QCACHE-VLA: QUANTIZATION AND DIFFUSION CACHE FOR EFFICIENT VISION-LANGUAGE-ACTION POLICY INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models have emerged as a promising paradigm for generalist robot control. But their deployment on resource-constrained robotic platforms remains challenging due to the heavy Vision-Language Model (VLM) backbone and multi-step diffusion-based action generation. Existing acceleration methods are not directly suited to VLA diffusion policies: post-training quantization can suffer from image-text distribution mismatch and action-agnostic quantization decisions, while fixed-interval diffusion caching cannot adapt to non-uniform redundancy across denoising steps. To address these issues, we propose **QCache-VLA**, a discrepancy-aware acceleration framework for efficient VLA inference. For the VLM backbone, we introduce action-aware post-training quantization with modality-aligned smoothing candidate generation, action-aware candidate selection, and modality-balanced weight reconstruction. For the action expert, we propose a residual-calibrated dynamic diffusion cache that reuses deep residuals only when adjacent denoising steps are sufficiently similar. We further integrate INT8 Triton kernels and diffusion-step precomputation to enable practical end-to-end acceleration. Experiments in simulation and real-world tasks show that QCache-VLA preserves task performance while reducing GPU memory usage by 37% and achieving a $4\times$ end-to-end speedup.

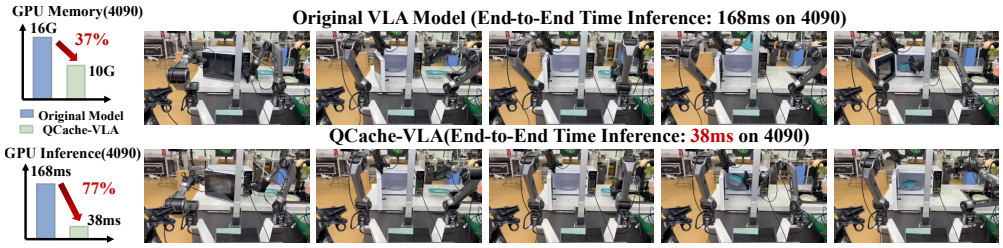


Figure 1: QCache-VLA preserves action quality while reducing end-to-end latency.

1 INTRODUCTION

Vision-Language-Action (VLA) models have recently become an important paradigm in embodied intelligence and robotics Kim et al. (2024); Black et al. (2024); Black et al.; Shukor et al. (2025). Representative models such as π_0 Black et al. (2024) and $\pi_{0.5}$ Black et al. typically consist of a large Vision-Language Model (VLM) backbone for visual-language understanding and an action expert for multi-step diffusion-based action generation. Although these models have shown strong capabilities in complex manipulation and cross-scenario generalization Black et al., their high memory consumption, computational cost, and inference latency make them difficult to deploy on resource-constrained robotic platforms Yang et al. (2026); Xu et al. (2026).

As illustrated in Fig. 2(d), the deployment bottleneck of VLA models such as $\pi_0/\pi_{0.5}$ mainly comes from two components. First, the VLM backbone accounts for a large portion of model parameters, leading to substantial memory and computation overhead. Second, the action expert requires multiple denoising steps to produce the final action, which further increases end-to-end inference latency.

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

Therefore, efficient VLA deployment requires reducing both the cost of the VLM backbone and the redundancy in diffusion-based action generation.

Post-training quantization (PTQ) provides a training-free way to compress the VLM backbone Xiao et al. (2023); Frantar et al. (2022); Lin et al. (2024); Shao et al. (2024); Liu et al. (2025c); Zheng et al. (2026). However, existing LLM/VLM quantization methods are not specifically designed for VLA diffusion policies Xu et al. (2026). In $\pi_0/\pi_{0.5}$, image and text tokens exhibit different activation distributions. Directly applying unified calibration statistics in existing methods can bias quantization parameters toward the modality with larger numerical ranges, amplifying cross-modal quantization errors Li et al. (2025). Since image observation and language instructions directly condition downstream action generation, such modality-biased errors can degrade final control performance.

In parallel, diffusion caching can reduce redundant computation in the action expert Ma et al. (2024); Wimbauer et al. (2024); Yang et al. (2026). However, existing cache methods usually rely on fixed intervals or multiple actually computed steps to determine feature reuse Yang et al. (2026); Bu et al.. Such strategies are less suitable for VLA action diffusion, where redundancy across denoising steps is highly non-uniform, as shown in Fig. 2(c). Fixed-interval caching may reuse stale features during rapidly changing steps or miss reuse opportunities during stable steps Bu et al.; Lyu et al. (2025). This motivates a dynamic cache mechanism that can determine reuse opportunities according to the current diffusion state.

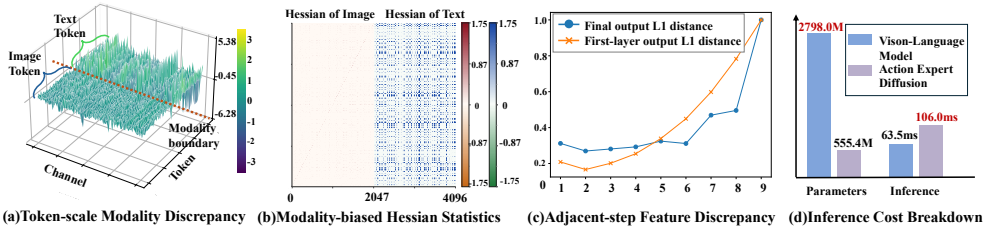
To address these challenges, we propose **QCache-VLA**, a training-free framework for efficient VLA deployment. QCache-VLA jointly addresses modality discrepancy in the VLM backbone and temporal discrepancy in the diffusion action expert. Specifically, it introduces action-aware PTQ that builds on modality balancing to reduce image-text quantization bias while prioritizing action-sensitive features, residual-calibrated dynamic caching to skip redundant diffusion computation, and INT8 Triton kernels Tillet et al. (2019) with diffusion-step precomputation for practical end-to-end acceleration. Experiments in simulation and real-world tasks show that QCache-VLA preserves task performance while significantly reducing memory usage and inference latency. Our main contributions are summarized as follows:

- We propose **QCache-VLA**, a training-free framework for efficient VLA deployment by jointly accelerating the VLM backbone and diffusion action expert.
- We introduce an action-aware PTQ method that incorporates action-sensitivity-weighted reconstruction loss into modality-aligned smoothing, complemented by modality-balanced weight reconstruction.
- We design a residual-calibrated dynamic diffusion cache that adaptively skips redundant deep computation across denoising steps.
- We build a deployment-oriented runtime backend with INT8 Triton kernels and diffusion-step precomputation, achieving up to $4\times$ end-to-end inference acceleration.

2 Related Work

2.1 Vision-Language-Action (VLA) Models.

Vision-Language-Action (VLA) models integrate visual perception, language understanding, and action generation into end-to-end robotic policies Kim et al. (2024); Black et al. (2024); Black et al.. Early policies such as ACT Zhao et al. (2023) and Diffusion Policy Chi et al. (2025) are mainly trained on task-specific data Zitkovich et al. (2023); Liu et al. (2025b), while recent VLA models leverage large-scale VLMs to improve cross-task generalization. OpenVLA Kim et al. (2024) enables language-conditioned action generation with pretrained VLMs, and π_0 Black et al. (2024) and $\pi_{0.5}$ Black et al. further generate continuous actions via flow matching Lipman et al. (2022) based on VLM backbones such as PaliGemma Beyer et al. (2024). Despite their strong performance, efficient VLA deployment remains challenging Wen et al. (2025); Zhang et al. (2026). SmolVLA reduces deployment cost with a compact architecture Shukor et al. (2025), while EfficientVLA exploits system-level redundancy for training-free acceleration Yang et al. (2026). Different from these works, we focus on two discrepancies in $\pi_0/\pi_{0.5}$-style models: the modality discrepancy between image and text tokens, and the temporal discrepancy across denoising steps. Based on this, we



(a)Token-scale Modality Discrepancy (b)Modality-biased Hessian Statistics (c)Adjacent-step Feature Discrepancy (d)Inference Cost Breakdown

Figure 2: Motivation of discrepancy-aware VLA acceleration.

combine modality-aware quantization, dynamic diffusion caching, and INT8 kernels for end-to-end efficient VLA inference.

2.2 Post-training Quantization

PTQ has been widely used for efficient LLM deployment without retraining Xiao et al. (2023); Frantar et al. (2022); Lin et al. (2024); Shao et al. (2024). Representative methods such as SmoothQuant Xiao et al. (2023) mitigate activation outliers through channel-wise equivalent scaling, while GPTQ Frantar et al. (2022) uses approximate second-order information to compensate for weight quantization errors. However, these methods are mainly designed for text models and assume relatively homogeneous calibration token statistics Xu et al. (2026). Recent multimodal quantization methods further reveal that image and text tokens differ in activation range, outlier distribution, and task sensitivity Li et al. (2025). Directly applying LLM-oriented quantization to multimodal models can therefore cause imbalanced cross-modal errors. For VLA models, such errors not only affect multimodal representations but also propagate to the action generation module and degrade control performance Xu et al. (2026). This motivates VLA-specific quantization that jointly preserves multimodal representations and action prediction accuracy.

2.3 Diffusion Cache

To reduce redundant computation in multi-step diffusion processes, existing methods cache and reuse intermediate features across timesteps Ma et al. (2024); Wimbauer et al. (2024); Liu et al. (2025a); Yang et al. (2026). In image and video generation, methods such as Block Caching Wimbauer et al. (2024) and DiCache Bu et al. exploit feature redundancy or multi-step differences to accelerate inference. However, these methods are mainly designed for visual generation, where the goal is to preserve image or video quality Bu et al.. Action diffusion in VLA models is more sensitive, as it directly produces continuous actions for robot execution. EfficientVLA Yang et al. (2026) explores action diffusion caching in VLA models, but its fixed-interval strategy struggles to adapt to dynamic variations across denoising steps. FasterCache Lyu et al. (2025) also shows that directly reusing adjacent-step features may lose subtle variations and degrade generation quality. Different from prior methods, our approach dynamically determines feature reuse based on the early action residual and reduces cache-induced errors through residual calibration, making it more suitable for low-step and real-time VLA action generation.

3 Method

3.1 Action-aware Post-training Quantization

Existing PTQ methods for VLMs commonly follow a smoothing-then-reconstruction paradigm Li et al. (2025).

Challenge A: Modality-biased smoothing and action-agnostic candidate selection. For input activations $\mathbf{X}$ and weights $\mathbf{W}$, channel-wise smoothing redistributes the quantization difficulty between activations and weights through the following equivalent transformation:

$$\mathbf{XW} = \left(\mathbf{XS}^{-1}\right)\left(\mathbf{SW}\right), \qquad \mathbf{S} = \mathrm{diag}(s_1, \ldots, s_C). \tag{1}$$

In VLA models, visual and language tokens exhibit substantially different activation-magnitude distributions. Directly aggregating all tokens to estimate the smoothing scales may therefore bias

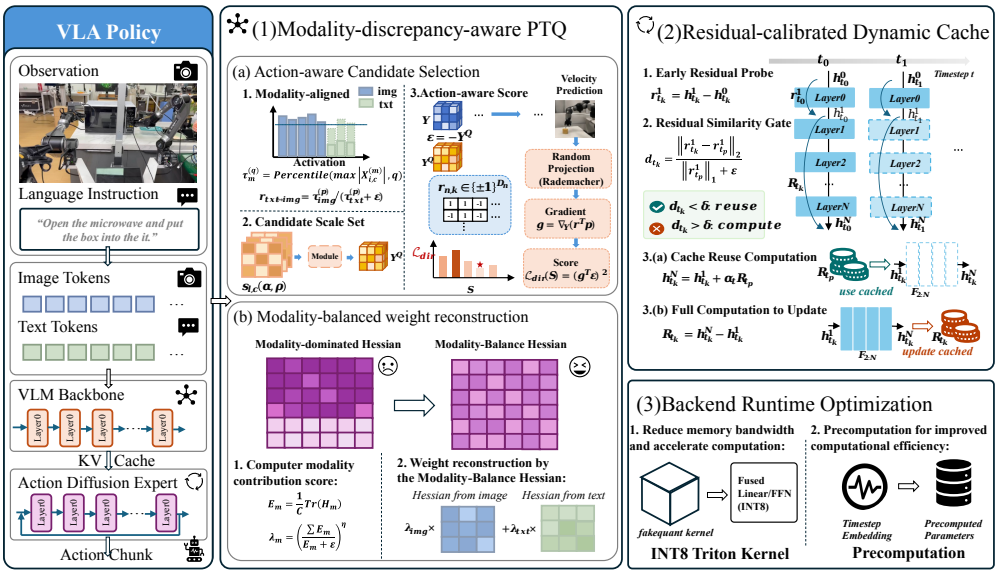


Figure 3: Overview of the QCache-VLA framework, which consists of three components.

the result toward the higher-magnitude modality. Meanwhile, smoothing candidates with similar local quantization errors may have substantially different effects on downstream velocity prediction. We thus decompose smoothing optimization into two stages: modality-aligned candidate generation and action-aware candidate selection.

Token-statistics-based modality-aligned candidate generation. Let the calibration activation of the condition stream at layer ℓ be $\mathbf{X}_\ell = \left[\mathbf{X}_\ell^{\text{img}}; \mathbf{X}_\ell^{\text{txt}}\right]$. For each modality $m \in \{\text{img}, \text{txt}\}$, we first compute the maximum channel magnitude of each token. To reduce the influence of extreme outliers, we use the q-th percentile to robustly estimate the overall activation scale of each modality:

$$v_{\ell,i}^{(m)} = \max_c \left|X_{\ell,i,c}^{(m)}\right|, \qquad \tau_{\ell,m}^{(q)} = \text{Percentile}\left(\left\{v_{\ell,i}^{(m)}\right\}_{i=1}^{N_m}, q\right). \tag{2}$$

For each candidate ratio $\rho \in \mathcal{R}$, with $\rho_{\ell,q} = \tau_{\ell,\text{img}}^{(q)}/(\tau_{\ell,\text{txt}}^{(q)} + \varepsilon)$, we virtually form $\widetilde{\mathbf{X}}_\ell(\rho) = [\mathbf{X}_\ell^{\text{img}}/\rho; \mathbf{X}_\ell^{\text{txt}}]$ only for smoothing-scale estimation, without modifying the actual model inputs during inference, while keeping the language activations unchanged. This alignment is used only to estimate the smoothing scales and does not modify the actual model inputs during inference. Based on the aligned activations, we compute the channel-wise statistics and construct the candidate scales as

$$a_{\ell,c}(\rho) = \text{Mean}_{b,n}\left|\widetilde{X}_{\ell,b,n,c}(\rho)\right|, \qquad s_{\ell,c}(\alpha,\rho) = a_{\ell,c}(\rho)^\alpha. \tag{3}$$

By varying $\alpha \in \mathcal{A}$ and $\rho \in \mathcal{R}$, we obtain the modality-aligned candidate set $\mathcal{S}_{\text{MAS},\ell}$, where $\mathcal{R}$ additionally includes the unaligned candidate $\rho = 1$.

Action-aware candidate selection. For smoothing group ℓ and calibration sample n, let $\mathbf{Y}_{\ell,n}$ denote the floating-point output of the complete attention or MLP branch being searched. Given a smoothing candidate $\mathbf{S}$, let $\mathbf{Y}_{\ell,n}^Q(\mathbf{S})$ denote its corresponding fake-quantized output. The resulting local quantization perturbation is

$$\boldsymbol{\epsilon}_{\ell,n}(\mathbf{S}) = \mathbf{Y}_{\ell,n}^Q(\mathbf{S}) - \mathbf{Y}_{\ell,n}. \tag{4}$$

Let $\mathbf{p}_n = F_{\ell,n}(\mathbf{Y}_{\ell,n})$ denote the floating-point velocity prediction, where $F_{\ell,n}$ represents the remaining network after the inspected module. Ideally, the selected candidate should minimize the resulting perturbation in the downstream velocity prediction:

$$\mathcal{L}_{\ell,\text{true}}(\mathbf{S}) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{D_n}\left\|F_{\ell,n}\left(\mathbf{Y}_{\ell,n} + \boldsymbol{\epsilon}_{\ell,n}(\mathbf{S})\right) - F_{\ell,n}(\mathbf{Y}_{\ell,n})\right\|_2^2, \tag{5}$$

where D_n denotes the number of valid elements in the velocity prediction, excluding padded dimensions and padded timesteps. By linearizing the downstream mapping around the floating-point module output, we obtain

$$\mathcal{L}_{\ell,\text{true}}(\mathbf{S}) \approx \frac{1}{N}\sum_{n=1}^{N}\frac{1}{D_n}\left\|\mathbf{J}_{\ell,n}\boldsymbol{\epsilon}_{\ell,n}(\mathbf{S})\right\|_2^2, \qquad \mathbf{J}_{\ell,n} = \frac{\partial \mathbf{p}_n}{\partial \mathbf{Y}_{\ell,n}}. \tag{6}$$

Explicitly constructing the full Jacobian is computationally and memory intensive. To avoid this cost, we sample K Rademacher vectors satisfying

$$\mathbb{E}\left[\mathbf{r}_{n,k}\mathbf{r}_{n,k}^{\mathsf{T}}\right] = \mathbf{I}, \tag{7}$$

and compute the corresponding randomly projected gradients:

$$\mathbf{g}_{\ell,n,k} = \nabla_{\mathbf{Y}_{\ell,n}}\frac{\mathbf{r}_{n,k}^{\mathsf{T}}\mathbf{p}_n}{\sqrt{D_n}} = \frac{\mathbf{J}_{\ell,n}^{\mathsf{T}}\mathbf{r}_{n,k}}{\sqrt{D_n}}. \tag{8}$$

The same directional gradients are shared by all smoothing candidates. We then evaluate each candidate using the following action-aware score:

$$\mathcal{L}_{\ell,\text{dir}}(\mathbf{S}) = \frac{1}{NK}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(\mathbf{g}_{\ell,n,k}^{\mathsf{T}}\boldsymbol{\epsilon}_{\ell,n}(\mathbf{S})\right)^2. \tag{9}$$

According to the properties of Rademacher vectors, this score satisfies

$$\mathbb{E}_{\mathbf{r}}\left[\left(\mathbf{g}_{\ell,n,k}^{\mathsf{T}}\boldsymbol{\epsilon}_{\ell,n}\right)^2\right] = \frac{1}{D_n}\left\|\mathbf{J}_{\ell,n}\boldsymbol{\epsilon}_{\ell,n}\right\|_2^2. \tag{10}$$

It therefore provides a matrix-free estimate of the first-order velocity prediction error induced by quantization. For a two-stream module, the inner products of the condition and action-expert streams are summed before squaring, thereby preserving the error interactions between the two streams.

Finally, the optimal scale for smoothing group ℓ is selected from the modality-aligned candidate set:

$$\boxed{\mathbf{S}_\ell^* = \underset{\mathbf{S}\in\mathcal{S}_{\text{MAS},\ell}}{\arg\min}\,\mathcal{L}_{\ell,\text{dir}}(\mathbf{S})}. \tag{11}$$

Challenge B. After smoothing, weight reconstruction is adopted to further reduce quantization errors by using approximate second-order information. Given the current quantized column c and the remaining columns f, let $\mathcal{F} = \{c\} \cup f$ denote the current set of unquantized columns before quantizing c. The reconstruction update can be written as

$$\Delta\mathbf{W}_f = -(\mathbf{W}_c - Q(\mathbf{W}_c))\left([\mathbf{H}_{\mathcal{F}}^{-1}]_{cc}\right)^{-1}[\mathbf{H}_{\mathcal{F}}^{-1}]_{cf}, \quad \mathbf{H} = \mathbf{X}^T\mathbf{X}. \tag{12}$$

For VLA models, the calibration activations contain tokens from different modalities: $\mathbf{H} = (\mathbf{X}^{img})^T\mathbf{X}^{img} + (\mathbf{X}^{txt})^T\mathbf{X}^{txt}$.

Modality-balanced weight reconstruction. To prevent the Hessian from being dominated by a single modality, we first compute the token-count-normalized Hessian contribution of each modality and use its trace to measure the modality contribution score. Specifically,

$$\mathbf{H}_m = \frac{1}{N_m}(\mathbf{X}^m)^T\mathbf{X}^m, \qquad E_m = \frac{1}{C}\,\text{Tr}(\mathbf{H}_m), \quad m \in \{img, txt\}, \tag{13}$$

where N_m is the number of token positions belonging to modality m and C is the hidden dimension.

Based on the modality contribution scores, we construct a balancing coefficient for each modality:

$$\lambda_m = \left(\frac{\sum_{j\in\mathcal{M}} E_j + \epsilon}{E_m + \epsilon}\right)^{\eta}, \qquad \mathcal{M} = \{img, txt\}, \tag{14}$$

where η controls the strength of modality balancing. When $\eta > 0$, modalities with larger Hessian contribution scores are relatively suppressed, while weaker modalities are enhanced. When $\eta = 0$, energy-based reweighting is disabled, while per-modality normalization is retained. Standard weight reconstruction Frantar et al. (2022) is recovered by disabling modality balancing.

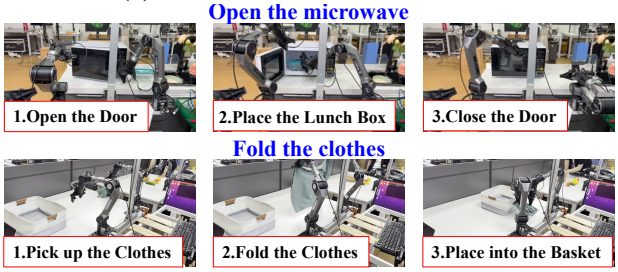
The final modality-balanced Hessian is formulated as: $\mathbf{H}_{bal} = \sum \lambda_m \mathbf{H}_m$. By substituting $\mathbf{H}_{bal}$ into Eq. 12, weight reconstruction is performed using balanced multimodal second-order statistics, enabling the quantized weights to better preserve both visual representations and language-conditioned information.

Table 1: Simulation results on RoboTwin.

Method	Quant.	Steps	Easy				Hard			
			beat_block hammer	stack_bowl three	adjust bottle	Avg.	beat_block hammer	stack_bowl three	adjust bottle	Avg.
$\pi_{0.5}$										
FP16	–	10	64	76	98	79.3	14	57	90	53.7
SmoothQuant	W8A8	10	58	75	97	76.7	18	54	89	53.7
MBQ	W8A8	10	58	76	97	77.0	18	50	90	52.7
QCache-VLA	W8A8	10	**64**	**79**	**98**	**80.3**	**20**	56	**90**	**55.3**
EfficientVLA*	W8A8	5	52	60	76	62.7	15	39	64	39.3
DiCache*	W8A8	3	63	72	95	76.7	15	53	84	50.7
QCache-VLA	W8A8	**2**	**63**	**75**	**96**	**78.0**	**15**	**54**	**85**	**51.3**
π_0										
FP16	–	10	30	52	94	58.7	11	39	72	40.7
SmoothQuant	W8A8	10	29	57	91	59.0	10	34	69	37.7
MBQ	W8A8	10	30	58	91	59.7	11	33	69	37.7
QCache-VLA	W8A8	10	**30**	**63**	**92**	**61.7**	**11**	**33**	**69**	**37.7**
EfficientVLA*	W8A8	5	31	58	92	60.3	8	33	68	36.3
DiCache*	W8A8	3	22	58	88	56.0	8	30	72	36.7
QCache-VLA	W8A8	**2**	**30**	**65**	**93**	**62.7**	**10**	**32**	**68**	**36.7**

*EfficientVLA and DiCache are combined with MBQ for a fair comparison under the same W8A8 quantization setting.

(a) Real-world task visualization



(b) Deployment efficiency

Method	Memory (MB)	Time (ms)
$\pi_{0.5}$	16064	168 (1.00×)
+ Quant	18478 (+15.0%)	174 (0.97×)
+ Cache	18478 (+15.0%)	102 (1.65×)
+ Triton	**10100** (−37.1%)	**38** (**4.42**×)

Figure 4: Real-world deployment and runtime analysis.

3.2 Residual-calibrated Dynamic Diffusion Cache

Challenge C. Although the action expert performs similar computations across denoising steps, the redundancy is highly non-uniform over time. As shown in Fig. 2(c), the normalized ℓ_1 distance between adjacent step outputs often follows a U-shaped pattern, where middle denoising steps are more stable and thus more suitable for cache reuse. Therefore, fixed-interval caching may reuse stale features during rapidly changing steps or miss reuse opportunities during stable steps.

Residual-calibrated dynamic cache. To address this issue, we use the early action residual as a lightweight indicator of step-wise redundancy. Specifically, let $\mathbf{h}_t^0$ denote the input hidden state of the action expert at the t-th denoising step. After the first expert layer, $\mathbf{h}_t^1 = F_1(\mathbf{h}_t^0)$. We define the early action residual as $\mathbf{r}_t^1 = \mathbf{h}_t^1 - \mathbf{h}_t^0$.

This residual captures the local transformation induced by the current diffusion state and serves as a lightweight proxy for estimating whether deeper layers can be reused. We measure the discrepancy between adjacent denoising steps by

$$d_t = \frac{\left\|\mathbf{r}_t^1 - \mathbf{r}_{t-1}^1\right\|_1}{\left\|\mathbf{r}_{t-1}^1\right\|_1 + \epsilon}. \tag{15}$$

If $d_t < \delta$, the current step is considered close to the previous step, and the deep computation can be approximated using cached residuals. When cache is not used, we compute all remaining layers and store the deep residual: $\mathbf{R}_t = \mathbf{h}_t^L - \mathbf{h}_t^1$.

Table 2: Results on LIBERO benchmark.

Method	Quant.	Diffusion Steps	Spatial	Object	Goal	Long	Avg.
			$\pi_{0.5}$				
FP Model	–	10	0.994	0.988	0.978	0.922	0.971
MBQ	W8A8	10	0.980	0.968	0.972	0.920	0.960(-1.1%)
QCache-VLA	W8A8	10	**0.992**	**0.984**	0.980	**0.924**	**0.970(-0.1%)**
EfficientVLA*	W8A8	2	0.950	0.960	0.924	0.846	0.920(-5.1%)
QCache-VLA	W8A8	2	0.986	0.982	**0.984**	0.910	0.966(-0.5%)
			OpenVLA-oft				
FP Model	–	-	0.972	0.974	0.958	0.942	0.962
SmoothQuant	W8A8	-	0.932	0.982	0.972	0.924	0.952(-1.0%)
MBQ	W8A8	-	0.924	0.978	0.966	0.936	0.951(-1.1%)
QCache-VLA	W8A8	-	**0.978**	**0.980**	**0.968**	**0.944**	**0.968(+0.6%)**

*EfficientVLA is combined with MBQ for a fair comparison under the same W8A8 quantization setting.

(a) Real-world task success rate

Method	Steps	Open the Microwave	Fold the Clothes
	$\pi_{0.5}$		
FP Model	10	9/10	10/10
MBQ	10	7/10	10/10
QCache-VLA	10	8/10	10/10
EfficientVLA	2	2/10	0/10
QCache-VLA	**2**	**8**/10	**10**/10

(b) Quantization error

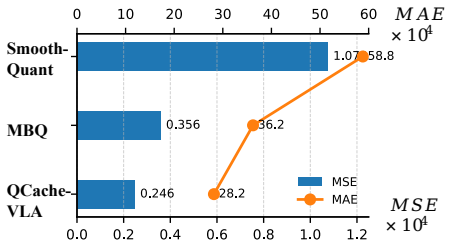


Figure 5: Real-world deployment and quantization error analysis.

When cache is enabled, we skip the remaining layers and reuse the cached residual from the previous step. To account for magnitude variations across denoising steps, we introduce a calibration factor:

$$\alpha_t = \frac{\left\|\mathbf{r}_t^1\right\|_2}{\left\|\mathbf{r}_{t-1}^1\right\|_2 + \epsilon}. \tag{16}$$

The final output of the action expert is then approximated as

$$\widehat{\mathbf{h}}_t^L = \mathbf{h}_t^1 + \alpha_t \mathbf{R}_{t-1}. \tag{17}$$

Compared with directly reusing the previous hidden state, this residual-based strategy preserves the current first-layer response $\mathbf{h}_t^1$, while α_t adapts the cached residual to the current diffusion state.

3.3 INT8 Triton Kernels and Diffusion Precomputation

PTQ and dynamic caching reduce numerical precision and redundant diffusion computation, but their practical speedup can still be limited by runtime overhead. To bridge this gap, we build a deployment-oriented backend with INT8 Triton kernels and diffusion-step precomputation. First, we replace the dominant PyTorch linear projections with specialized INT8 Triton kernels. Quantized weights and scaling factors are packed offline, while the runtime kernels perform INT8 matrix multiplication with fused dequantization and output accumulation. This avoids unnecessary dequantized weight materialization and reduces memory. Second, we exploit the fixed diffusion schedule during inference. For timestep-conditioned modules in the action expert, we precompute timestep embeddings and the corresponding modulation parameters before deployment. During inference, each denoising step directly loads the precomputed parameters, avoiding repeated time-embedding and modulation computations inside the diffusion loop. Together, INT8 Triton execution accelerates dense linear computation, while diffusion precomputation removes redundant timestep-conditioned operations, enabling practical end-to-end acceleration for VLA inference.

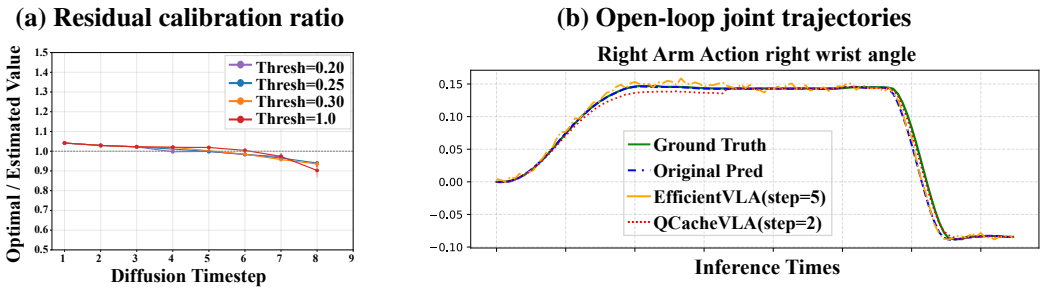


Figure 6: Analysis of the residual-calibrated diffusion cache and open-loop trajectories comparison. (a) Ratio between the optimal α and the estimated α across diffusion timesteps. (b) Open-loop joint trajectories comparing ground truth, original prediction, EfficientVLA, and QCache-VLA.

4 Experiment

4.1 Experimental Settings

Model and implementation. We evaluate QCache-VLA on RoboTwin Chen et al. (2025), LIBERO Liu et al. (2023), and real-world robotic tasks using π_0 Black et al. (2024), $\pi_{0.5}$ Black et al., and OpenVLA-OFT Kim et al. (2025) as backbone policies. For RoboTwin, we select three representative tasks: *adjust bottle*, *beat block hammer*, and *stack bowls three*, and report success rates under both easy and hard settings. For LIBERO, we evaluate on four suites: *Spatial*, *Object*, *Goal*, and *Long*, each comprising 10 tasks with 50 evaluation trials per task. For real-world deployment, we conduct 10 trials per task: 10 repeated microwave trials and five clothes-folding trials for each of two garment colors, as shown in Fig. 4(a). These physical experiments assess repeatability and appearance variation, complementing our broader simulation evaluations.

Baselines and details. For quantization, we compare with SmoothQuant Xiao et al. (2023) and MBQ Li et al. (2025). For diffusion cache acceleration, we compare with EfficientVLA Yang et al. (2026) and DiCache Bu et al.. Quantization is performed on a single NVIDIA A800 GPU, while simulation and real-world evaluations are conducted on a single NVIDIA RTX 4090 GPU. For quantization, we use per-channel weight quantization by default and group-wise quantization (group size 128) for modality-balanced reconstruction, with per-token activation quantization. The hyperparameter η in Eq. 14 is set to 0.5. For diffusion caching, we set the distance threshold δ in Eq. 15 to 1.0. Under this aggressive setting, only the first and final denoising steps are fully computed. More implementation details are provided in the Appendix.

4.2 Results on Simulation Environment

Comparison on RoboTwin. Table 1 compares QCache-VLA with two quantization baselines, as well as two diffusion cache baselines. QCache-VLA achieves consistently strong success rates on both π_0 and $\pi_{0.5}$. Compared with existing quantization methods, our strategy improves the success rate by up to 3.6%. Notably, EfficientVLA and DiCache require 5 and 3 denoising steps to maintain stable performance, while their 2-step variants suffer from clear degradation. QCache-VLA, however, achieves comparable or better success rates with only 2 steps, demonstrating stronger robustness under aggressive diffusion acceleration. More details are provided in the Appendix.

Comparison on LIBERO. Table 2 reports the performance of QCache-VLA on $\pi_{0.5}$ and OpenVLA-OFT. Under W8A8 quantization, QCache-VLA outperform MBQ by 1.0% while closely matching the full-precision model. On OpenVLA-OFT, QCache-VLA also achieves a higher average success rate than both MBQ and the full-precision model. Under the more aggressive 2-step inference setting, QCache-VLA improves the success rate by up to 4.6% over EfficientVLA. These results demonstrate that QCache-VLA effectively preserves task performance while substantially reducing full expert computation.

4.3 Results on Real-world Robotic Deployment

Comparison on Real-world tasks. We evaluate QCache-VLA on two real-world manipulation tasks using an AgileX ALOHA bimanual robot, as shown in Fig. 5(a). Over 10 trials for each task,

Table 3: Ablation study and threshold sensitivity on $\pi_{0.5}$.

(a) Component ablation on LIBERO

Method	Modality-aligned Smoothing	Modality-balanced Reconstruction	Residual Dynamic Cache	Acc.
FP Model	–	–	–	0.971
+ MAS	✓	–	–	0.964
+ MBR	–	✓	–	0.966
+ RDC	–	–	✓	0.972
+ MAS + MBR	✓	✓	–	0.970
QCache-VLA	✓	✓	✓	**0.966**

(b) Threshold sensitivity on RoboTwin

Threshold	Avg. Steps	Acc.
–	10.0	64
0.20	5.4	**65**
0.25	4.3	63
0.30	4.0	63
1.00	**2.0**	63

QCache-VLA maintains high success rates on both tasks. QCache-VLA achieves stable execution under the aggressive 2-step setting and successfully completes both tasks. In contrast, the fixed-interval caching strategy of EfficientVLA struggles to handle the dynamic variations in complex manipulation tasks. By contrast, EfficientVLA suffers from unstable actions: in the microwave task, arm jitter leads to inaccurate button pressing and unreliable grasping of the lunch box; in the clothes-folding task, unstable motion and gripper commands prevent the robot from smoothly spreading and folding the clothes within the 3-minute limit. This highlights the importance of adaptive cache reuse for real-world long-horizon manipulation. These results further demonstrate the plug-and-play deployability of QCache-VLA.

Analysis on end-to-end deployment. As shown in Fig. 4(b), we further evaluate the end-to-end memory footprint and inference latency. Fake-quantized inference slightly increases memory usage and latency due to the lack of hardware-friendly execution. Diffusion caching reduces latency by skipping redundant denoising computation, while the runtime backend further converts quantization into practical speedup. Overall, QCache-VLA achieves up to 4.4$\times$ acceleration and reduces memory usage by approximately 37%, demonstrating the necessity of combining algorithmic compression with hardware-efficient execution.

4.4 Analysis and Ablation Study

Quantization Analysis. As shown in Fig. 5(b), we compare the output errors of SmoothQuant, MBQ, and QCache-VLA after quantization using MSE and MAE. QCache-VLA achieves substantially lower errors on both metrics, demonstrating that the proposed modality-discrepancy-aware quantization better preserves the original model outputs.

Diffusion Cache Analysis. Fig. 6(a) shows that the ratio between the oracle and estimated calibration factors stays close to 1 across denoising timesteps, validating the effectiveness of our early-residual-based calibration. Fig. 6(b) further shows that QCache-VLA produces open-loop joint trajectories closer to the ground truth and original prediction than EfficientVLA, indicating better preservation of action dynamics under aggressive diffusion caching.

Component Ablation. Table 3(a) presents the component ablation results on LIBERO. Applying modality-aligned smoothing or modality-balanced reconstruction individually introduces modest performance degradation, while combining them recovers near full-precision performance. With residual dynamic caching further integrated, QCache-VLA maintains comparable task success rates while substantially reducing diffusion computation.

5 Conclusion

We present QCache-VLA, a training-free framework that addresses modality discrepancy in the VLM backbone and temporal discrepancy in the diffusion action expert. QCache-VLA combines modality-aware quantization, residual-calibrated dynamic caching, and INT8 runtime optimization for efficient VLA deployment. Experiments in simulation and real-world tasks show that QCache-VLA reduces memory usage and inference latency while preserving task success rates, demonstrating the effectiveness of jointly optimizing quantization, diffusion reuse, and hardware execution.

AI USE STATEMENT

In this work, generative AI tools were used to assist with Chinese–English translation, language editing, and identifying and correcting unclear or inconsistent descriptions in the manuscript. They were also used for limited code review and debugging, including identifying potential implementation errors and suggesting corrections. Generative AI tools were not used to formulate the research hypotheses, develop the proposed methodology, design the experiments, generate or modify experimental data, or analyze and interpret the experimental results. Tasks involving theoretical claims or proofs and qualitative data analysis were not applicable to this work. All AI-assisted text was manually reviewed and revised by the authors, and all AI-assisted code corrections were inspected and tested through the experiments reported in this paper. The authors take full responsibility for the final content of this work, including its text, claims, results, and software artifacts.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide detailed experimental settings in the appendix, including dataset configurations, the complete algorithmic procedure, and the specific parameter settings used in our experiments. We will publicly release the code and relevant configuration files upon acceptance. We believe these details are sufficient to reproduce the main results reported in this paper.

REFERENCES

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. $\pi_0.5$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. In *The Fourteenth International Conference on Learning Representations*.

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.

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.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

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

Shiyao Li, Yingchun Hu, Xuefei Ning, Xihui Liu, Ke Hong, Xiaotao Jia, Xiuhong Li, Yaqi Yan, Pei Ran, Guohao Dai, et al. Mbq: Modality-balanced quantization for large vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 4167–4177, 2025.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

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

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.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 7353–7363, 2025a.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025b.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. In *International Conference on Learning Representations*, volume 2025, pp. 92009–92032, 2025c.

Zhengyao Lyu, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K Wong. Fastercache: Training-free video diffusion model acceleration with high quality. In *International Conference on Learning Representations*, volume 2025, pp. 33132–33156, 2025.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15762–15772, 2024.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Gao Peng, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *International Conference on Learning Representations*, volume 2024, pp. 45472–45496, 2024.

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.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6211–6220, 2024.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pp. 38087–38099. PMLR, 2023.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization. *arXiv preprint arXiv:2602.03782*, 2026.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2026.

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, Dan Wang, Yuan Du, and Shanghang Zhang. Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18764–18772, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Xingyu Zheng, Haotong Qin, Yuye Li, Haoran Chu, Jiakai Wang, Jinyang Guo, Michele Magno, and Xianglong Liu. First-order error matters: Accurate compensation for quantized large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 28883–28891, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183. PMLR, 2023.

A APPENDIX

This document supplements the main paper as follows:

- Sec. A.1 provides additional implementation details of QCache-VLA, including quantization, diffusion cache, and INT8 Triton kernel implementations.
- Sec. A.2 describes the experimental settings, including simulation benchmarks, real-world robot setup, baselines and evaluation protocols.
- Sec. A.3 reports additional results on RoboTwin and real-world robotic tasks.

A.1 METHOD DETAILS

A.1.1 QUANTIZATION IMPLEMENTATION DETAILS

For a fair comparison, all quantization methods are applied to the same set of linear layers in the policy network. Specifically, SmoothQuant Xiao et al. (2023), MBQ Li et al. (2025), and QCache-VLA all quantize the VLM and action expert attention projections, including query, key, value, and output projections, as well as the MLP projections, including gate, up, and down projections. Other modules are kept in their original precision.For QCache-VLA, the proposed modality-aligned smoothing, action-aware candidate selection, and modality-balanced reconstruction are applied only to the VLM backbone. For the action expert, we adopt conventional channel-wise smoothing followed by GPTQ-based weight reconstruction.

Table 4: Quantization configurations.

Method	Calibration Samples	Weight Bit	Weight Granularity	Activation Bit	Activation Granularity	Symmetric	Key Hyperparameters
SmoothQuant	128	8	Per-channel	8	Per-token	True	$\alpha = 0.5$
MBQ	128	8	Per-group	8	Per-token	True	`loss=mae`
QCache-VLA	128	8	Per-group	8	Per-token	True	$\mathcal{L}_{\mathrm{dir}}, \eta = 0.5$

Table 4 summarizes the quantization configurations. We use 128 calibration samples for all methods. For activations, all methods adopt 8-bit per-token symmetric quantization. For weights, SmoothQuant use 8-bit per-channel quantization, while MBQ and QCache-VLA uses 8-bit group-wise quantization with a group size of 128 during Hessian-based reconstruction Frantar et al. (2022). SmoothQuant uses a smoothing factor $\alpha = 0.5$. MBQ follows its modality reweighting setting. For QCache-VLA, we adopt the proposed action-aware directional loss (Eq. 9) for smoothing candidate selection and set the modality-balancing hyperparameter η to 0.5.

Our quantization pipeline consists of two stages. In the first stage, we perform modality-aligned smoothing with action-aware candidate selection. We first compute randomly projected gradients of the full-precision velocity prediction with respect to the target module output. These gradients are shared across all smoothing candidates. We then construct candidate scales using modality-aligned activation statistics and select the candidate that minimizes the estimated first-order perturbation in the downstream velocity prediction, as defined in Eq. 9. The detailed procedure is provided in Algorithm 1. In the second stage, we apply modality-balanced Hessian accumulation for weight reconstruction. Image and text tokens are separated when constructing the Hessian statistics, and their contributions are balanced using the coefficients defined in Eq. 14. The detailed procedure is provided in Algorithm 2.

A.1.2 DIFFUSION CACHE IMPLEMENTATION DETAILS

We compare QCache-VLA with two baselines, EfficientVLA Yang et al. (2026) and DiCache Bu et al..

For EfficientVLA, we follow the original fixed-interval caching strategy. The action expert is fully evaluated every N denoising steps, where the attention and MLP features are updated and stored. For the intermediate denoising steps, the cached attention/MLP features from the most recent computed step are reused.

Algorithm 1 QCVLA: Action-aware Modality-aligned Smoothing Search

Input: Calibration dataset $\mathcal{D}$, full-precision policy P, target module F with weights W, quantile set $\mathcal{Q}$, smoothing exponent set $\mathcal{A}$, projection count K
Output: Optimal channel scales $s^\star$
▷ Step 1: Precompute action-space directional gradients
for each sample $n \in \mathcal{D}$ **do**
  $p_n \leftarrow P(o_n, a_{\tau_n}, \tau_n)$ ▷ Full-precision velocity prediction
  Cache module input x_n and output y_n
  $D_n \leftarrow \max(\|M_n\|_1, 1)$ ▷ M_n: valid action mask
  for $k = 1, \ldots, K$ **do**
    $r_{n,k} \sim \text{Rademacher}$
    $z_{n,k} \leftarrow \dfrac{\langle r_{n,k}, M_n \odot p_n \rangle}{\sqrt{D_n}}$
    $g_{n,k} \leftarrow \nabla_{y_n} z_{n,k}$
  end for
end for
Fix $\{g_{n,k}\}$ and calibration states ▷ Shared across candidates
▷ Step 2: Generate modality-aligned smoothing candidates
for $m \in \{\text{img}, \text{txt}\}$ **do**
  $v_{m,n,t} \leftarrow \max_c |x_{m,n,t,c}|$
end for
$\mathcal{R} \leftarrow \{1\} \cup \left\{ \dfrac{\text{Quantile}(v_{\text{img}}, q)}{\text{Quantile}(v_{\text{txt}}, q) + \varepsilon} : q \in \mathcal{Q} \right\}$
▷ Step 3: Action-aware candidate selection
$\mathcal{E}_{\min} \leftarrow \infty$
for each $\rho \in \mathcal{R}$ **do**
  $\tilde{x} \leftarrow [x_{\text{img}}/\rho;\, x_{\text{txt}}]$ ▷ Virtual alignment
  $a_c(\rho) \leftarrow \text{Mean}_{n,t} |\tilde{x}_{n,t,c}|$
  for each $\alpha \in \mathcal{A}$ **do**
    $s_c \leftarrow \max(a_c(\rho)^\alpha, \varepsilon)$
    $S \leftarrow \text{diag}(s_1, \ldots, s_C)$
    $x'_n \leftarrow x_n S^{-1}$
    $W' \leftarrow \text{FakeQuant}_w(SW)$
    $y_n^Q(S) \leftarrow F_{W'}(\text{FakeQuant}_a(x'_n)), \quad \forall n \in \mathcal{D}$
    $\epsilon_n(S) \leftarrow y_n^Q(S) - y_n$
    $\mathcal{L}_{\text{dir}}(S) \leftarrow \dfrac{1}{|\mathcal{D}|K} \displaystyle\sum_{n \in \mathcal{D}} \sum_{k=1}^{K} \left(\langle g_{n,k}, \epsilon_n(S) \rangle \right)^2$
    if $\mathcal{L}_{\text{dir}}(S) < \mathcal{E}_{\min}$ **then**
      $\mathcal{E}_{\min} \leftarrow \mathcal{L}_{\text{dir}}(S)$
      $s^\star \leftarrow s$
    end if
  end for
end for
return $s^\star$

For DiCache, we adapt its shallow-probe-based caching mechanism Bu et al. to VLA action diffusion. To reduce runtime overhead, we use the output of the first expert layer as the online probe. At each denoising step, we compare the current first-layer expert output with that from the previous step and accumulate the normalized discrepancy as a cache error indicator. If the accumulated error is below the threshold and the current step is not the final denoising step, DiCache reuses cached features. Following its dynamic trajectory alignment strategy, we estimate an interpolation coefficient from the first-layer residual trajectories of recent denoising steps and use it to approximate the current cached output. We also maintain two-step action residuals to construct the cached action prediction in the VLA action space.

Algorithm 2 QCVLA: Modality-Balanced Hessian Accumulation

Input: Calibration batches $\mathcal{D}$, balancing strength η, small constant ε
Output: Modality-balanced Hessian $\mathbf{H}_{\mathrm{bal}}$
▷ Step 1: Accumulate global modality statistics
for $m \in \{\mathrm{img}, \mathrm{txt}\}$ **do**
  $\mathbf{G}_m \leftarrow \mathbf{0}, \quad N_m \leftarrow 0$
end for
for each batch $\mathbf{X} \in \mathcal{D}$ **do**
  Split $\mathbf{X}$ into $\mathbf{X}_{\mathrm{img}}$ and $\mathbf{X}_{\mathrm{txt}}$
  for $m \in \{\mathrm{img}, \mathrm{txt}\}$ **do**
    Flatten modality token positions: $\mathbf{X}_m \in \mathbb{R}^{N_{m,b} \times C}$
    $\mathbf{G}_m \leftarrow \mathbf{G}_m + \mathbf{X}_m^\top \mathbf{X}_m$
    $N_m \leftarrow N_m + N_{m,b}$
  end for
end for
▷ Step 2: Compute global modality weights
for $m \in \{\mathrm{img}, \mathrm{txt}\}$ **do**
  $\mathbf{H}_m \leftarrow \dfrac{\mathbf{G}_m}{N_m}$
  $E_m \leftarrow \dfrac{\mathrm{Tr}(\mathbf{H}_m)}{C}$
end for
$E_{\mathrm{sum}} \leftarrow E_{\mathrm{img}} + E_{\mathrm{txt}}$
for $m \in \{\mathrm{img}, \mathrm{txt}\}$ **do**
  $\lambda_m \leftarrow \left(\dfrac{E_{\mathrm{sum}} + \varepsilon}{E_m + \varepsilon} \right)^{\eta}$
end for
▷ Step 3: Construct the balanced Hessian
$\mathbf{H}_{\mathrm{bal}} \leftarrow \lambda_{\mathrm{img}} \mathbf{H}_{\mathrm{img}} + \lambda_{\mathrm{txt}} \mathbf{H}_{\mathrm{txt}}$

In contrast, QCache-VLA uses the early residual rather than the raw first-layer output as the cache indicator. The detailed procedures are provided in Algorithm 3. This design directly measures the local transformation induced by the current diffusion state. When the early residual discrepancy is below the threshold, QCache-VLA skips the remaining expert layers and reuses the cached deep residual. The reused residual is further calibrated by the ratio between the current and previous early residual magnitudes. This residual-based formulation preserves the current first-layer response while adapting the cached deep transformation to the current denoising state. We disable cache reuse at the final denoising step for all cache-based methods.

A.1.3 Triton kernels Implementation Details

We implement a deployment-oriented Triton backend to translate algorithmic compression into practical latency and memory savings. The backend converts the original PyTorch checkpoint into a compact weight dictionary used by the Triton inference path. For INT8 execution, we quantize the selected linear weights offline and store the quantized weights together with their scaling factors. The runtime backend pre-allocates intermediate buffers, including hidden states, attention buffers, RoPE buffers, and INT8 temporary buffers, to avoid repeated memory allocation during denoising. The timestep-dependent AdaRMS modulation is precomputed offline for all denoising steps. Specifically, we precompute the timestep embeddings and fold the induced scale, shift, and gate parameters into the corresponding attention, FFN, and action output projection weights. During inference, each denoising step directly loads the precomputed step-specific weights and biases, avoiding repeated timestep embedding and modulation computations inside the diffusion loop.

A.2 Experimental Details

A.2.1 Simulation Settings

We evaluate QCache-VLA on two simulation benchmarks: RoboTwin 2.0 and LIBERO.

For RoboTwin, we use task-specific π_0 and $\pi_{0.5}$ checkpoints fine-tuned from the original pretrained models on three representative bimanual manipulation tasks: *adjust_bottle*, *beat_block_hammer*, and *stack_bowls_three*. Following the RoboTwin 2.0 training protocol [1], each checkpoint is fine-tuned for 30,000 training steps on two NVIDIA A800 GPUs with a per-GPU batch size of 32. Each RoboTwin task is evaluated 100 episodes under both easy and hard modes, with random seeds initialized from 0 and increased sequentially.

For LIBERO, we evaluate $\pi_{0.5}$ and OpenVLA-OFT on four standard suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. For $\pi_{0.5}$, we follow the OpenPI LIBERO fine-tuning and evaluation workflow[2]. Due to hardware constraints, we train the PyTorch implementation of $\pi_{0.5}$ with a batch size of 32, while the official OpenPI setting uses a larger batch size of 256. This may lead to a slightly lower full-precision baseline than the official reported performance. However, this does not affect the validity of our comparison, since all quantization and cache methods are applied to the same trained checkpoint without any additional fine-tuning. The resulting full-precision checkpoint is used as the original policy, and all reported improvements are measured under the same evaluation protocol.

A.2.2 Real-world Robot Settings

We conduct real-world experiments on an ALOHA-AgileX bimanual robot using a $\pi_{0.5}$-based fine-tuned policy. The policy is deployed in a server-client manner: the inference server loads the policy checkpoint, tokenizer, and normalization statistics, and performs model inference on GPU, while the robot-side client collects observations through ROS and sends them to the server for action prediction. Each inference input consists of three RGB views, including one front camera and two wrist cameras, together with the 14-dimensional joint state of the two robot arms. The robot publishes joint commands at 30 Hz with a default action chunk size of 50. We adopt an asynchronous sliding-window inference strategy, where a new inference request is triggered once the remaining buffered actions reach 10.

We evaluate two real-world manipulation tasks: *Open the Microwave* and *Fold the Clothes*. In *Open the Microwave*, the robot opens the microwave door, places a lunch box inside, and closes the door. In *Fold the Clothes*, the robot picks up a piece of clothing, folds it, and places it into a basket. Each method is evaluated over 10 trials per task. A trial is counted as successful only if all task stages are completed without human intervention. Unsafe or clearly failed executions are terminated and counted as failures. For example, in the *Open the Microwave* task, a trial is considered failed if the lunch box is dropped. In the *Fold the Clothes* task, we set a maximum execution time of 3 minutes; trials exceeding this limit are counted as failures.

A.2.3 Baselines and Evaluation Protocol

We compare QCache-VLA with both quantization and diffusion-cache baselines. For quantization, we use SmoothQuant and MBQ as representative post-training quantization methods. All quantization methods are applied to the same set of linear layers and use the same W8A8 setting. For diffusion-cache acceleration, we compare with EfficientVLA and DiCache. EfficientVLA follows its fixed-interval cache strategy, while DiCache is adapted to the VLA action expert using the first expert layer as the shallow online probe. For fair comparison under quantized inference, cache-based baselines are combined with MBQ in the W8A8 setting.

[1] https://github.com/RoboTwin-Platform/RoboTwin

[2] https://github.com/Physical-Intelligence/openpi

Algorithm 3 QCache-VLA: Residual-calibrated Dynamic Diffusion Cache

Input: Current expert input $\mathbf{h}_t^0$, denoising step index t, total steps T, threshold δ
Cache states: Previous early residual $\mathbf{r}_{prev}$, previous deep residual $\mathbf{R}_{prev}$ (computed or estimated)
Output: Expert output $\widehat{\mathbf{h}}_t^L$

▷ Step 1: Early residual probe

$\mathbf{h}_t^1 \leftarrow F_1(\mathbf{h}_t^0)$
$\mathbf{r}_t \leftarrow \mathbf{h}_t^1 - \mathbf{h}_t^0$
`use_cache` $\leftarrow$ `False`

▷ Step 2: Dynamic cache decision

if $\mathbf{r}_{prev}$ is available **and** $\mathbf{R}_{prev}$ is available **and** $t < T - 1$ **then**
    $d_t \leftarrow \dfrac{\|\mathbf{r}_t - \mathbf{r}_{prev}\|_1}{\|\mathbf{r}_{prev}\|_1 + \epsilon}$
    if $d_t < \delta$ **then**
        `use_cache` $\leftarrow$ `True`
    end if
end if

▷ Step 3: Cache reuse or full computation

if `use_cache` **then**
    $\alpha_t \leftarrow \dfrac{\|\mathbf{r}_t\|_2}{\|\mathbf{r}_{prev}\|_2 + \epsilon}$
    $\widehat{\mathbf{h}}_t^L \leftarrow \mathbf{h}_t^1 + \alpha_t \mathbf{R}_{prev}$
    $\mathbf{R}_{prev} \leftarrow \alpha_t \mathbf{R}_{prev}$
else
    $\mathbf{h}_t^L \leftarrow F_{2:L}(\mathbf{h}_t^1)$
    $\mathbf{R}_{prev} \leftarrow \mathbf{h}_t^L - \mathbf{h}_t^1$
    $\widehat{\mathbf{h}}_t^L \leftarrow \mathbf{h}_t^L$
end if
$\mathbf{r}_{prev} \leftarrow \mathbf{r}_t$
if $t = T - 1$ **then**
    Reset cache states
end if
return $\widehat{\mathbf{h}}_t^L$

Table 5: Detailed low-step cache comparison on RoboTwin.

Method	Quant.	Steps	Easy				Hard			
			beat_block hammer	stack_bowl three	adjust bottle	Avg.	beat_block hammer	stack_bowl three	adjust bottle	Avg.
$\pi_{0.5}$										
FP16	–	10	64	76	98	79.3	14	57	90	53.7
SmoothQuant	W8A8	10	58	75	97	76.7	18	54	89	53.7
MBQ	W8A8	10	58	76	97	77.0	18	50	90	52.7
QCache-VLA	W8A8	10	**64**	**79**	**98**	**80.3**	**20**	56	**90**	**55.3**
EfficientVLA*	W8A8	5	52	60	76	62.7	15	39	64	39.3
QCache-VLA	W8A8	**5**	**68**	**73**	**97**	**79.3**	**20**	**52**	**84**	**52**
DiCache*	W8A8	3	63	72	95	76.7	15	53	84	50.7
QCache-VLA	W8A8	**3**	**63**	**76**	**95**	**78**	**20**	**53**	**86**	**53**
EfficientVLA*	W8A8	2	17	0	30	15.6	6	0	7	4.3
DiCache*	W8A8	2	4	0	31	11.6	6	0	8	4.6
QCache-VLA	W8A8	**2**	**63**	**75**	**96**	**78.0**	**15**	**54**	**85**	**51.3**
π_0										
FP16	–	10	30	52	94	58.7	11	39	72	40.7
SmoothQuant	W8A8	10	29	57	91	59.0	10	34	69	37.7
MBQ	W8A8	10	30	58	91	59.7	11	33	69	37.7
QCache-VLA	W8A8	10	**30**	**63**	**92**	**61.7**	**11**	**33**	**69**	**37.7**
EfficientVLA*	W8A8	5	31	58	92	60.3	8	33	68	36.3
QCache-VLA	W8A8	**5**	**31**	**59**	**89**	**59.7**	**9**	**30**	**67**	**35.3**
DiCache*	W8A8	3	22	58	88	56.0	8	30	72	36.7
QCache-VLA	W8A8	**3**	**23**	**55**	**90**	**56.0**	**8**	**31**	**66**	**35.0**
DiCache*	W8A8	2	3	0	22	8.3	4	0	5	3.0
QCache-VLA	W8A8	**2**	**30**	**65**	**93**	**62.7**	**10**	**32**	**68**	**36.7**

*EfficientVLA and DiCache are combined with MBQ for a fair comparison under the same W8A8 quantization setting.

A.3 Additional Results

A.3.1 RoboTwin Results

In the main paper, we report EfficientVLA, DiCache, and QCache-VLA with 5, 3, and 2 denoising steps, respectively. These settings correspond to the stable low-step configurations of the baseline cache methods, while QCache-VLA is evaluated under a more aggressive 2-step setting. To provide a complete comparison, Table 5 reports the performance of different cache methods under 5-, 3-, and 2-step inference on RoboTwin.

The results show that QCache-VLA consistently achieves stronger performance under low-step inference. In particular, when reducing the inference process to only 2 denoising steps, EfficientVLA and DiCache suffer from severe performance degradation, while QCache-VLA still maintains competitive success rates on both π_0 and $\pi_{0.5}$. This demonstrates that the proposed residual-calibrated cache better preserves action dynamics under aggressive diffusion acceleration, validating the effectiveness of reusing calibrated deep residuals for intermediate denoising steps. These results also highlight the importance of accurate and adaptive cache reuse for complex simulation and real-world manipulation tasks.

Fig. 7 provides a qualitative comparison of different cache methods under the aggressive 2-step inference setting.

EfficientVLA and DiCache exhibit unstable action execution, such as inaccurate reaching, failed grasping, or severe trajectory deviation, which explains their sharp success-rate degradation in Table 5. In contrast, QCache-VLA maintains a more consistent manipulation trajectory and successfully completes the task, further demonstrating that accurate and adaptive cache reuse is critical for preserving action quality under extreme diffusion acceleration.

Table 6 compares output reconstruction errors after quantization. QCache-VLA consistently achieves the lowest MSE and comparable or lower MAE across RoboTwin tasks, demonstrating better output fidelity than SmoothQuant and MBQ.

Table 6: Quantization error analysis on RoboTwin tasks. We report output reconstruction errors after quantization, where both MSE and MAE are scaled by 10^4.

Method	*beat_block_hammer*		*stack_bowl_three*		*adjust_bottle*	
	MSE	MAE	MSE	MAE	MSE	MAE
SmoothQuant	1.10	52.18	0.98	41.94	1.07	58.80
MBQ	0.14	**21.43**	0.92	**33.99**	0.36	36.22
QCache-VLA	**0.12**	21.87	**0.88**	35.10	**0.25**	**28.17**

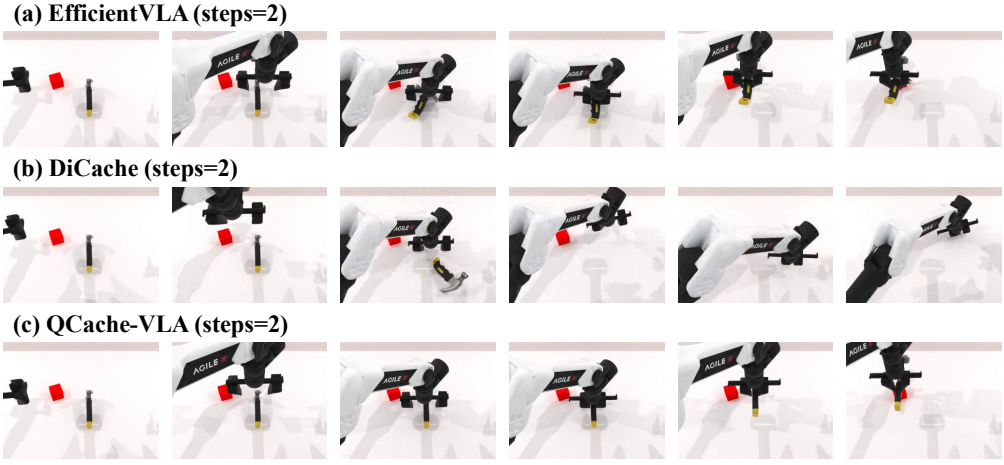


Figure 7: Qualitative comparison under aggressive 2-step inference on RoboTwin *beat_block_hammer* task. EfficientVLA and DiCache show unstable action execution, while QCache-VLA better preserves the action trajectory.

Fig. 8 and Fig. 9 compare open-loop action trajectories on RoboTwin. EfficientVLA introduces noticeable trajectory fluctuations under low-step inference, while QCache-VLA closely follows the original prediction and ground truth across most action dimensions. This confirms that the proposed modality-aware quantization and residual-calibrated cache better preserves action smoothness and temporal consistency during aggressive diffusion acceleration.

A.3.2 Real World Task

Fig. 10 and Fig. 11 compare different methods on the *Open the Microwave* and *Fold the Clothes* tasks. QCache-VLA achieves stable execution under the aggressive 2-step setting and successfully completes both tasks. By contrast, EfficientVLA suffers from unstable actions: in the microwave task, arm jitter leads to inaccurate button pressing and unreliable grasping of the lunch box; in the clothes-folding task, unstable motion and gripper commands prevent the robot from smoothly spreading and folding the clothes within the 3-minute limit. This highlights the importance of adaptive cache reuse for real-world long-horizon manipulation.

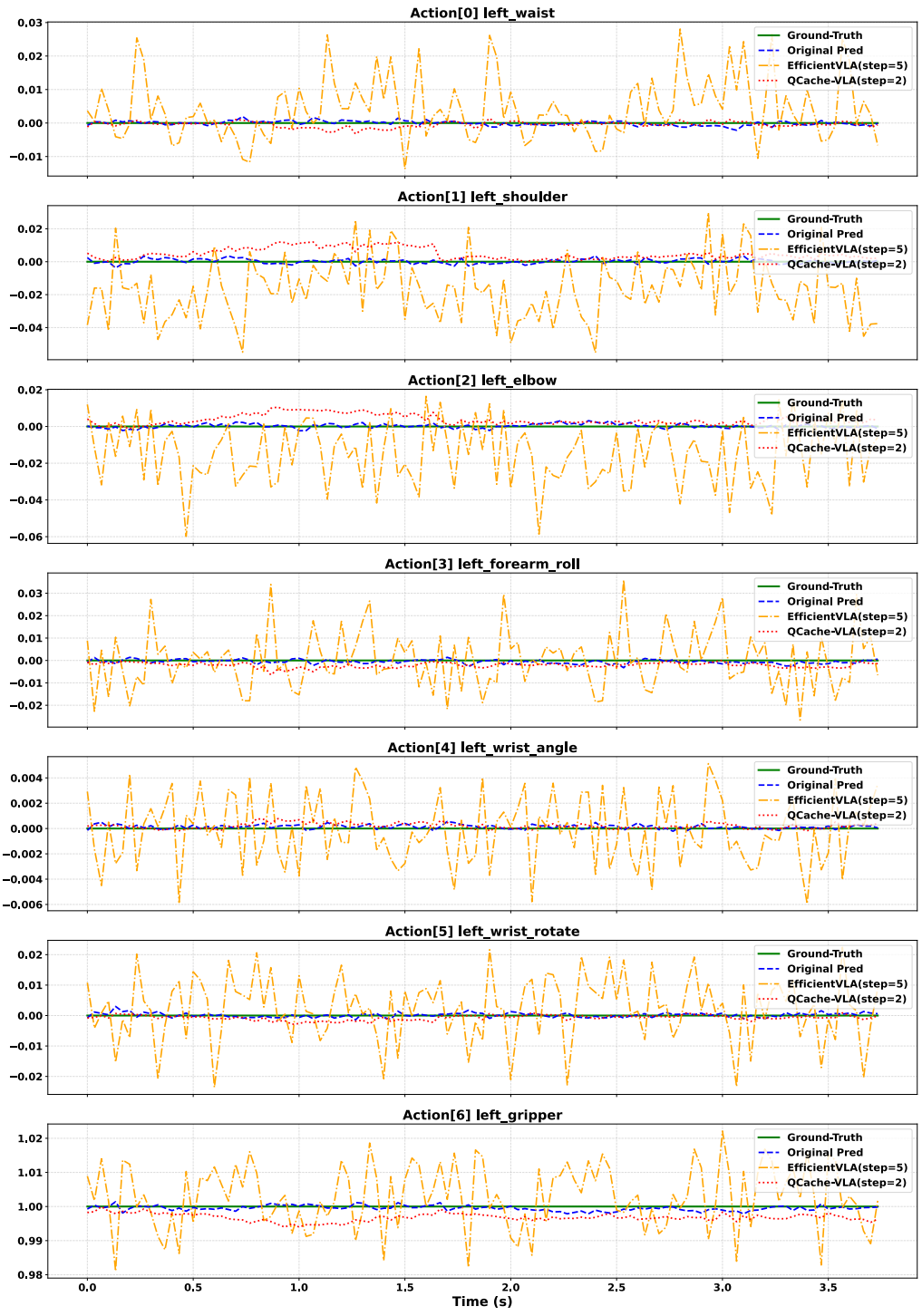


Figure 8: Open-loop trajectory comparison of the left-arm actions on RoboTwin *beat_block_hammer* task. We compare ground-truth actions, original model predictions, EfficientVLA, and QCache-VLA across seven left-arm action dimensions.

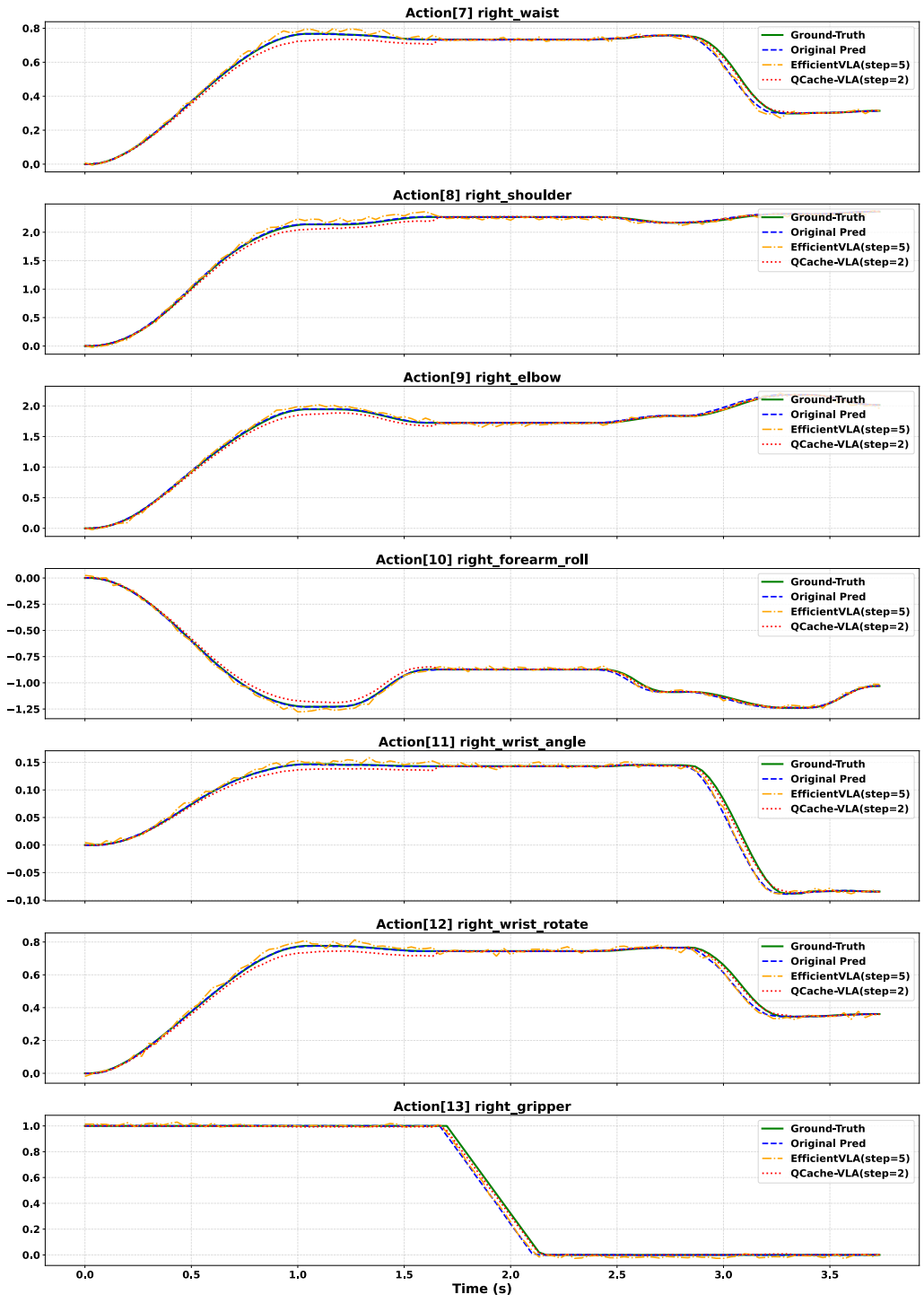


Figure 9: Open-loop trajectory comparison of the right-arm actions on RoboTwin *beat_block_hammer* task. QCache-VLA closely follows the original prediction and ground-truth trajectories, while EfficientVLA introduces larger fluctuations.

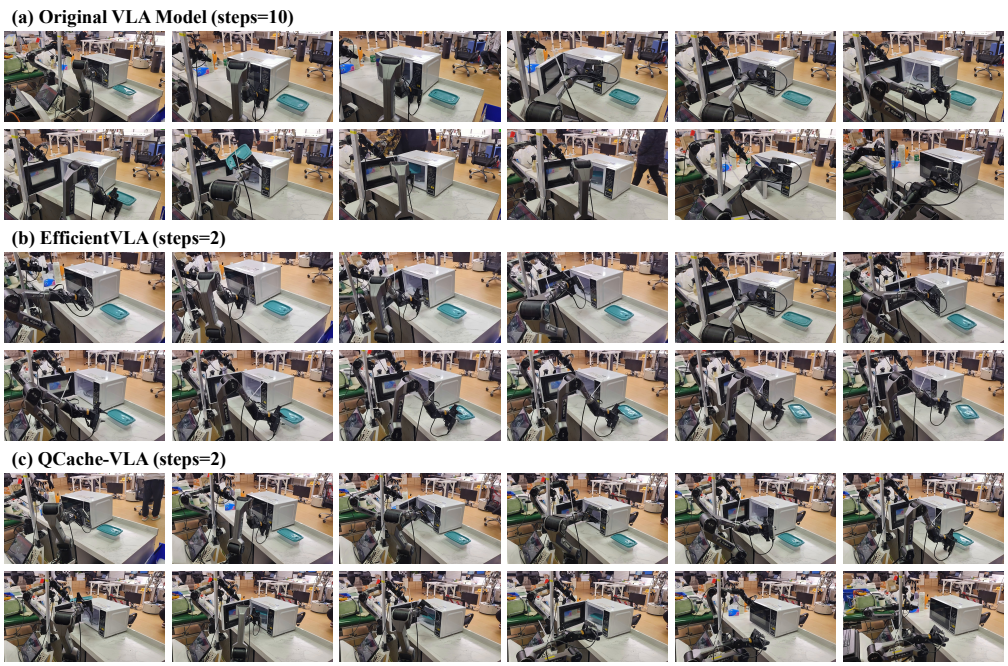


Figure 10: Qualitative comparison on the real-world *Open_the_Microwave* task.

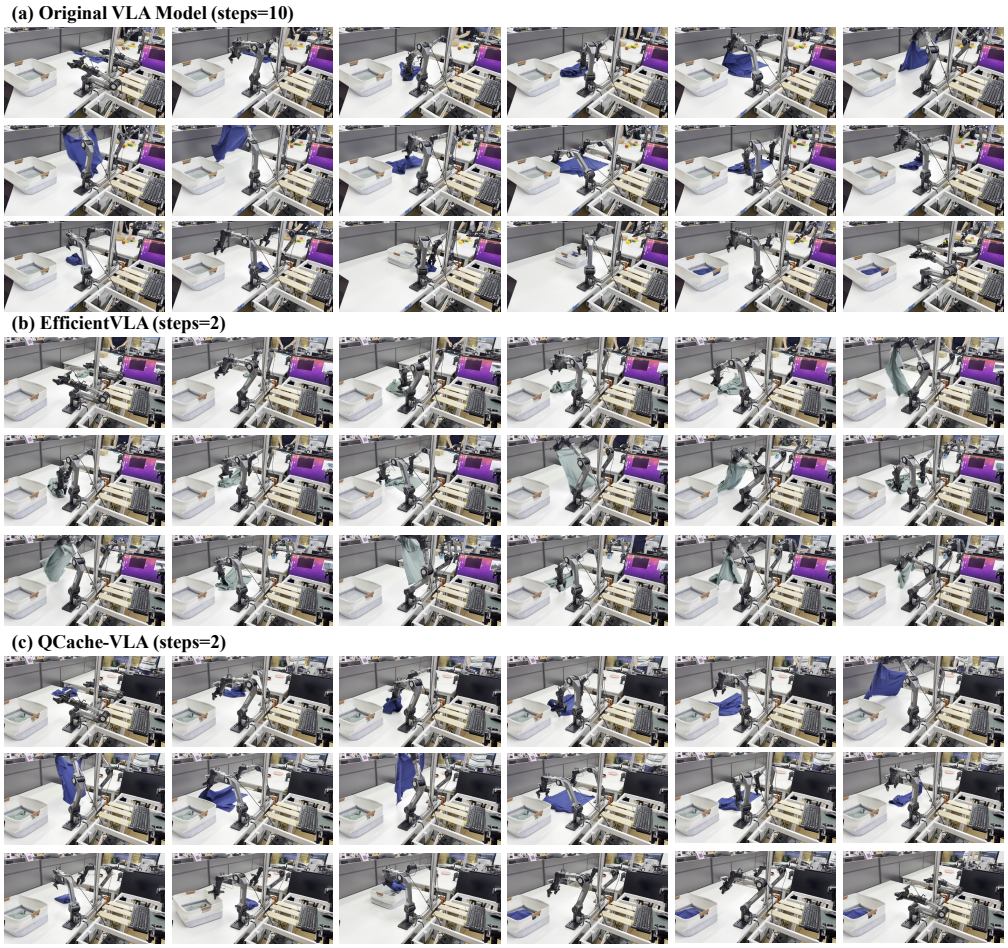


Figure 11: Qualitative comparison on the real-world *Fold_the_Clothes* task.