioned on a shared observation-prefix KV cache). In contrast, TokenPB stays below 0.2 s even at M=64—roughly one order of magnitude faster than RoboMonkey and MG-Select. The key reason is that TokenPB's sampling and verification share the same observation prefix within a single VLM backbone: the observation-prefix KV cache can be computed once and reused to score all M candidates via a single batched teacher-forced pass over candidate-specific token continuations, yielding near-constant overhead with respect to M and moderate memory growth due to KV-cache reuse (Figure 4(left) in Section 5.3).

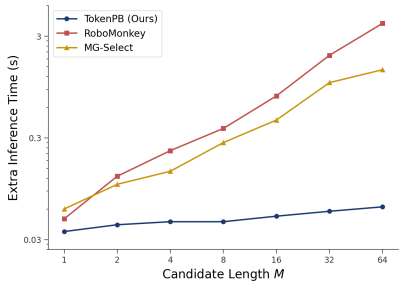

Figure 3: **Inference time comparison with RoboMonkey and MG-Select.**

The timing comparison enables the respective prefix-sharing and batching optimizations; additional protocol details and Qwen2.5-VL-3B memory measurements are provided in Appendix I.3.

Table 2: **Comparison with baseline verification methods on LIBERO and SimplerEnv-WidowX.** Original baseline rows retain results from their official publications; only LIBERO-Long is available for the original RoboMonkey row, and V-GPS is goal-conditioned. Added rows report MG-Select single-sample inference and a reproduced RoboMonkey-Qwen2.5 verifier with the shared $\pi_{0.5}$ single-sample baseline. Reproduction details are in Appendix I.1.

	LIBERO (%)					SimplerEnv-WidowX (SR)				
Method	**Spatial**	**Object**	**Goal**	**Long**	**Avg.**	**Spoon on Towel**	**Carrot on Plate**	**Stack Cubes**	**Eggplant in Basket**	**Avg.**
V-GPS	–	–	–	–	–	0.46	0.16	0.07	0.84	0.38
RoboMonkey	–	–	–	56.5	–	0.68	0.72	0.44	0.18	0.51
MG-Select (single)	97.4	95.4	95.6	79.6	92.0	0.67	0.71	0.42	0.08	0.47
MG-Select	97.2	98.0	94.5	82.7	93.1	0.69	**0.75**	0.43	0.14	0.50
AsyncVLA	98.0	98.2	96.4	96.0	97.4	0.71	0.67	**0.58**	0.88	0.71
$\pi_{0.5}$ (single, shared)	97.8	98.4	96.6	93.0	96.5	0.74	0.65	0.38	0.87	0.66
RoboMonkey-Qwen2.5	**98.8**	98.2	97.0	95.0	97.3	0.76	0.65	0.49	0.95	0.71
TokenPB (Ours)	**98.4**	**99.4**	**98.2**	**97.6**	**98.4**	**0.86**	0.71	0.47	**1.00**	**0.76**

Table 3: **Comparison with baseline verification methods on CALVIN ABC→D.** All baseline results are from their respective official publications.

Method	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Avg. Len.**
MoDE	96.2	88.9	81.1	71.8	63.5	4.01
MoDE + RoVer	97.1	90.9	82.5	74.9	66.6	4.12 (+0.11)
$\pi_{0.5}$	93.5	86.7	79.6	75.3	71.1	4.07
$\pi_{0.5}$ + TokenPB (Ours)	**95.4**	**90.6**	**85.3**	**79.6**	**75.8**	**4.27 (+0.20)**

5.3 Test-Time Scaling Behavior

We study how TokenPB's performance and overhead scale with M on LIBERO-Long (Figure 4). Because all candidates share the same observation prefix and scoring is batched in a single forward pass, the extra inference time remains nearly constant regardless of M. A naïve implementation would duplicate the full KV cache for each candidate, causing raw memory to exceed 60 GB at $M{=}64$. By reusing the observation-prefix KV cache—which is identical across all candidates—we reduce memory to $\sim$32 GB at $M{=}64$ and $\sim$20 GB at $M{=}16$, comfortably fitting on a single consumer-grade GPU. We adopt $M{=}16$ as the default.

Performance improves steadily with M: success rate climbs from $\sim$90% ($M{=}1$) to over 95% ($M{=}32$), while action RMSE decreases monotonically. In contrast, random choice and majority voting plateau or fluctuate and fail to benefit from larger candidate pools. TokenPB thus converts additional test-time candidates into tangible gains at nearly fixed inference cost.

5.4 Ablation and Analysis

Disentangling co-training and verification. We isolate the two components by removing each in turn (Table 4). Without the bucket objective, randomly selecting among candidates provides no gain over a single sample—candidate diversity alone is useless without a scoring function. Adding the bucket objective during training already improves success rate by +2.0 on LIBERO and +4–6 on RoboTwin2.0. Applying verification on top yields a further +2.9 on LIBERO and up to +16 on RoboTwin2.0 Hard. The total gain exceeds the sum of individual components, confirming that co-training provides the scoring function and verification exploits it at test time.

Score discriminability. We validate the bucket score from two angles (Figure 5). First, injecting random noise into all but one clean candidate on LIBERO (left panel), the verifier's clean-candidate selection rate rises to nearly 100% at 10% noise magnitude while random choice degrades sharply. Second, successful trajectories consistently exhibit significantly higher scores than failed ones across all LIBERO suites (middle panel); moreover, the relative ordering of average scores across suites mirrors the ordering of task success rates, confirming that the score reliably reflects task difficulty. TokenPB-selected actions on RoboTwin2.0 Hard further improve success rate by +12.0 points with fewer collisions and more recovery attempts (Table 5). Qualitative examples are in Appendix A;

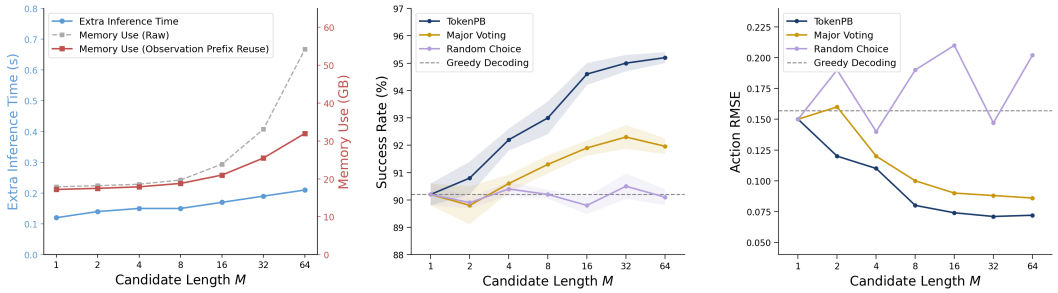

Figure 4: **Test-time cost–performance trade-off on LIBERO-Long as M varies from 1 to 64.** *Left:* Extra inference time (blue, left axis) remains nearly constant across all M. The gray dashed line shows raw GPU memory without optimization; by reusing the observation-prefix KV cache (red), memory grows only to $\sim$32 GB even at M=64—roughly half the raw cost. *Middle:* Success rate versus M. TokenPB (blue) improves steadily, whereas random choice (purple) and majority voting (yellow) plateau. *Right:* Action RMSE versus M. TokenPB drives a monotonic decrease, confirming that selected actions become progressively closer to the ground truth. We adopt M=16 as the default.

Table 4: **Disentangling co-training representation benefit from test-time verification benefit.** All variants use Qwen2.5-VL-3B as the VLM backbone. "Co-trained + Single (M=1)" corresponds to the GR00T baseline in Table 1. "No Bucket" removes the auxiliary token-bucket cross-entropy $\mathcal{L}_{\text{bucket}}$ during training (the action expert is still trained with $\mathcal{L}_{\text{fm}}$); consequently, no learned bucket-likelihood scorer is available at test time and multi-sample selection reduces to random choice. "Full TokenPB" is the complete framework with co-training and sampling-and-selection (M=16).

Configuration	LIBERO (%)					RoboTwin2.0 (%)					
	Spatial	Object	Goal	Long	Avg.	Blocks Ranking		Open Laptop		Shake Bottle	
						Easy	Hard	Easy	Hard	Easy	Hard
No Bucket + Single (M=1)	92.4	95.6	91.8	87.6	91.9	32	7	33	13	79	16
No Bucket + Random Select (M=16)	92.0	95.2	91.4	87.8	91.6	30	6	31	12	77	15
Co-trained + Single (M=1)	93.6	96.2	92.8	90.2	93.2	38	11	39	18	81	21
Full TokenPB (M=16)	**96.6**	**98.8**	**97.0**	**94.6**	**96.8**	**52**	**24**	**55**	**27**	**86**	**37**

additional backbone, LIBERO-Plus, RoboCasa, and embodiment-pretraining evaluations are in Appendix I.4.

Bucket granularity. We vary the vocabulary size from 64 to 4096 to examine how discretization resolution affects verification performance (Figure 5, right). Success rate improves monotonically and saturates beyond 1024 tokens at approximately 97%. With a small vocabulary, coarse discretization can map distinct continuous actions to the same token sequence, limiting the verifier's ability to distinguish candidates with different execution quality. Increasing the vocabulary provides finer action representations and allows the bucket score to capture distinctions that would otherwise be lost during tokenization. However, further increasing the vocabulary beyond 1024 yields only marginal gains, suggesting that finer discretization alone is insufficient to substantially improve selection. This trend is consistent with discretization becoming a less restrictive bottleneck at sufficient granularity, with further improvements depending more on the quality of the sampled candidates and the verifier's ability to rank them accurately.

6 Limitations and Discussion

Memory scaling with candidate set size. Although observation-prefix KV-cache reuse substantially reduces per-candidate cost, GPU memory still grows linearly with the number of candidates M, which may limit deployment on memory-constrained edge devices. See Appendix I.3 for details.

Table 5: **Fine-grained trajectory metrics on RoboTwin2.0 (Hard).** TokenPB-selected actions are compared against single-sample ones from the same co-trained checkpoint. *Recovery Attempts*: higher values indicate more retry attempts when the initial action fails, reflecting stronger error-correction behavior. *Env. Collision Before Grasp*: lower values indicate fewer irrelevant collisions with the environment before grasping the target object.

Metric	Adjust Bottle		Blocks Ranking RGB		Open Laptop		Shake Bottle	
	TokenPB	Baseline	TokenPB	Baseline	TokenPB	Baseline	TokenPB	Baseline
Avg. Success Steps (↓)	**107.3** (-45.2)	152.5	**754.0** (-282.9)	1036.9	**142.3** (-17.1)	159.4	**95.3** (-54.4)	149.7
Recovery Attempts (↑)	**1.1** (+0.7)	0.4	**3.3** (+2.6)	0.7	**2.9** (+2.6)	0.3	**1.3** (+0.6)	0.7
Env. Collision Before Grasp (%, ↓)	**3** (-43)	46	**14** (-15)	29	**0** (-14)	14	**4** (-34)	38
Success Rate (%, ↑)	**26** (+10)	16	**24** (+13)	11	**27** (+9)	18	**37** (+16)	21

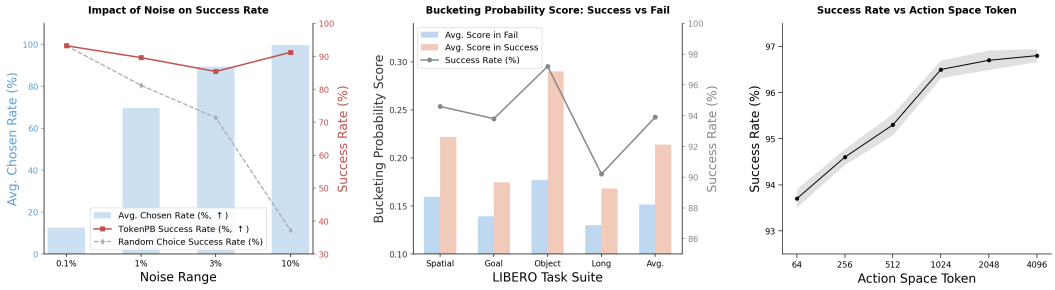

Figure 5: **Verification effectiveness analysis.** Left: TokenPB maintains near-constant success rate under noise injection compared to random choice. Middle: Bucketing probability scores of successful trajectories are significantly higher than failed ones, and the score magnitude tracks task success rate across suites. Right: Task success rate saturates beyond 1024 buckets.

Frequency-based scoring and multi-modal action distributions. When multiple valid actions exist for the same observation, the verifier favors the most *frequent* demonstration action rather than the globally optimal one, since the scoring function is trained to approximate the conditional demonstration distribution $P^*(\cdot \mid s)$. However, this concern is mitigated by the observation-conditioned nature of the bucket distribution: in states where atypical actions (e.g., obstacle avoidance detours) are appropriate, these actions are well-represented in the *conditional* distribution $P^*(c \mid s_t)$ for the relevant observations, so the verifier does not penalize them. Empirically, the Real-World Unseen gains (+16.4 points, Table 1) are comparable to in-distribution gains (+17.1), and the increased recovery attempts on RoboTwin2.0 Hard (Table 5) confirm that the verifier successfully identifies non-typical corrective actions rather than blindly favoring high-frequency patterns.

Dependence on demonstration data coverage. For action types entirely absent from the training demonstrations, TokenPB cannot produce meaningful scores, but this is a fundamental data coverage limitation shared by all behavior-cloning-based VLA methods. Future work could incorporate broader demonstration data and online interaction feedback to improve scoring reliability for actions beyond the original demonstration coverage.

7 Conclusion

We propose Token Probability Bucketing (TokenPB), a self-verification framework that unifies action generation and verification within a single VLA checkpoint. By co-training the VLM backbone to predict observation-conditioned bucket distributions, TokenPB enables test-time candidate selection without any external verifier, achieving consistent gains across multiple architectures and benchmarks with near-constant overhead. However, GPU memory grows with the number of candidates, and demonstration likelihood may not fully reflect task utility, motivating future work on more memory-efficient verification and scoring that better captures task outcomes.

Acknowledgements

This work is supported by the National Key Research & Development Plan (2023YFF0725100) and the National Natural Science Foundation of China (92570121, 62322214, U23A20299, U24B20144). This work was completed during the author's internship at Z.ai.

References

[1] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π0: A vision-language-action flow model for general robot control. corr, abs/2410.24164, 2024. doi: 10.48550. *arXiv preprint ARXIV.2410.24164*, 2025.

[3] Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025.

[4] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[5] 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.

[6] Mingtong Dai, Lingbo Liu, Yongjie Bai, Yang Liu, Zhouxia Wang, Rui Su, Chunjie Chen, Liang Lin, and Xinyu Wu. Rover: Robot reward model as test-time verifier for vision-language-action model. *arXiv preprint arXiv:2510.10975*, 2025.

[7] Danny Driess, Jost Tobias Springenberg, Brian Ichter, Lili Yu, Adrian Li-Bell, Karl Pertsch, Allen Z Ren, Homer Walke, Quan Vuong, Lucy Xiaoyang Shi, et al. Knowledge insulating vision-language-action models: Train fast, run fast, generalize better. *arXiv preprint arXiv:2505.23705*, 2025.

[8] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021. URL `https://arxiv.org/abs/2106.06860`.

[9] Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

[10] Ankit Goyal, Hugo Hadfield, Xuning Yang, Valts Blukis, and Fabio Ramos. Vla-0: Building state-of-the-art vlas with zero modification. *arXiv preprint arXiv:2510.13054*, 2025.

[11] Suhyeok Jang et al. Verifier-free test-time sampling for vision-language-action models. *arXiv preprint arXiv:2510.05681*, 2025.

[12] Yuhua Jiang et al. Asyncvla: Asynchronous flow matching for vision-language-action models. *arXiv preprint arXiv:2511.14148*, 2025.

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

[14] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning*, pages 2679–2713. PMLR, 2025.

[15] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit Q-learning. *arXiv preprint arXiv:2110.06169*, 2021. URL https://arxiv.org/abs/2110.06169.

[16] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020. URL https://arxiv.org/abs/2006.04779.

[17] Jacky Kwok, Christopher Agia, Rohan Sinha, Matt Foutter, Shulu Li, Ion Stoica, Azalia Mirhoseini, and Marco Pavone. Robomonkey: Scaling test-time sampling and verification for vision-language-action models. *arXiv preprint arXiv:2506.17811*, 2025.

[18] Jason Lee, Jiafei Duan, Haoquan Fang, Yuquan Deng, Shuo Liu, Boyang Li, Bohan Fang, Jieyu Zhang, Yi Ru Wang, Sangho Lee, et al. Molmoact: Action reasoning models that can reason in space. *arXiv preprint arXiv:2508.07917*, 2025.

[19] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[20] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

[21] Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. Hybridvla: Collaborative diffusion and autoregression in a unified vision-language-action model. *arXiv preprint arXiv:2503.10631*, 2025.

[22] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022.

[23] Mitsuhiko Nakamoto, Oier Mees, Aviral Kumar, and Sergey Levine. Steering your generalists: Improving robotic foundation models via value guidance. *arXiv preprint arXiv:2410.13816*, 2024.

[24] 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.

[25] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[26] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[27] Zhide Zhong, Haodong Yan, Junfeng Li, Xiangchen Liu, Xin Gong, Tianran Zhang, Wenxuan Song, Jiayi Chen, Xinhu Zheng, Hesheng Wang, et al. Flowvla: Visual chain of thought-based motion reasoning for vision-language-action models. *arXiv preprint arXiv:2508.18269*, 2025.

A Qualitative Verification Examples

To provide an intuitive understanding of the verification mechanism, Figure 6 visualizes representative examples of TokenPB scoring across diverse tasks from RoboTwin, LIBERO, and real-world environments. In each case, the verifier assigns the highest bucketing probability score to the candidate whose action semantics best match the task instruction, while candidates with incorrect spatial targets or object selections receive substantially lower scores. These examples illustrate that the co-trained VLM backbone captures task-relevant action semantics through the bucket probability distribution, enabling reliable best-candidate selection even in visually complex multi-object scenes.

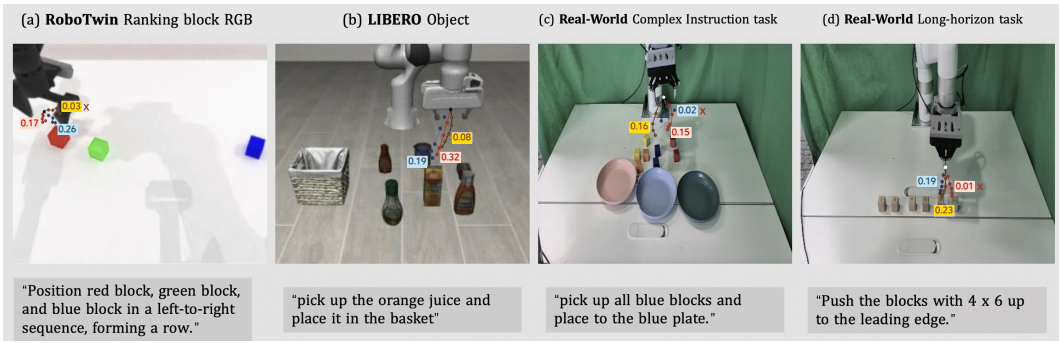

Figure 6: **Qualitative visualization of TokenPB verification across diverse tasks.** Each panel shows a task scene with candidate actions and their bucketing probability scores. Candidates with higher scores (shown in blue boxes with check marks) are selected for execution, while lower-scored candidates (red boxes with crosses) are rejected. TokenPB correctly assigns the highest probability to semantically appropriate actions: (a) placing blocks in the correct spatial order in RoboTwin, (b) grasping the target object in LIBERO, (c) selecting the correct objects in a complex real-world instruction task, and (d) pushing the correct blocks in a long-horizon task.

B Scoring Monotonicity under Cross-Entropy Training

We establish a stronger result: the cross-entropy training dynamics guarantee that the predicted probabilities eventually become monotone in the true probabilities, even before exact calibration is achieved.

B.1 Finite-Sample Convergence

Assumption B.1 (Realizability)**.** The model class $\{\pi_\theta : \theta \in \Theta\}$ contains the true conditional distribution, i.e., there exists $\theta^* \in \Theta$ such that $\pi_{\theta^*}(\cdot \mid s) = P^*(\cdot \mid s)$ for all s in the support of the training distribution.

Assumption B.2 (Regularity)**.** The loss landscape $\mathcal{L}_{\text{CE}}(\theta)$ satisfies: (i) $\nabla^2 \mathcal{L}_{\text{CE}}(\theta^*)$ is positive definite (non-degenerate minimum); (ii) $\pi_\theta(c \mid s)$ is continuous in θ for all c, s.

Theorem B.3 (Scoring monotonicity)**.** *Under Assumptions B.1 and B.2, suppose that gradient-based optimization on $\mathcal{L}_{CE}$ converges to a neighborhood of θ^*, i.e., $\|\theta_T - \theta^*\| \le \epsilon$ for sufficiently large T. Then for any observation s and any two candidate actions $a_m, a_{m'}$ with $P^*(h(a_m) \mid s) > P^*(h(a_{m'}) \mid s)$ (i.e., the sequence-level probability of a_m's token sequence is higher under the true distribution), there exists $\epsilon_0 > 0$ such that for all $\epsilon < \epsilon_0$:*

$$\log \pi_{\theta_T}\big(h(a_m) \mid s\big) > \log \pi_{\theta_T}\big(h(a_{m'}) \mid s\big). \tag{5}$$

That is, the verifier's log-probability score preserves the true quality ranking at the sequence level.

Proof. Recall that the sequence-level probability factorizes as $\pi_\theta(h(a) \mid s) = \prod_{\ell=1}^{L} \pi_\theta(c_\ell \mid s, c_{<\ell})$, hence $\log \pi_\theta(h(a) \mid s) = \sum_{\ell=1}^{L} \log \pi_\theta(c_\ell \mid s, c_{<\ell})$. By Assumption B.1, $\pi_{\theta^*}(h(a) \mid s) = P^*(h(a) \mid s)$ for all a, s.

By continuity of $\pi_\theta(c_\ell \mid s, c_{<\ell})$ in θ (Assumption B.2(ii)) and the composition with the continuous log function (noting that probabilities are bounded away from zero on the support of P^*), $\log \pi_\theta(h(a) \mid s)$ is continuous in θ for any fixed a, s. Therefore, for any $\delta > 0$ there exists $\epsilon_0 > 0$ such that $\|\theta - \theta^*\| < \epsilon_0$ implies $|\log \pi_\theta(h(a) \mid s) - \log \pi_{\theta^*}(h(a) \mid s)| < \delta$ for all a, s in the support.

Since $\pi_{\theta^*}(h(a_m) \mid s) = P^*(h(a_m) \mid s) > P^*(h(a_{m'}) \mid s) = \pi_{\theta^*}(h(a_{m'}) \mid s)$, we have $\log P^*(h(a_m) \mid s) > \log P^*(h(a_{m'}) \mid s)$. Let $\gamma = \log P^*(h(a_m) \mid s) - \log P^*(h(a_{m'}) \mid s) > 0$. Choosing $\delta = \gamma/3$ and the corresponding ϵ_0, for $\|\theta_T - \theta^*\| < \epsilon_0$:

$$\log \pi_{\theta_T}(h(a_m) \mid s) > \log P^*(h(a_m) \mid s) - \gamma/3, \tag{6}$$

$$\log \pi_{\theta_T}(h(a_{m'}) \mid s) < \log P^*(h(a_{m'}) \mid s) + \gamma/3. \tag{7}$$

Therefore $\log \pi_{\theta_T}(h(a_m) \mid s) - \log \pi_{\theta_T}(h(a_{m'}) \mid s) > \gamma - 2\gamma/3 = \gamma/3 > 0$. □

B.2 Relaxation of Realizability

In practice, the model class may not contain the exact Bayes predictor. We show that the result holds approximately under misspecification.

Proposition B.4 (Approximate monotonicity under misspecification)**.** *Without Assumption B.1, let* $\theta^* = \arg\min_{\theta\in\Theta} \mathcal{L}_{CE}(\theta)$ *be the best-in-class minimizer. Define the log-space approximation error* $\varepsilon_{\text{approx}} = \max_{s,a} |\log \pi_{\theta^*}(h(a) \mid s) - \log P^*(h(a) \mid s)|$ *and the optimization residual* $\epsilon_{\text{opt}} = \max_{s,a} |\log \pi_{\theta_T}(h(a) \mid s) - \log \pi_{\theta^*}(h(a) \mid s)|$*. Then the scoring monotonicity holds for all candidate pairs satisfying* $\log P^*(h(a_m) \mid s) - \log P^*(h(a_{m'}) \mid s) > 2(\varepsilon_{\text{approx}} + \epsilon_{\text{opt}})$.

Proof. By triangle inequality in log-space:

$$\begin{aligned} &|\log \pi_{\theta_T}(h(a) \mid s) - \log P^*(h(a) \mid s)| \\ &\quad \le |\log \pi_{\theta_T}(h(a) \mid s) - \log \pi_{\theta^*}(h(a) \mid s)| + |\log \pi_{\theta^*}(h(a) \mid s) - \log P^*(h(a) \mid s)| \\ &\quad \le \epsilon_{\text{opt}} + \varepsilon_{\text{approx}}. \end{aligned} \tag{8}$$

Applying the same argument as Theorem C.5 with total log-space error $\epsilon_{\text{opt}} + \varepsilon_{\text{approx}}$ replacing ε_{cal} yields the result. □

Remark B.5 (Approximation and optimization). Proposition B.4 separates limitations of the model class from incomplete optimization. Increasing model capacity may reduce approximation error, whereas improving optimization may reduce the residual relative to the best-in-class predictor. Ranking is preserved only when the candidate log-probability gap exceeds twice their combined error.

C Proof of Verifier Soundness via Calibration Theory

We establish that the cross-entropy-trained bucket head provides a theoretically grounded scoring function for action verification. The analysis is architecture-agnostic and applies directly to the VLMs used in our experiments; the guarantees are conditional on the stated assumptions and on sufficiently accurate estimation of the demonstration distribution.

C.1 Setup and Notation

Let the training set be $\mathcal{D} = \{(s_j, a_j)\}_{j=1}^n$, where s_j is an observation and a_j is the corresponding expert action. The tokenizer $h : \mathcal{A} \to \{1, \dots, C\}^L$ maps an action chunk to a token sequence of length L. The VLM backbone π_θ defines an autoregressive distribution over token sequences conditioned on the observation:

$$\pi_\theta(c \mid s) = \prod_{\ell=1}^{L} \pi_\theta(c_\ell \mid s, c_{<\ell}), \tag{9}$$

where each factor $\pi_\theta(c_\ell \mid s, c_{<\ell})$ is a softmax over the vocabulary $\{1, \dots, C\}$. The true conditional distribution is $P^*(c \mid s) = \Pr[h(a) = c \mid s]$, representing the probability that a demonstration action at observation s produces token sequence c.

Notation convention for the proofs below. For notational compactness, we use $\pi_\theta(c \mid s)$ and $P^*(c \mid s)$ to denote the *sequence-level* probabilities $\pi_\theta(h(a) \mid s)$ and $P^*(h(a) \mid s)$ throughout this section. All results hold under the autoregressive factorization above, since the NLL loss decomposes into per-token cross-entropies, each of which is a strictly proper scoring rule.

C.2 Proper Scoring Rule Foundation

Definition C.1 (Strictly proper scoring rule [9])**.** A scoring rule $S : \mathcal{P} \times \mathcal{Y} \to \mathbb{R}$ is *strictly proper* if, for all distributions P over $\mathcal{Y}$:

$$\mathbb{E}_{y\sim P}[S(P, y)] > \mathbb{E}_{y\sim P}[S(q, y)] \quad \forall\, q \neq P. \tag{10}$$

Proposition C.2 (Cross-entropy as strictly proper scoring rule)**.** *The negative log-likelihood scoring rule $S(q, y) = \log q(y)$ is strictly proper. Consequently, the cross-entropy loss:*

$$\mathcal{L}_{CE}(\theta) = -\mathbb{E}_{(s,c)\sim\mathcal{D}_{bucket}}[\log \pi_\theta(c \mid s)] \tag{11}$$

has a unique global minimizer (over all measurable functions) at $\pi_\theta(\cdot \mid s) = P^*(\cdot \mid s)$ for each s.

Proof. This is a classical result. By the information inequality (Gibbs' inequality), for any distributions p and q over $\{1, \ldots, C\}$:

$$-\sum_{c=1}^{C} p(c) \log p(c) \leq -\sum_{c=1}^{C} p(c) \log q(c), \tag{12}$$

with equality if and only if $p = q$. Taking $p = P^*(\cdot \mid s)$, the expected cross-entropy $\mathbb{E}_s[-\sum_c P^*(c \mid s) \log \pi_\theta(c \mid s)]$ is uniquely minimized when $\pi_\theta(\cdot \mid s) = P^*(\cdot \mid s)$ for all s. □

C.3 Verifier Soundness Theorem

The proper scoring rule property has a direct consequence for action verification: as training converges, the bucket head's scoring function becomes consistent with the true quality ordering.

Definition C.3 (Calibration error)**.** The log-space calibration error of π_θ over a candidate set $\mathcal{A} = \{a_1, \ldots, a_M\}$ is:

$$\varepsilon_{\text{cal}}(\theta) \coloneqq \mathbb{E}_s\left[\max_{m\in\{1,\ldots,M\}} \left|\log \pi_\theta\big(h(a_m) \mid s\big) - \log P^*\big(h(a_m) \mid s\big)\right|\right], \tag{13}$$

where $\pi_\theta(h(a_m) \mid s)$ and $P^*(h(a_m) \mid s)$ denote the sequence-level probabilities under the autoregressive factorization (Section C, Setup and Notation).

Definition C.4 (Quality gap)**.** For two candidates $a_m, a_{m'}$ at observation s, the *log-probability quality gap* is:

$$\gamma(a_m, a_{m'}, s) \coloneqq \log P^*\big(h(a_m) \mid s\big) - \log P^*\big(h(a_{m'}) \mid s\big). \tag{14}$$

When $\gamma > 0$, candidate a_m is "better" in the sense that its token-sequence probability under the expert distribution is higher.

Theorem C.5 (Verifier soundness)**.** *Suppose the log-space calibration error satisfies $\varepsilon_{\text{cal}}(\theta) < \gamma_{\min}/2$, where $\gamma_{\min} = \min_{s,m\neq m':\gamma(a_m,a_{m'},s)>0} \gamma(a_m, a_{m'}, s)$ is the minimum positive log-probability quality gap. Then for all observations s and all candidate pairs with $P^*(h(a_m) \mid s) > P^*(h(a_{m'}) \mid s)$:*

$$\log \pi_\theta\big(h(a_m) \mid s\big) > \log \pi_\theta\big(h(a_{m'}) \mid s\big), \tag{15}$$

i.e., the verifier's log-probability ranking is consistent with the true quality ordering. Equivalently, score(a_m) > score$(a_{m'})$ under the scoring function defined in Section 4.2.

Proof. By the triangle inequality and the definition of log-space calibration error:

$$\begin{aligned}
&\log \pi_\theta(h(a_m) \mid s) - \log \pi_\theta(h(a_{m'}) \mid s) \\
&\quad \geq \big[\log P^*(h(a_m) \mid s) - \varepsilon_{\text{cal}}\big] - \big[\log P^*(h(a_{m'}) \mid s) + \varepsilon_{\text{cal}}\big] && (16)\\
&\quad = \gamma(a_m, a_{m'}, s) - 2\varepsilon_{\text{cal}} && (17)\\
&\quad \geq \gamma_{\min} - 2\varepsilon_{\text{cal}} > 0. && (18)
\end{aligned}$$

□

Remark C.6 (Practical implications). Theorem C.5 characterizes when estimation error is small enough to preserve the demonstration-likelihood ordering. Cross-entropy provides a population objective for learning this distribution, but finite-data optimization and distribution shift can prevent the required error bound from holding. The result is therefore a sufficient condition for ranking consistency; identifying likelihood with task utility requires the additional alignment assumption discussed in Section 4.3.

D Co-Training Synergy: Theoretical Perspective

The empirical observation that co-training with the bucket classification loss accelerates flow-matching convergence (Figure 1) admits a natural explanation within the calibration framework.

Proposition D.1 (Multi-task regularization)**.** *Let $f_\theta = (f_\theta^{shared}, f_\theta^{fm}, f_\theta^{bucket})$ denote the decomposition into shared backbone, flow-matching head, and bucket head. The co-training objective $\mathcal{L}_{total} = \mathcal{L}_{fm} + \beta \cdot \mathcal{L}_{bucket}$ implicitly regularizes the shared representation f_θ^{shared} by encouraging it to encode information sufficient for both action generation and action classification. Specifically, the bucket objective enforces that $f_\theta^{shared}(s)$ must distinguish observations where different action buckets are optimal—precisely the same discriminative structure needed for the flow-matching head to denoise toward the correct action mode.*

This explains why the gradient cosine similarity between the two objectives is positive ($+0.28$ at the default β=3, Table 6): both objectives push the backbone toward representations that are informative about which action is appropriate for a given observation. The bucket objective, by requiring explicit mode discrimination through its classification structure, provides a stronger gradient signal early in training when the flow-matching loss landscape is relatively flat.

D.1 Empirical Validation of Co-Training Dynamics

We measure the gradient alignment and representation quality throughout co-training to validate the multi-task regularization hypothesis.

Table 6: **Multi-objective training analysis.** All configurations use Qwen2.5-VL-3B trained for 64 epochs on LIBERO. $\|\Delta\theta\|/\|\theta_0\|$: relative parameter displacement. Cosine(∇_{fm}, ∇_{bucket}): average cosine similarity between the gradients of the two objectives during training.

Training Configuration	$\|\Delta\theta\|/\|\theta_0\|$	**Cos**(∇_{fm}, ∇_{bucket})	**LIBERO SR (%)**
Flow-matching Loss only	0.0081	–	91.9
Bucket Loss only	0.0063	–	–
Co-training (β=1)	0.0092	+0.31	95.4
Co-training (β=3, default)	0.0107	+0.28	96.8
Co-training (β=5)	0.0126	+0.24	96.2
Co-training (β=20)	0.0168	+0.19	94.7

Analysis. Table 6 reveals that the positive gradient cosine similarity between the two objectives ($+0.28$ at default β=3) confirms they push the backbone in partially aligned directions. This gradient coherence is the structural reason why co-training improves over single-objective training: the bucket objective provides an additional discriminative signal that accelerates the formation of action-relevant representations in the shared backbone. Excessive β ($\geq$ 20) degrades performance because the classification gradient dominates and the backbone overspecializes for bucket discrimination at the expense of smooth action generation.

D.2 Expected Selection Error Bound

We derive an upper bound on the expected sub-optimality incurred by selecting a candidate action via the bucket probability verifier rather than the true Q-function.

D.2.1 Setup

Consider M candidate actions $\mathcal{A} = \{a_1, \ldots, a_M\}$ sampled from the diffusion expert at observation s. Let

$$a^* = \arg\max_{a_m \in \mathcal{A}} Q(a_m \mid s) \tag{19}$$

denote the oracle-best candidate (highest true Q-value), and

$$\hat{a} = \arg\max_{a_m \in \mathcal{A}} \pi_\theta\big(h(a_m) \mid s\big) \tag{20}$$

the verifier's selection (highest bucket probability). Define the per-candidate sub-optimality gap $\Delta_m = Q(a^* \mid s) - Q(a_m \mid s) \geq 0$.

Note on Q. The action-value function $Q(a \mid s)$ here denotes the oracle expected return (e.g., task success probability) and serves purely as an analytical reference for measuring sub-optimality. It is *not* a learned component of our system—TokenPB does not train or deploy any Q-network. The bound quantifies how much regret is incurred by using the bucket probability $\pi_\theta(h(a) \mid s)$ as a proxy for Q.

D.2.2 Crude Bound

Let $q_m = \Pr[\hat{a} = a_m]$ denote the probability that the verifier selects candidate m (accounting for stochasticity in the sampling process and possible ties). The expected selection error is:

$$\mathbb{E}\big[Q(a^* \mid s) - Q(\hat{a} \mid s)\big] = \sum_{m=1}^{M} q_m \, \Delta_m \;\leq\; \delta \cdot \Delta_{\max}, \tag{21}$$

where $\delta = \Pr[\hat{a} \neq a^*] = 1 - q_{m^*}$ is the misselection probability and $\Delta_{\max} = \max_m \Delta_m$ is the worst-case gap. This bound is loose because it ignores the structure of q_m—specifically, that the scoring monotonicity (Theorem B.3) concentrates probability mass on high-Q candidates.

D.2.3 Tight Bound via Gibbs Modeling

To obtain a tighter characterization, we model the verifier's selection mechanism more precisely. The bucket log-probability $p_m = \log \pi_\theta(h(a_m) \mid s)$ serves as the scoring function. Since the $\arg\max$ selector is deterministic given $\{p_m\}$, we account for uncertainty by modeling the effective selection as a Gibbs distribution with inverse temperature $\lambda > 0$:

$$q_m = \frac{\exp(\lambda \cdot p_m)}{\sum_{m'=1}^{M} \exp(\lambda \cdot p_{m'})}. \tag{22}$$

This models the regime where the $\arg\max$ is "soft"—appropriate when score differences are moderate relative to noise. The $\arg\max$ selector is recovered as $\lambda \to \infty$.

Proposition D.2 (Selection error upper bound)**.** *Under the Gibbs selection model with inverse temperature λ, and assuming the scoring monotonicity (Theorem B.3) holds with calibration error bounded by ε_{cal}, the expected selection error satisfies:*

$$\mathbb{E}\big[Q(a^* \mid s) - Q(\hat{a} \mid s)\big] \;\leq\; \frac{\log M}{\lambda} + L_Q \cdot \varepsilon_{\text{cal}}, \tag{23}$$

where L_Q is the Lipschitz constant of Q with respect to the log-probability metric (i.e., the maximum Q-variation between candidates whose log-probabilities $p_m = \log \pi_\theta(h(a_m) \mid s)$ differ by ε_{cal}).

Proof. The first term follows from the standard regret bound for Gibbs sampling. For any distribution q over M items with rewards r_m and $q_m \propto \exp(\lambda \cdot r_m)$:

$$\max_m r_m - \sum_m q_m r_m \leq \frac{\log M}{\lambda}. \tag{24}$$

This is a direct consequence of the log-sum-exp inequality: $\max_m r_m \leq \frac{1}{\lambda} \log \sum_m \exp(\lambda r_m) \leq \max_m r_m + \frac{\log M}{\lambda}$.

In our setting, if p_m were a perfect proxy for $Q(a_m \mid s)$ (up to affine transformation), this bound would apply directly with $r_m = Q(a_m \mid s)$. The scoring monotonicity guarantees that p_m is monotonically related to the true quality $P^*(h(a_m) \mid s)$, which in turn correlates with $Q(a_m \mid s)$ through the expert demonstration distribution. The deviation from a perfect proxy arises from calibration error: when $\pi_\theta \neq P^*$, the bucket probability may mis-rank candidates by at most

$$|p_m - p_m^{\text{ideal}}| \leq \varepsilon_{\text{cal}}. \tag{25}$$

The resulting additional regret from this perturbation is bounded by $L_Q \cdot \varepsilon_{\text{cal}}$ (since the Q-function varies by at most L_Q per unit of log-probability perturbation), yielding the second term. □

Table 7: **Action Expert Model Parameters.**

Category	Parameter	Value
Model Size	Number of layers	16
	Hidden size	1536
	Number of attention heads	32
	Parameter size	$\sim$500M
Action Config	De-noising step	4
	Number of candidate action chunks M	16
Layer Structure	Self-attention layer	All **even** layers
	Cross-attention layer	All **odd** layers
	Dropout	0.2
Activation	Activation function	GELU

D.2.4 Interpretation

The selection bound separates the effect of a finite-temperature selector from estimation error. The $\log M/\lambda$ term decreases as the selector becomes sharper for a fixed candidate set, while the calibration term captures the effect of imperfect likelihood estimates. This interpretation depends on the assumed relationship between demonstration likelihood and task utility; the likelihood objective alone does not establish that relationship.

E Experimental Settings

Experimental settings mainly include the key parameters of the action expert model, training configurations, and the evaluation criteria for real-world experiments.

E.1 Action Expert Model

Inspired by GR00T-N1, our action expert utilizes an alternating sequence of DiT and self-attention during inference, as detailed in Table 7. To balance training efficiency with strong generalization, we implemented a 16-layer Transformer framework where self-attention and cross-attention layers are interleaved. With a total parameter count of roughly 500M, the model is trained entirely from scratch using random weights for each experiment.

E.2 Training Settings

Our experiments are primarily conducted on LIBERO, RoboTwin2.0, and in real-world environments. For each benchmark, we use its corresponding dataset to co-train the action expert for continuous actions and the VLM for discrete action tokens. For the evaluation stage, we test the performance of the three inference schemes—auto-regression, diffusion, and the TokenPB framework—using the same checkpoint. Refer to Table 8 for specific training settings.

E.3 Complete TokenPB Workflow

We present the full TokenPB pipeline in Algorithm 1 (co-training) and Algorithm 2 (inference with verification).

Key design notes. (1) The co-training loss $\mathcal{L}_{\text{total}}$ jointly trains both the VLM backbone and the action expert via a shared hidden feature f_c, allowing the bucket classification objective to shape action-relevant representations that also benefit flow-matching generation. (2) At inference time, all M candidates share the same observation prefix, so their token sequences can be scored in a

Table 8: Training Configuration on LIBERO, CALVIN, RoboTwin2.0 and Real-world benchmark.

Category	Parameter	Real-World	LIBERO	CALVIN	RoboTwin2.0*
Dataset Statistics	Total episodes	2005	1693	17870	2500
	Total frames	366152	273465	1071743	549787
	Task	4	40	389	50
	Instruction type	741	40	389	2420
	FPS	5	10	10	50
Training Parameters	Batch size	144	144	144	120
	Epoch	32	64	16	96
	Learning rate	5×10^{-5}	5×10^{-5}	5×10^{-5}	2×10^{-5}
	Weight decay	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Other Settings	Seed	42	42	42	42
	Discrete tokenizer	Textual FAST	Textual FAST	Textual FAST	Textual FAST
	Action chunk length H	5	16	16	32
	Co-training loss coefficient β	3	3	3	1

* In actual training, fine-tuning is performed on every single task, consistent with the official settings.

Algorithm 1 TokenPB Co-Training

Require: Training set $\mathcal{D} = \{(s_j, a_j)\}_{j=1}^{n}$; VLM backbone f_θ; diffusion action expert g_ϕ; tokenizer h; token vocabulary size C; co-training loss coefficient β
Ensure: Co-trained parameters (θ^*, ϕ^*)

Initialize VLM from pretrained weights θ_0; initialize action expert randomly ϕ_0
for each mini-batch $\{(s, a^{\text{demo}})\} \sim \mathcal{D}$ **do**
  $f_c \leftarrow f_\theta(s)$ ▷ observation-conditioned VLM feature
  $c^{\text{demo}} = (c_1, \ldots, c_L) \leftarrow h(a^{\text{demo}})$ ▷ tokenize demo action into sequence of length L
  $\mathcal{L}_{\text{bucket}} \leftarrow -\frac{1}{L}\sum_{\ell=1}^{L} \log \pi_\theta(c_\ell \mid s, c_{<\ell})$ ▷ autoregressive NLL over token sequence
  Sample noise $\epsilon \sim \mathcal{N}(0, I)$, timestamp $\tau \sim \text{Uniform}(0, 1)$
  $a^\tau \leftarrow \tau \cdot a^{\text{demo}} + (1-\tau)\cdot\epsilon$ ▷ noisy action at time τ
  $\mathcal{L}_{\text{fm}} \leftarrow \|g_\phi(a^\tau, \tau, f_c) - (a^{\text{demo}} - \epsilon)\|^2$ ▷ flow-matching objective
  $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{fm}} + \beta \cdot \mathcal{L}_{\text{bucket}}$
  Update $(\theta, \phi) \leftarrow (\theta, \phi) - \eta\nabla_{(\theta,\phi)}\mathcal{L}_{\text{total}}$
end for
return (θ^*, ϕ^*)

single batched forward pass via teacher-forcing; the verification overhead is therefore nearly constant with respect to M. (3) The total inference cost is approximately two VLM forward passes (one for drafting, one for verification) plus M lightweight flow-matching sampling steps (4-step Euler integration each), making TokenPB nearly constant-cost with respect to M.

E.4 Real-World Experiment Scoring Details

For each rollout, we record a normalized success rate $S \in [0, 1]$, representing the percentage of task completion. This metric accounts for partial completion, which is common in real-world manipulation, and is more informative than binary success indicators. To ensure clarity, we report the success rate as Success Rate $= 100\% \times S$ in the main paper, while maintaining $S \in [0, 1]$ in the appendix tables.

- **Push Blocks**: In the block-pushing task, the total score is comprised of a partial score and a completion score. For example, in the "push CAT forward" task, pushing the "C", "A", and "T" blocks accounts for 25% of the partial score each. The remaining 25% is awarded if all three blocks are successfully pushed and no other blocks are moved.
  For each block-pushing trial, we assign an individual score $s \in [0, 1]$ based on the final block configuration, using the following rubric:
  - $s = 0.0$: The current block remains untouched throughout the experiment.
  - $s = 0.3$: The current block is touched but not successfully pushed.
  - $s = 0.5$: The current block is pushed but did not reach the designated finish line.

Algorithm 2 TokenPB Inference: Multi-Candidate Verification

Require: Observation s_t; co-trained VLM f_{θ^*}; diffusion action expert g_{ϕ^*}; number of candidates M
Ensure: Selected action chunk $\hat{a}^*$

// Stage 1: Draft — generate M candidate action chunks
$f_c \leftarrow f_{\theta^*}(s_t)$ ▷ single VLM forward pass for observation feature
for $m = 1, \ldots, M$ **do**
  Sample $\epsilon_m \sim \mathcal{N}(0, I)$ ▷ independent noise seeds
  $a_m \leftarrow g_{\phi^*}(\epsilon_m, f_c)$ ▷ 4-step Euler ODE integration
end for
$\mathcal{A}_{s_t} \leftarrow \{a_1, \ldots, a_M\}$

// Stage 2: Verify — score candidates via token-sequence probability
for $m = 1, \ldots, M$ **do**
  $c^{(m)} = (c_1^{(m)}, \ldots, c_L^{(m)}) \leftarrow h(a_m)$ ▷ tokenize candidate
  $\text{score}(a_m) \leftarrow \frac{1}{L} \sum_{\ell=1}^{L} \log \pi_{\theta^*}(c_\ell^{(m)} \mid s_t, c_{<\ell}^{(m)})$ ▷ teacher-forcing scoring
end for
$\hat{a}^* \leftarrow \arg\max_{a_m \in \mathcal{A}_{s_t}} \text{score}(a_m)$
return $\hat{a}^*$

  - $s = 1.0$: The current block is successfully pushed past the finish line.

  We define s^r as the aggregate score for the N target blocks in the instructions, and s^w as the penalty score for the M unmentioned blocks. Ideally, an optimal trajectory should successfully push all mentioned blocks while leaving the unmentioned blocks untouched, leading to the following scoring criteria:

$$S_{\text{push}} = \frac{1}{N+1}(\sum_{i=1}^{N} s_i^r + 1 - \frac{1}{M}\sum_{i=1}^{M} s_i^w).$$

- **Clean Table**: The table-cleaning task is categorized into three distinct phases: opening the drawer, placing all items (typically three) from the table into the drawer sequentially, and closing the drawer.

  We define the score for opening and closing a drawer as binary values, $s_o, s_c \in \{0, 1\}$, where 1 indicates the drawer was successfully manipulated, and 0 indicates manipulation failure or no attempt. For each pick-up attempt, we assign an individual nesting score $s \in [0, 1]$ using the following rubric based on the final block configuration:

  - $s = 0.0$: The current object remains untouched throughout the experiment.
  - $s = 0.8$: The current object is picked up but not successfully placed in the drawer.
  - $s = 1.0$: The current object is successfully picked up and placed in the drawer.

  We denote s^d as the score for the N objects on the table. The final score for the table-cleaning task is calculated as follows:

$$S_{\text{clean}} = \frac{1}{N+2}(s_o + s_c + \sum_{i=1}^{N} s_i^d)$$

- **Pick & Place**: In the pick-and-place task, the total score is comprised of a 60% completion component and a 40% accuracy component. The completion score s^d is awarded based on the successful grasping and placement of the N target objects specified in the instruction, following the same protocol as the table-cleaning task. The accuracy score s^r refers to the proportion of N correct objects identified by the model among a total of M objects. Notably, attempts are counted towards the accuracy score even if the grasp ultimately fails. we define score $s \in \{0, 1\}$ based on whether the object is identified or not, For each object j, define $r_j = 1$ if it is mentioned; define a_j=1 if the agent attempts to grasp it at least once.

$$S_{\text{pick}} = 60\% \times \frac{1}{N}\sum_{i=1}^{N} s_i^d + 40\% \times (1 - \frac{1}{M}\sum_{j=1}^{M}[a_j \neq r_j])$$

- **Hang Cups**: The "hang cups" task consists of two primary processes: picking up the cup and hanging the cup. To better evaluate the model's sensitivity to instruction-specified orientations, we decompose the task into five sequential components: moving above the corresponding target s_m, successfully grasping the cup s_g, moving to the vicinity of the hook in the corresponding orientation s_h, aligning the cup handle with the hook s_a, and releasing the gripper to successfully hang the cup s_r:

$$S_{\text{cup}} = \frac{1}{5} \times (s_m + s_g + s_h + s_a + s_r)$$

E.5 Statistical Analysis of Real-World Results

To quantify the reliability of our real-world evaluation, we report the number of evaluation rollouts per task, standard deviations, 95% bootstrap confidence intervals, and paired t-test p-values comparing TokenPB against the single-sample baseline (from the same co-trained checkpoint). For the in-distribution setting, each instruction variant is evaluated with 3 independent rollouts under randomized initial object configurations. For the unseen setting, we increase the evaluation density to 18 rollouts per instruction to obtain more reliable estimates under distribution shift. The total rollout counts are shown in Tables 9 and 10.

Table 9: **Statistical analysis of Real-World (In-Distribution) results.** n: total evaluation rollouts per task. $\mu \pm \sigma$: mean success rate (%) $\pm$ standard deviation across rollouts. CI: 95% bootstrap confidence interval (1000 resamples). p: paired t-test p-value comparing TokenPB vs. single-sample baseline. All models use Qwen2.5-VL-3B.

Task	n	**Baseline** ($\mu \pm \sigma$)	**TokenPB** ($\mu \pm \sigma$)	**TokenPB 95% CI**	p**-value**
Push Blocks	33	56.2 ± 18.4	75.4 ± 15.7	[70.1, 80.8]	< 0.001
Clean Table	24	52.5 ± 21.6	72.5 ± 17.3	[65.4, 79.6]	< 0.005
Pick & Place	30	45.5 ± 22.1	75.0 ± 16.8	[68.8, 81.2]	< 0.001
Hang Cups	24	43.7 ± 5.8	62.5 ± 5.2	[60.4, 64.6]	< 0.001
Overall	**111**	49.5 ± 18.7	71.4 ± 15.1	[68.6, 74.2]	< 0.001

Table 10: **Statistical analysis of Real-World (Unseen) results.** Each of the 8 unseen instructions is evaluated with 18 independent rollouts ($n = 144$ total). All models use Qwen2.5-VL-3B.

Task	n	**Baseline** ($\mu \pm \sigma$)	**TokenPB** ($\mu \pm \sigma$)	**TokenPB 95% CI**	p**-value**
Push Blocks	54	5.0 ± 8.7	45.0 ± 20.4	[39.5, 50.5]	< 0.001
Clean Table	18	8.0 ± 13.8	24.0 ± 18.2	[15.5, 32.5]	< 0.01
Pick & Place	54	28.0 ± 19.5	36.0 ± 17.8	[31.3, 40.7]	< 0.01
Hang Cups	18	32.0 ± 5.1	40.0 ± 4.5	[37.9, 42.1]	< 0.001
Overall	**144**	18.3 ± 16.8	36.3 ± 18.9	[33.2, 39.4]	< 0.001

As shown in Tables 9 and 10, the in-distribution real-world improvements are statistically significant across all four tasks (all $p < 0.001$), with non-overlapping 95% confidence intervals between the TokenPB and baseline conditions. For the unseen environment, the overall improvement (+18.0 points) achieves strong statistical significance ($p < 0.001$) with 144 total rollouts, and all four task categories reach significance individually (all $p < 0.01$). Across both settings, the Hang Cups task exhibits substantially lower variance ($\sigma \approx 5$) than the other tasks, reflecting its more constrained action space and deterministic success criteria—the cup either hooks onto the rack or does not, leaving little room for partial-credit variability. This low variability yields tight confidence intervals and strong significance even with fewer rollouts. We note that the relatively large standard deviations observed in Push Blocks, Clean Table, and Pick & Place (9–22%) are characteristic of real-world robotic manipulation evaluations where environmental stochasticity (object pose variation, gripper slip, lighting changes) contributes substantial trial-to-trial variability. Despite this inherent noise, the consistent improvement direction across all tasks and the aggregate statistical significance support the robustness of the reported gains.

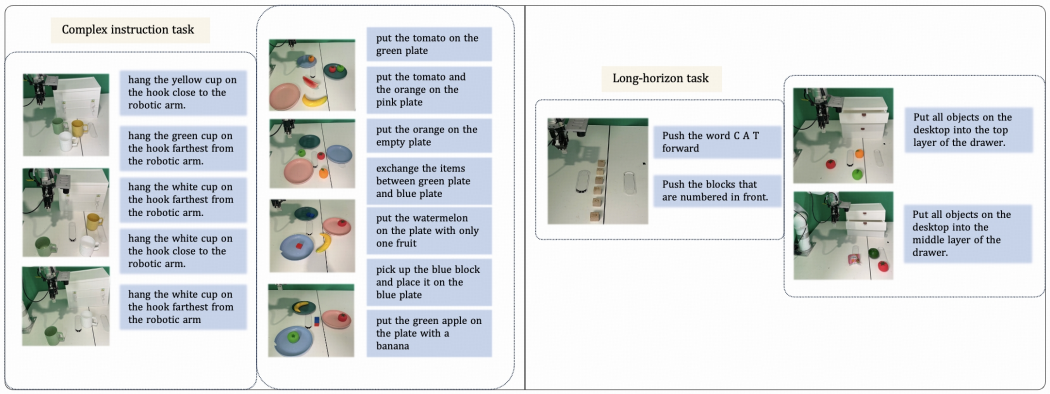

Figure 7: **Partial real machine environment and instruction diagram.** Our testing is based on the xArm robotic arm and DaHuan gripper. In real-world we collect 2000 trajectories and select for testing in this work.

F Task List of Real-World Tasks

We collected over 2,000 real-world trajectories involving more than 50 distinct objects in a physical robot environment. Notably, certain task configurations were excluded to serve as held-out scenarios for evaluation. For the Unseen tasks, neither the instructions nor the environments were present in the training set.
Our primary test environment is illustrated in Figure 7. The evaluation includes 34 distinct instruction types; to ensure representative results, some instructions were sampled multiple times in proportion to their natural frequency in the dataset.

F.1 Push Blocks Task

- Push the word C A T forward.
- Push the word G L M forward.
- Push the word D O G forward.
- Push the word V L A forward.
- Push the word R O S forward.
- Push the word I C E forward.
- Push the word Z H I P U forward.
- Push the word F L Y forward.
- Push the word L I K E forward.
- Push the word B O X forward.
- Push the blocks that are numbered in front.

F.2 Clean Table Task

- Put all objects on the desktop into the top layer of the drawer.
- Put all objects on the desktop into the middle layer of the drawer.

F.3 Pick & Place Task

- Put the tomato on the green plate.
- Put the tomato and the orange on the pink plate.
- Put the orange on the empty plate.
- exchange the items between the green plate and the blue plate.
- Put the green apple on the plate with a banana.

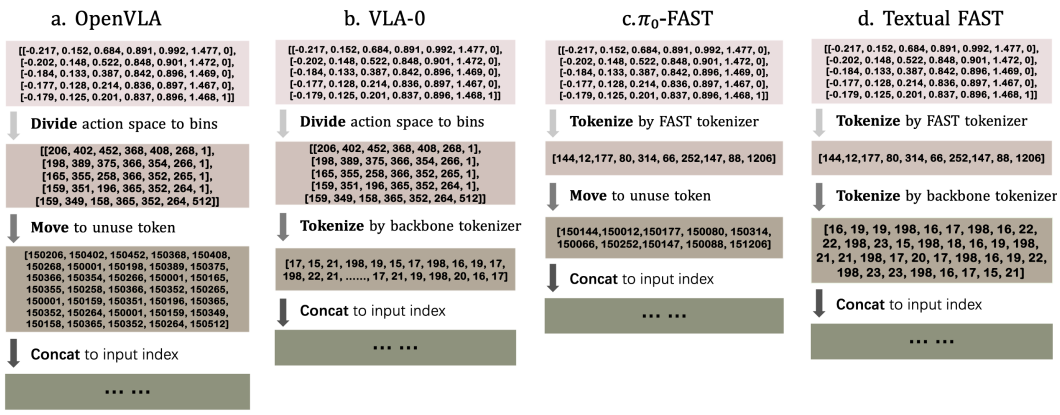

Figure 8: **Examples of four discrete action token representation methods.** Assume VLM tokenizer encode '\n' to 198, '0' to 14, '1' to 15, '2' to 16 and so on.

- Pick up the blue block and place it on the blue plate.
- Put the banana on the plate with two blue blocks.
- Put all items on the table into the pink plate.
- Put the orange on the plate with a red block.
- Put the watermelon on the plate with only one fruit.

F.4 Hang Cups Task

- Hang the white cup on the hook farthest from the robotic arm.
- Hang the white cup on the hook close to the robotic arm.
- Hang the yellow cup on the hook close to the robotic arm.
- Hang the green cup on the hook farthest from the robotic arm.

F.5 Unseen Task

- Exchange the position of the tomato and the pear.
- Put the tomato on the plate that has only one item of fruit.
- put as more as possible fruit to the green plate.
- Push the blocks with 7 multiplied by 6 forward.
- Push ahead the two blocks whose sum is over 14.
- Push the blocks that have numbers on them forward.
- Hang the green cup on the hook close to the robotic arm.
- Put all the blue blocks into the top layer of the drawer.

G Four Action Tokenization Methods

In this section, we illustrate the tokenization differences among four discrete encoding methods, using a concrete example where **action chunk**=5 and **action dim**=7 (see Figure 8).

G.1 Action Bins

The Action Bins method, adopted by RT-2 and OpenVLA, determines range boundaries using the 1st and 99th percentiles of every dimension of action space and divides each action dimension into 512 bins. Although simple and intuitive, such methods perform poorly when learning dexterous skills with high-frequency control. Highly correlated action tokens diminish the effectiveness of the

next token prediction objective used in auto-regressive VLAs. Intuitively, in such cases low token prediction loss can often be achieved with mappings as trivial as simply copying the most recent action token, leaving models in poor local optima [24].

G.2 FAST

The FAST tokenizer transforms continuous robot actions into tokens through the following steps:
Quantile Normalization: Raw action chunks are normalized such that the 1st and 99th quantiles of the training data map to $[-1, 1]$. This ensures robustness to outliers and facilitates cross-embodiment training.
Discrete Cosine Transform (DCT): Each action dimension is converted into the frequency domain via DCT, transforming time-series signals into frequency coefficients.
Quantization: DCT coefficients are scaled by a hyperparameter γ and rounded to integers. This creates a sparse matrix where high-frequency components often reduce to zero, effectively compressing the signal.
Flattening: The sparse coefficient matrix is flattened into a one-dimensional vector of integers using a "column-first" order. By prioritizing low-frequency components across dimensions, this order improves the stability of auto-regressive prediction.
Byte Pair Encoding (BPE): The flattened sequence is further compressed into a shorter sequence of dense tokens using a trained BPE tokenizer. This step losslessly merges frequent coefficient combinations and eliminates redundant zero-valued components.
In the introduction to Textual FAST, we refer to this entire process simply as **FAST tokenization**.

G.3 VLA-0

Common VLM tokens and VLA action tokens often follow distinct distributions, which can lead to interference during the training process. VLA-0 proposes a simple method to circumvent this issue. First, VLA-0 divides each action dimension into several bins. Then, VLA-0 treats the sequence of actions as strings and encodes the action strings using VLM's tokenizer. In this way, the action sequence is processed similarly to numerical tasks in natural language processing, thereby reducing the modality gap. This optimization effectively accelerates convergence speed for VLAs that lack prior pre-training on discrete action tokens.

G.4 Textual FAST

Building upon VLA-0, Textual FAST replaces dividing each action dimension into several bins with processing using FAST tokenization. Specifically, Textual FAST first processes actions into action tokens via FAST tokenization, and subsequently treats the sequence of actions as a string and encodes this action string using the VLM's tokenizer.

Prediction architecture. The compressed FAST representation is serialized as numerical text and encoded with the VLM's existing tokenizer. Its tokens are entries in the pretrained vocabulary, so the original language-modeling head predicts the bucket-token sequence directly; no additional prediction head or vocabulary expansion is required. Here, "bucket head" refers to the existing language-modeling head trained with the auxiliary bucket-token objective. This design reuses the pretrained token representations and can be integrated into the evaluated VLM–action-expert architectures without changing their prediction heads. The co-training convergence results in Figure 1 and the tokenization comparison below support this choice.

Score normalization. We use the mean token log-likelihood in Section 4.2, dividing each candidate's sequence log-likelihood by its own token count. This reduces the direct dependence of the summed log-likelihood on sequence length. It is the implemented ranking rule, rather than an additional normalization of the scores across candidates.

Empirical comparison. To quantify the impact of the tokenization scheme on downstream verification performance, we train Qwen2.5-VL with each of the four methods under identical settings and evaluate on LIBERO with TokenPB (M=16). Table 11 summarizes the results. Among the four schemes, Textual FAST achieves the highest average success rate (96.8%), outperforming Action Bins (93.0%), FAST (94.4%), and VLA-0 (94.9%) by clear margins.

Table 11: **Impact of tokenization methods on LIBERO (%).** All variants use Qwen2.5-VL-3B with TokenPB (M=16). The first row (Qwen2.5-VL-Diffusion) reports the single-sample baseline without verification. ↑/↓ indicate improvement/degradation relative to this baseline.

LIBERO	Spatial	Object	Goal	Long	Avg.
Qwen2.5-VL-Diffusion	94.6	97.2	93.8	90.2	93.9
Action Bins	92.8	95.6	94.4	89.0	93.0 ↓
FAST	95.0	97.2	93.6	91.8	94.4 ↑
VLA-0	**96.8**	95.4	96.0	91.4	94.9 ↑
Textual FAST	96.6	**98.8**	**97.0**	**94.6**	**96.8** ↑

Table 12: **Test results on the LIBERO dataset.** "Model-FAST": FAST tokenizer-assisted auto-regression model. "Model-GR00T" and "Model-GR00T+TokenPB" share the same co-trained checkpoint (with both bucket classification and diffusion objectives) and differ only at inference: GR00T uses a single diffusion sample (M=1), while GR00T+TokenPB applies multi-candidate verification (M=16). Action chunk size per inference of our model is 16.

Models	Spatial	Object	Goal	Long	Avg.
Diffusion Policy [5]	78.3	92.5	68.3	50.5	72.4
HybridVLA [21]	93.6	90.6	90.2	84.4	90.2
FlowVLA [27]	93.2	95.0	91.6	72.6	88.1
SmolVLA [25]	93.0	94.0	91.0	77.0	88.8
OpenVLA-OFT [13]	<u>97.6</u>	98.4	**97.9**	94.5	<u>97.1</u>
UniVLA [3]	95.4	<u>98.8</u>	93.6	94.0	95.5
Octo [26]	78.9	85.7	84.6	51.1	75.1
OpenVLA [14]	84.7	88.4	79.2	53.7	76.5
π_0-FAST [10, 24]	90.0	86.0	95.0	73.0	86.0
VLA-0 [18]	94.4	97.6	93.0	90.6	93.9
GR00T-N1 [1]	94.4	97.6	93.0	90.6	93.9
π_0 [2]	96.8	<u>98.8</u>	95.8	85.2	94.2
$\pi_{0.5}$-KI [7, 18]	**98.0**	97.8	95.6	85.8	94.3
Discrete Diffusion VLA [13]	97.2	98.6	<u>97.4</u>	92.0	96.3
Qwen2.5-VL-FAST	90.2	88.6	96.2	76.4	87.9
Qwen2.5-VL-GR00T	94.6	97.2	93.8	90.2	93.9
Qwen2.5-VL-GR00T+TokenPB	96.6	<u>98.8</u>	97.0	<u>94.6</u>	96.8
InternVL3.5-FAST	89.6	92.8	93.6	82.8	89.6
InternVL3.5-GR00T	95.8	96.0	94.2	92.8	94.7
InternVL3.5-GR00T+TokenPB	<u>97.6</u>	**99.4**	97.2	**96.8**	**97.8**

Two complementary factors account for the advantage of Textual FAST. First, it inherits the strengths of both FAST and VLA-0: the DCT-based compression from FAST produces compact, low-redundancy frequency-domain representations, while the textual encoding from VLA-0 bridges the modality gap between action tokens and the VLM's pretrained vocabulary, facilitating more efficient co-training. Second, compared with Action Bins, Textual FAST distributes actions across a substantially larger number of hash buckets with lower per-bucket collision rates, yielding higher inter-bucket independence. This finer-grained and more orthogonal bucket structure enables the VLM to learn sharper probability distributions over action tokens, which in turn provides a more discriminative scoring signal for test-time verification.

Based on these results, we adopt Textual FAST as the default tokenization method in all experiments reported throughout this paper.

Table 13: **RoboTwin 2.0 Simulation Benchmark.**

Simulation Task	RDT		Pi0		ACT		DP		TokenPB		Qwen-Dif	
	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard	Easy	Hard
Adjust Bottle	81%	**75%**	90%	56%	**97%**	23%	**97%**	0%	**97%**	26%	96%	16%
Beat Block Hammer	**77%**	**37%**	43%	21%	56%	3%	42%	0%	42%	0%	35%	0%
Blocks Ranking RGB	3%	0%	19%	5%	1%	0%	0%	0%	**52%**	**24%**	38%	11%
Blocks Ranking Size	0%	0%	7%	**1%**	0%	0%	1%	0%	**22%**	0%	16%	0%
Click Alarmclock	61%	12%	**63%**	11%	32%	4%	61%	5%	52%	**22%**	50%	19%
Click Bell	**80%**	9%	44%	3%	58%	3%	54%	0%	54%	**13%**	51%	12%
Dump Bin Bigbin	64%	**32%**	83%	24%	68%	1%	49%	0%	**77%**	15%	68%	21%
Grab Roller	74%	43%	96%	**80%**	94%	25%	**98%**	0%	78%	46%	66%	44%
Handover Block	45%	**14%**	45%	8%	42%	0%	10%	0%	**62%**	6%	54%	5%
Handover Mic	90%	**31%**	**98%**	13%	85%	0%	53%	0%	96%	2%	91%	4%
Hanging Mug	**23%**	**16%**	11%	3%	7%	0%	8%	0%	11%	0%	2%	0%
Lift Pot	72%	9%	84%	**36%**	88%	0%	39%	0%	**89%**	9%	82%	7%
Move Can Pot	25%	12%	**58%**	**21%**	22%	4%	39%	0%	56%	11%	48%	14%
Move Pillbottle Pad	8%	0%	**21%**	**1%**	0%	0%	1%	0%	17%	1%	**21%**	0%
Move Playingcard Away	43%	11%	**53%**	**22%**	36%	0%	47%	0%	50%	17%	48%	6%
Move Stapler Pad	**2%**	0%	0%	**2%**	0%	0%	1%	0%	0%	0%	0%	0%
Open Laptop	59%	32%	**85%**	**46%**	56%	0%	49%	0%	55%	27%	39%	18%
Open Microwave	37%	20%	80%	**50%**	**86%**	0%	5%	0%	62%	39%	55%	37%
Pick Diverse Bottles	2%	0%	**27%**	**6%**	7%	0%	6%	0%	21%	2%	22%	1%
Pick Dual Bottles	42%	**13%**	**57%**	12%	31%	0%	24%	0%	45%	9%	40%	5%
Place A2B Left	3%	**1%**	**31%**	**1%**	1%	0%	2%	0%	16%	0%	4%	0%
Place A2B Right	1%	1%	**27%**	**6%**	0%	0%	13%	0%	18%	0%	6%	1%
Place Bread Basket	10%	2%	17%	**4%**	6%	0%	14%	0%	**23%**	2%	11%	0%
Place Bread Skillet	5%	**1%**	**23%**	**1%**	7%	0%	11%	0%	9%	**1%**	8%	**1%**
Place Burger Fries	50%	**27%**	**80%**	4%	49%	0%	72%	0%	68%	11%	45%	2%
Place Can Basket	19%	**6%**	**41%**	5%	1%	0%	18%	0%	21%	2%	15%	3%
Place Cans Plasticbox	6%	**5%**	34%	2%	16%	0%	**40%**	0%	11%	0%	14%	0%
Place Container Plate	78%	17%	**88%**	**45%**	72%	1%	41%	0%	82%	11%	68%	2%
Place Dual Shoes	4%	**4%**	**15%**	0%	9%	0%	8%	0%	12%	0%	8%	0%
Place Empty Cup	56%	7%	37%	**11%**	**61%**	0%	37%	0%	44%	3%	37%	0%
Place Fan	12%	2%	**20%**	**10%**	1%	0%	3%	0%	4%	1%	2%	0%
Place Mouse Pad	1%	0%	**7%**	**1%**	0%	0%	0%	0%	0%	0%	1%	0%
Place Object Basket	**33%**	**17%**	16%	2%	15%	0%	15%	0%	22%	4%	14%	1%
Place Object Scale	1%	**0%**	**10%**	**0%**	0%	**0%**	1%	**0%**	0%	**0%**	0%	**0%**
Place Object Stand	15%	5%	**36%**	**11%**	1%	0%	22%	0%	9%	0%	11%	0%
Place Phone Stand	15%	6%	**35%**	**7%**	2%	0%	13%	0%	17%	6%	11%	2%
Place Shoe	**35%**	**7%**	28%	6%	5%	0%	23%	0%	24%	2%	16%	0%
Press Stapler	41%	24%	**62%**	**29%**	31%	6%	6%	0%	55%	15%	44%	7%
Put Bottles Dustbin	21%	4%	**54%**	**13%**	27%	1%	22%	0%	28%	9%	14%	6%
Put Object Cabinet	33%	**18%**	**68%**	**18%**	15%	0%	42%	0%	55%	0%	42%	0%
Rotate QRcode	50%	5%	**68%**	**15%**	1%	0%	13%	0%	35%	9%	26%	12%
Scan Object	4%	**1%**	**18%**	**1%**	2%	0%	9%	0%	11%	0%	9%	0%
Shake Bottle Horizontally	84%	**51%**	**99%**	**51%**	63%	4%	59%	18%	87%	42%	83%	19%
Shake Bottle	74%	45%	**97%**	**60%**	74%	10%	65%	8%	86%	37%	84%	21%
Stack Blocks Three	2%	**0%**	**17%**	**0%**	0%	**0%**	0%	**0%**	2%	**0%**	0%	0%
Stack Blocks Two	21%	**2%**	**42%**	1%	25%	0%	7%	0%	17%	**2%**	18%	1%
Stack Bowls Three	51%	17%	**66%**	24%	48%	0%	63%	0%	61%	**26%**	52%	9%
Stack Bowls Two	76%	30%	**91%**	41%	82%	0%	61%	0%	84%	**48%**	75%	24%
Stamp Seal	1%	0%	3%	**4%**	2%	0%	2%	0%	**4%**	0%	0%	0%
Turn Switch	35%	15%	27%	**23%**	5%	2%	**36%**	1%	22%	9%	15%	7%
Average (%)	34.5	<u>13.7</u>	**46.4**	**16.3**	29.7	1.7	28.0	0.6	<u>39.3</u>	10.2	30.1	6.7

H Full Test Results on LIBERO and RoboTwin2.0

H.1 LIBERO

Settings. For the LIBERO benchmark, we trained our model on 1,693 trajectories derived from data replay process [14], and subsequently evaluated it on four tasks: Spatial, Object, Long, and Goal.

Baselines. In addition to auto-regressive baselines and diffusion VLAs, we compared our method against two main categories of models. First, we considered models that improve performance through inference framework modifications, such as Diffusion Policy, HybridVLA, VLA-0, and Discrete Diffusion VLA. Second, we included foundational models pre-trained on large-scale embodied datasets, such as RDT, π_0, GR00T-N1, and Octo.

For reference, we report published numbers for prior work when available in Table 12. Note that these results may differ in training data, action chunking, and evaluation protocol; our controlled comparisons are among Qwen / InternVL / $\pi_{0.5}$ variants in Table 1. The results in Table 12 demonstrate that our method consistently outperforms the baselines across various objects. Notably, even without additional pre-training, our model achieves success rates comparable to those of large-scale pre-trained

models on specific datasets. To further validate the effectiveness of the TokenPB framework, we also performed a series of ablation studies.

H.2 RoboTwin 2.0

A partial selection of the RoboTwin test results is presented in the main body of the paper. For the full benchmark results of Qwen2.5-VL-GR00T (Qwen-Dif) and Qwen2.5-VL-GR00T+TokenPB (TokenPB), please refer to Table 13.

In addition to the Qwen2.5-VL backbone, we evaluate TokenPB with InternVL3.5-2B and $\pi_{0.5}$ on selected RoboTwin2.0 (Hard) tasks to demonstrate cross-architecture generality. As shown in Table 14, TokenPB consistently improves performance across all model–task combinations, with particularly notable gains for $\pi_{0.5}$ on multi-step manipulation tasks (Bottle Cap and Ranking Block RGB).

Table 14: **RoboTwin2.0 (Hard) results with InternVL3.5-2B and $\pi_{0.5}$ backbones.** Success rates (%) on selected tasks. TokenPB uses M=16 candidates.

Method	Adjust Bottle	Move Pill Bottle Cap	Ranking Block RGB
InternVL3.5-2B	14.0	0	3.0
InternVL3.5-2B + TokenPB	**22.0**	0	**5.0**
$\pi_{0.5}$	39.0	4.0	11.0
$\pi_{0.5}$ + TokenPB	**43.0**	**11.0**	**23.0**

I Additional Baseline and Backbone Evaluations

I.1 Baseline Availability and Shared-Sampler Comparison

Table 2 distinguishes published results from our shared-sampler reproduction. At the time of our experiments, the released RoboMonkey checkpoints covered LIBERO-Long and SimplerEnv; RoVer's code and checkpoints were unavailable, so its CALVIN results use the published MoDE configuration. MG-Select targets autoregressive VLAs, and its published π_0-FAST results cannot be reproduced with TokenPB's flow-matching action expert. Differences in architecture, training data, and action chunking therefore remain in comparisons across these published systems. AsyncVLA's confidence-based correction is part of denoising, and its paper does not provide a corresponding unverified single-sample result.

For a controlled comparison, we train a RoboMonkey-Qwen2.5 verifier using Qwen2.5-VL-3B and more than 20 million downstream trajectory preference pairs, following the RoboMonkey training protocol. RoboMonkey-Qwen2.5 and TokenPB use the same $\pi_{0.5}$ sampler and M=16. Their shared single-sample baseline, the reproduced verifier, and the available MG-Select single-sample results are included as additional rows in Table 2. TokenPB improves the LIBERO average from 96.5% to 98.4%, compared with 97.3% for RoboMonkey-Qwen2.5; on SimplerEnv-WidowX the corresponding rates are 65.8%, 75.8%, and 71.3%.

Additional benchmarks. Table 15 extends the baseline evaluation to LIBERO-Plus, RoboCasa, and RoboTwin 2.0. On RoboCasa, TokenPB improves success from 54.9% to 56.4%, compared with 55.5% for RoboMonkey.

I.2 Comparison with Learned Q-Functions

Learning target and supervision. TokenPB estimates the observation-conditioned demonstration distribution, using only the observation–action pairs already available for behavior cloning. A Q-function instead estimates expected cumulative reward and requires reward supervision and temporal credit assignment. Success-focused demonstration datasets may provide limited reward contrast among near-expert candidates. TokenPB uses the finer distributional structure of these demonstrations for candidate ranking, but demonstration frequency is not a guarantee of global action optimality.

Table 15: **Additional baseline results on LIBERO-Plus, RoboCasa, and RoboTwin 2.0.** Success rates are reported in percent. The RoboTwin 2.0 result is the mean ± standard deviation over five training seeds.

Benchmark	Method	Success rate (%)
LIBERO-Plus	Single sample	76.5
LIBERO-Plus	RoboMonkey	77.6
LIBERO-Plus	TokenPB	76.9
RoboCasa	Single sample	54.9
RoboCasa	RoboMonkey	55.5
RoboCasa	TokenPB	56.4
RoboTwin 2.0	RoboMonkey	89.8 ± 0.5

Table 16: **Shared-sampler comparison with learned Q-functions on LIBERO (%).** All methods select among $M{=}16$ candidates. The upper block reports task success; the lower block reports oracle-best candidate selection accuracy on the diagnostic subset. Q-functions use replay-derived rewards, RoboMonkey-Qwen2.5 uses additional preference pairs, and TokenPB uses demonstrations alone.

Method	Spatial	Object	Goal	Long	Avg.
Task success rate					
Random selection	92.0	95.2	91.4	87.8	91.6
CQL Q-function	91.8	96.4	90.6	89.2	92.0
TD3+BC Q-function	92.6	96.0	92.4	91.4	93.1
IQL Q-function	93.4	97.8	92.2	91.6	93.7
RoboMonkey-Qwen2.5	97.0	98.4	95.8	92.4	95.9
TokenPB	96.6	98.8	97.0	94.6	96.8
Verification accuracy					
Random selection	5.7	6.4	6.1	6.8	6.3
IQL Q-function	46.1	35.5	29.6	40.9	38.2
TokenPB	86.2	87.6	81.8	89.4	86.2

Conditioning on the observation allows an action that is rare overall to receive a high score in an appropriate recovery state. The unseen-scene and recovery results in Tables 1 and 5 are consistent with this interpretation, without establishing reliable extrapolation to arbitrary unseen actions.

Protocol. We obtain binary success/failure rewards by replaying trajectories from the sampler's training dataset in simulation; unsuccessful replay outcomes provide failure cases. We train CQL, TD3+BC, and IQL critics and use each Q-function to rerank $M{=}16$ candidates from the same sampler. The Q-network receives the VLM context feature f_c and an action chunk; IQL uses expectile $\tau{=}0.7$. This comparison gives the critics replay-derived reward supervision in addition to the demonstrations. Such replay requires access to the training environment and may be unavailable for offline real-world data. The shared-sampler comparison here is separate from the $\pi_{0.5}$ comparison in Table 2.

Results. Table 16 shows that the learned critics yield smaller gains than TokenPB under this protocol, even with additional reward supervision. IQL is the strongest of the three Q-based scorers. CQL's conservative penalty, TD3+BC's policy regularization, and IQL's expectile-based updates address offline learning in different ways, but all retain a return-estimation objective. Limited reward contrast is a possible explanation for their weaker candidate discrimination here; these results do not establish a general limitation of value learning. The published V-GPS comparison on SimplerEnv-WidowX is retained in Table 2.

I.3 Computational Cost and GPU Memory

Comparison protocol. The inference-time comparison in Figure 3 enables the respective prefix-sharing and batching optimizations for RoboMonkey and MG-Select. Prefix caching is an established optimization. TokenPB's architectural advantage is that scoring directly reuses the observation-prefix

Table 17: **GPU memory scaling with Qwen2.5-VL-3B (GB).** RoboMonkey uses an external Qwen2.5-VL-3B verifier with prefix sharing enabled. The scoring allocation is the total minus the reported resident sampler and, where applicable, external-verifier footprints. Unmeasured entries are left blank.

Component / candidate count	TokenPB	RoboMonkey
Resident sampler	18.2	18.2
Additional verifier	0.0	15.9
Scoring allocation, $M=1$	0.1	1.2
Scoring allocation, $M=16$	1.1	2.1
Scoring allocation, $M=32$	2.7	3.3
Total, $M=1$	18.3	35.3
Total, $M=16$	19.3	36.2
Total, $M=32$	20.9	37.4
Total, $M=64$	23.6	–

KV cache already computed by the sampler. An external verifier must encode the observation with its own parameters before it can reuse that prefix across candidates. Prefix sharing and batched continuation scoring can be combined, but the external prefix encoding and verifier weights remain additional costs. MG-Select integrates scoring with autoregressive action-token generation, while TokenPB separates flow-matching generation from token-likelihood verification; component-level timing comparisons should account for this architectural difference.

Memory components. Model parameters do not grow with M. TokenPB also computes its observation-prefix KV cache once; candidate-specific token continuations require separate KV entries, whose total storage grows approximately linearly with M at fixed sequence length. The lightweight action expert shares the context feature f_c, while its temporary sampling workspace depends on the batching schedule. Table 17 reports measured total memory for Qwen2.5-VL-3B. The sampler's reported resident footprint is approximately 18.2 GB; the additional external verifier occupies approximately 15.9 GB. We report the remaining scoring allocation in aggregate to avoid counting the shared prefix twice.

At $M=16$, TokenPB uses approximately 19.3 GB compared with 36.2 GB for the external-verifier system. Its memory continues to grow with candidate count, despite prefix reuse, reaching approximately 23.6 GB at $M=64$.

I.4 VLM Backbones and Embodiment-Specific Pretraining

Metrics and setup. We compare VLM backbones under the same training setting within each benchmark. Verification accuracy is the fraction of diagnostic cases in which the scorer selects the clean candidate among $M=16$ candidates after random noise is injected into the other candidates. Recall accuracy measures selection of a ground-truth demonstration action from the training dataset manually inserted into the candidate set. These diagnostics use representative subsets rather than the full task-evaluation set and are distinct from downstream task success. Table 18 includes LIBERO, the more challenging LIBERO-Plus benchmark, and RoboCasa. Verification results from different diagnostic subsets are reported separately and should not be treated as identical estimates.

Backbone effects. On LIBERO, verification accuracy and task success follow the same ordering across backbones, whereas recall accuracy does not. A possible explanation is that a stronger co-trained sampler makes the inserted demonstration harder to distinguish from sampled candidates; the diagnostic does not isolate this mechanism. On LIBERO-Plus, Qwen3.5-4B has the highest verification accuracy, while InternVL3.5-2B has the highest reported overall success. Category-level results are given in Table 19. Within the Qwen family on RoboCasa, both verification accuracy and task success improve from Qwen2.5-VL to Qwen3-VL and Qwen3.5. These results indicate that backbone suitability depends on architecture, embodiment-relevant capabilities, and training data, rather than general multimodal capability alone. Resolution handling and spatial representations are possible contributors, but these experiments do not isolate a dominant architectural factor.

Table 18: **Backbone and pretraining comparisons (%).** Verification and recall accuracies use diagnostic subsets. Recall was measured only for the LIBERO backbone comparison. The pretraining rows use Qwen2.5-VL-3B and 10K hours of real-robot data to pretrain bucket-token prediction; other rows use the standard training setting.

Backbone / setting	Verif. acc.	Recall acc.	Success rate
LIBERO			
Qwen2.5-VL-3B	85.5	33.9	96.8
Qwen3.5-4B	86.0	37.1	97.4
InternVL3.5-2B	87.4	32.2	97.8
Qwen2.5-VL-3B + pretraining	88.2	–	98.6
LIBERO-Plus			
Qwen2.5-VL-3B	69.7	–	73.5
Qwen3.5-4B	76.4	–	76.9
InternVL3.5-2B	71.3	–	79.7
Qwen2.5-VL-3B + pretraining	83.3	–	81.5
RoboCasa			
Qwen2.5-VL	68.9	–	53.5
Qwen3-VL	84.6	–	56.4
Qwen3.5	84.9	–	56.7

Table 19: **LIBERO-Plus success rates by perturbation category (%).** All backbones use TokenPB.

Category	Qwen2.5-VL-3B	Qwen3.5-4B	InternVL3.5-2B
Background	92.7	96.7	97.6
Light	93.6	95.6	95.1
Language	88.4	86.7	87.0
Noise	74.6	79.1	80.8
Layout	79.7	83.1	79.0
Robot initialization	54.2	55.2	62.5
Camera	43.2	52.2	41.0
Overall	73.5	76.9	79.7

Embodiment-specific pretraining. Pretraining bucket-token prediction on 10K hours of real-robot data improves Qwen2.5-VL-3B on both LIBERO and LIBERO-Plus (Table 18). The gains are larger on LIBERO-Plus: verification accuracy increases from 69.7% to 83.3% and task success from 73.5% to 81.5%. This supports the value of embodiment-relevant pretraining beyond changing the VLM backbone.

J Societal Impacts

We believe that our work will not have an immediate negative impact on society, as its primary objective is to improve the reliability of Vision-Language-Action models by enabling action self-verification at test time.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction claim that TokenPB (i) unifies action generation and verification within a single VLA checkpoint, (ii) achieves consistent improvements across multiple benchmarks and backbones, and (iii) incurs nearly constant verification overhead. These claims are directly supported by the experimental results in Section 5 (Tables 1–3 and Figures 2–4). Limitations are discussed in Appendix 6.

Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion discusses two key limitations: GPU memory still grows with candidate set size M despite observation-prefix KV-cache reuse, and the frequency-based nature of the scoring signal—when multiple valid actions exist for the same observation, the verifier favors the most frequent rather than the optimal one.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 4.3 states the selection error bound with explicit assumptions (scoring monotonicity, calibration error). Full proofs are provided in the Appendix, including verifier soundness via calibration theory (Appendix C), scoring monotonicity (Appendix B), and expected selection error bound (Appendix D.2). All theorems and propositions are numbered and cross-referenced with their assumptions stated explicitly.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides complete training configurations for all benchmarks (Table 8 in Appendix E.2), action expert architecture details (Appendix E.1), the full algorithmic procedure in pseudocode (Algorithms 1–2 in Appendix E.3), evaluation protocols and scoring criteria (Appendix E.4), and all hyperparameters including learning rate, batch size, epochs, loss coefficients, and random seed. All simulation benchmarks (LIBERO, CALVIN, RoboTwin2.0) are publicly available.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code for training and inference has been submitted as supplementary material via the submission portal. All simulation benchmarks used (LIBERO, CALVIN, RoboTwin2.0) are publicly available. We plan to publicly release the code and pre-trained checkpoints upon acceptance.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 describes the experimental setup including model configurations, backbone choices, and benchmark details. The Appendix provides comprehensive training configurations (Table 8), action expert architecture (Table 7), dataset statistics, tokenization method comparison (Appendix G), real-world task descriptions (Appendix F), and scoring criteria (Appendix E.4).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For real-world experiments, Appendix E.5 reports standard deviations, 95% bootstrap confidence intervals (1000 resamples), and paired t-test p-values for both in-distribution and unseen settings (Tables 9–10). For simulation benchmarks, we follow the standard evaluation protocols of LIBERO, CALVIN, and RoboTwin2.0, which report success rates over fixed evaluation episodes with deterministic seeds.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports GPU memory usage at inference time (Section 5.3, Figure 4) and inference latency comparisons (Figure 3). Training is conducted on NVIDIA A100/H100 GPUs. Detailed per-experiment training time is not reported but will be included in the camera-ready version.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. The work focuses on improving robotic manipulation through better action selection and does not involve human subjects, deceptive practices, or dual-use concerns. All benchmarks used are publicly available and properly cited.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Appendix J briefly discusses societal impacts. The work improves reliability of robotic manipulation policies, which has positive implications for safe deployment of robots in human environments. The method does not introduce new capabilities that pose direct societal risks beyond those inherent to general robotic manipulation research.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release pre-trained models or datasets that pose a high risk for misuse. The method is a verification framework for robotic manipulation policies and does not generate content that could be misused (e.g., deepfakes or harmful text).

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All benchmarks (LIBERO, CALVIN, RoboTwin2.0, SimplerEnv), backbone models (Qwen2.5-VL, InternVL3.5, $\pi_{0.5}$), and baseline methods (RoboMonkey, MG-Select, RoVer, AsyncVLA, V-GPS) are properly cited with their original publications. The simulation benchmarks are publicly available under their respective open-source licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The source code for the TokenPB framework has been submitted as supplementary material via the submission portal, with documentation for reproducing the main experimental results. The algorithmic procedure is also fully described in the paper (Algorithms 1–2 in Appendix E.3).

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects. All experiments are conducted in simulation environments or with physical robots operated by the research team.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects research. All real-world experiments involve robotic manipulation of inanimate objects (blocks, cups, plates, etc.) and do not pose risks to human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method does not use LLMs as a component. While the VLM backbone (Qwen2.5-VL, InternVL3.5) contains a language model component, it is used as a standard vision-language encoder for robotic policy learning—not as an LLM for text generation, reasoning, or any non-standard purpose. The VLM is fine-tuned end-to-end as part of the proposed co-training framework.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.