Q-WAM: 4-BIT QUANTIZATION OF WORLD ACTION MODELS WITH ACTION-SUBSPACE PROTECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

World Action Models (WAMs) jointly generate video and robot actions through iterative diffusion and perform strongly in robotic manipulation. However, their prohibitive compute and memory costs pose substantial deployment challenges. Post-training quantization (PTQ) can reduce these costs, but existing PTQ methods such as smoothing and rotation are insufficient to maintain the precision of action generation. To overcome this limitation, we propose **Q-WAM**, a new 4-bit weight-activation quantization for WAMs that preserves the actions the model generates. Specifically, we introduce the *Action Observability Gramian (AOG)*, which measures how much rounding errors in each weighted combination of a layer's input channels change the final action through all denoising steps. We also develop *Action-Subspace Protection (ASP)*, which keeps the few most action-sensitive channel combinations in a tiny 16-bit low-rank branch and quantizes the complementary weights and activations to 4 bits, both as dense matrix multiplications that run efficiently on GPUs. Finally, to preserve action quality with minimal overhead, we identify the experts that matter most for the generated action by aggregating the AOG-derived action mass across the layers of each expert and apply ASP only to those experts. We evaluate Q-WAM on three WAMs, both in simulation and in real-world deployment. On the RoboTwin 2.0 benchmark, it reaches 89.6–93.0% average success rate, within 1.1 percentage points of the 16-bit models, while reducing the memory of the quantized blocks by **3.1–3.4**×. Our method outperforms the strongest baseline, SVDQuant, by 2.5–8.7 percentage points. On a Unitree G1 humanoid and a bimanual UR3 robot, it improves success over SVDQuant by 12.8–17.6 percentage points. Our code is available.

1 INTRODUCTION

World Action Models (WAMs) couple action generation with prediction of future visual states to learn manipulation policies (Li et al., 2026; Ye et al., 2026b; Yuan et al., 2026). Despite strong task performance, their large backbones and iterative denoising make deployment costly. Recent efforts accelerate inference by removing test-time future generation (Yuan et al., 2026), simplifying action modules (Ma et al., 2026), and distilling denoising steps (Akbari et al., 2026b; Lyu et al., 2026). Memory demand remains a deployment bottleneck; for example, running ImageWAM (Zhang et al., 2026b) in bf16 requires 19.9 GB of GPU memory. Post-training quantization complements these advances by reducing weight and activation precision without retraining the full model. Achieving these savings requires keeping the generated actions accurate for reliable closed-loop control.

Mixed-precision quantization balances compression and accuracy by keeping sensitive components at higher precision and quantizing the rest more aggressively. Existing VLA methods allocate precision across weight tensors or individual channels (Akbari et al., 2026a; Xu et al., 2026). SVDQuant (Li et al., 2025) instead keeps a small low-rank part of each layer in 16 bits. This moves activation outliers into the weights and stores the dominant components of each weight in a 16-bit low-rank branch, so that only the residual is quantized to 4 bits. This works well for text-to-image diffusion models, but like most post-training methods, SVDQuant decides what to protect by reducing the quantization error of each linear layer's output on its own. In a WAM, however, what matters is the generated action, and a layer's own output error says little about how much the action changes once that error passes through the rest of the network and the remaining denoising steps. Because visual world modeling and action generation are tightly coupled through iterative denoising, choosing

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

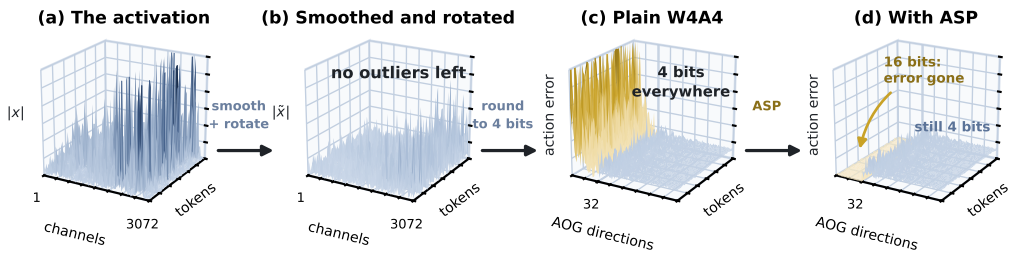

Figure 1: **Removing outliers is not enough.** **(a)** The activation entering a layer has a few outliers. **(b)** Smoothing and rotation remove them. **(c)** Quantizing the layer still damages the action, through a few AOG directions. **(d)** Our proposed ASP protects those directions and the damage is gone.

what to keep in higher precision requires tracing how quantization errors propagate through the full action-generation process. Our analysis shows that this downstream action sensitivity concentrates in compact activation subspaces, motivating the central question: *which activation directions should retain higher precision to keep the generated action accurate?*

A natural first question is whether reducing activation outliers suffices to preserve action fidelity. Smoothing (Xiao et al., 2023) and rotation (Ashkboos et al., 2024) reduce these outliers, but rounding the resulting activations to 4 bits still introduces considerable error, leaving existing quantization methods unsuitable for WAMs. Figure 1 illustrates the distinction between activation magnitude and action sensitivity: smoothing and rotation make activation magnitudes more uniform (Figure 1b), yet the resulting quantization errors can still distort the final action along a few sensitive directions (Figure 1c). In the examined layers, a compact subspace accounts for much of the final-action sensitivity, motivating selective protection of these directions (Figure 1d). This suggests keeping the action-sensitive subspace in higher precision and the complementary computation in W4A4.

Building on these observations, we propose **Q-WAM**, a post-training quantization framework for W4A4 world-action models. Q-WAM estimates an Action Observability Gramian (AOG), a per-layer sensitivity matrix measuring how activation errors affect the final action, using label-free randomized probes through the unrolled denoising process during calibration. Furthermore, we introduce Action-Subspace Protection (ASP), which splits each protected linear layer into two parallel computations whose outputs are summed. A low-dimensional branch projects activations onto the AOG's most action-sensitive directions and applies the corresponding projected weights in 16-bit precision, while the complementary computation uses 4-bit weights and activations. Since the selected subspace is fixed during inference, the 16-bit branch is small and adds negligible computational overhead. For a fixed protection rank, we show that the selected subspace minimizes a local approximation to action distortion when activation rounding errors have equal variance in all directions. For WAMs with Mixture-of-Transformers architecture, Q-WAM aggregates AOG sensitivity scores to concentrate protection on experts with high estimated action sensitivity.

We evaluate Q-WAM on three WAMs across 50 RoboTwin 2.0 tasks (Chen et al., 2026) and five real-world tasks on bimanual UR3 arms and a Unitree G1 humanoid. In simulation, Q-WAM limits the success rate drop from bf16 to at most 1.03 points while reducing targeted-block memory by 68–71%, and on physical robots it improves success over SVDQuant, the strongest prior method, by 12.8–17.6 points. Our contributions are as follows:

- We introduce the Action Observability Gramian (AOG), a per-layer matrix predicting how much a rounding error in each weighted combination of the layer's input channels changes the final action. We estimate it label-free with random probes.
- We propose Action-Subspace Protection (ASP), which keeps the most action-sensitive channel combinations of each layer in a small 16-bit branch and quantizes the rest to W4A4. Moreover, we introduced a method to use the AOG to decide which experts to protect.
- Across Fast-WAM, ImageWAM, and LingBot-VA, Q-WAM preserves near-bf16 success on RoboTwin 2.0 with substantially reduced memory. Evaluation on Unitree G1 and bimanual UR3 demonstrates improved physical task success over SVDQuant, while ablations examine subspace protection, weight quantization, and expert selection.

2 Related Works

World Action Models World Action Models couple action generation with prediction of future visual states, using world modeling to learn representations for control. UWM (Zhu et al., 2025) couples video and action diffusion, WorldVLA (Cen et al., 2025) unifies autoregressive image and action generation, and DreamZero (Ye et al., 2026b) and LingBot-VA (Li et al., 2026) jointly model video and actions for manipulation. Motus (Bi et al., 2026) combines understanding, video, and action experts in a Mixture-of-Transformers, and Cosmos Policy (Kim et al., 2026) adds actions, future observations, and values to a pretrained video model's latent diffusion. Recent work reduces deployment cost: Fast-WAM (Yuan et al., 2026) and GigaWorld-Policy (Ye et al., 2026a) keep video prediction only as training supervision, ImageWAM (Zhang et al., 2026b) conditions actions on image-editing representations, and Flash-WAM (Akbari et al., 2026b) distills sampling steps.

Post-Training Quantization PTQ reaches W4A4 in language models by smoothing (Xiao et al., 2023) or rotating (Ashkboos et al., 2024; Liu et al., 2025) activation outliers, and in diffusion models by handling timestep-dependent activations (Li et al., 2023; Wu et al., 2024; Zhao et al., 2025) or absorbing outliers into a 16-bit low-rank branch (Li et al., 2025), whose subspace is chosen from a local statistic, the weight's singular vectors. For robot policies, QVLA (Xu et al., 2026) and ActQuant (Akbari et al., 2026a) use action sensitivity but allocate precision to channels or whole tensors, while QuantVLA (Zhang et al., 2026a) calibrates scales for the action head. Q-WAM instead selects the protected subspace by the sensitivity of the generated action through the full denoising process.

Concurrent Work QuantWAMs (Zhou et al., 2026) studies WAM quantization through shared-basis outlier calibration, joint video–action saliency for layer-wise precision allocation, and rollout-guided denoising-step protection. Q-WAM focuses on protecting dense activation subspaces selected from the sensitivity of the final generated action through the unrolled sampler.

3 Preliminaries

World Action Models At each control step t, a policy receives an observation o_t, made of camera views and, for some policies, the robot's proprioceptive state, together with a language instruction l. It outputs an action chunk $a = a_{t:t+h} \in \mathbb{R}^m$ of h future steps of a d_a-dimensional control, with $m = h\, d_a$. A World Action Model produces this chunk through a world model. The observation and the instruction are encoded once into a cached conditioning $\mathbf{z}(o_t, l)$, which an action head reads at every denoising step. Some WAMs implement the world model and the action head as a Mixture-of-Transformers (Liang et al., 2025), in which per-modality experts hold separate weights and interact only through joint attention. Others share a single backbone between video and action (Li et al., 2026). The action head is trained by flow matching (Lipman et al., 2023). At inference it starts from $a^{(0)} \sim \mathcal{N}(0, I)$ and integrates the learned velocity field v_θ over T Euler steps,

$$a^{(s+1)} = a^{(s)} + \eta_s\, v_\theta\Big(a^{(s)}, \tau_s, \mathbf{z}\Big), \qquad s = 0, \dots, T-1, \tag{1}$$

and returns $a = a^{(T)}$. Each step is one transformer pass over the h action tokens, one per future step.

Post-Training Quantization A linear layer ℓ maps an activation $x_\ell \in \mathbb{R}^{d_\ell}$ to $y_\ell = W_\ell^\top x_\ell$ with $W_\ell \in \mathbb{R}^{d_\ell \times d'_\ell}$. Given a tensor X, b-bit quantization maps X onto a scaled integer grid,

$$Q_b(X) = \varsigma \cdot \mathrm{round}(X/\varsigma), \qquad \varsigma = \max|X| \,/\, (2^{b-1} - 1), \tag{2}$$

with one scale ς per group of consecutive entries. Weights are quantized once offline and activations per token at run time, and with 4-bit weights and activations the layer computes $Q_4(W_\ell)^\top Q_4(x_\ell)$ as one integer general matrix multiplication (GEMM). The largest entry of a group sets ς, so a single outlier coarsens the grid for the whole group, and activation outliers reach one to two orders of magnitude above typical values and move with the input. We therefore apply two exact transforms before quantization. SmoothQuant (Xiao et al., 2023) migrates outlier magnitude from the activation

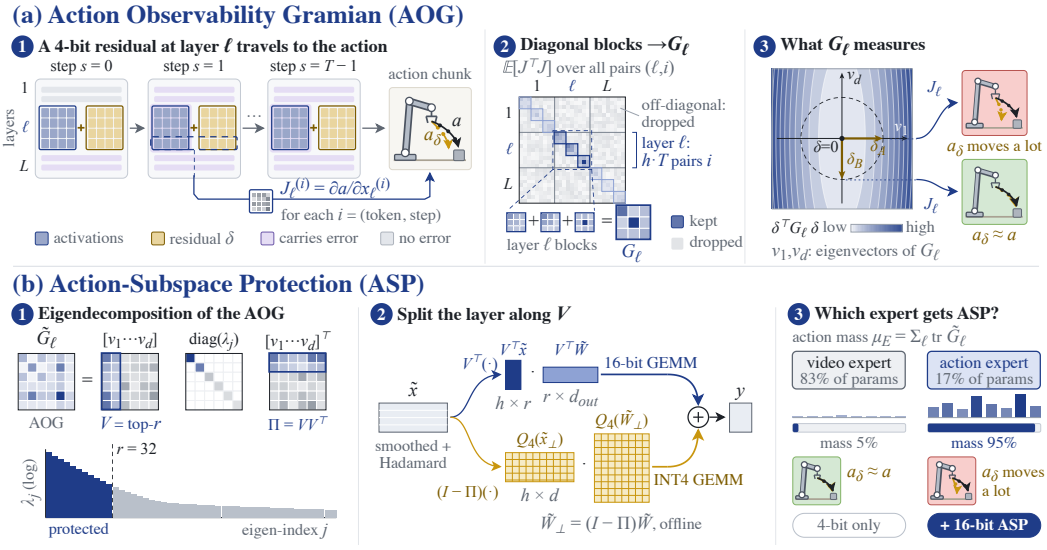

Figure 2: **Overview of Q-WAM. (a)** The AOG G_ℓ measures how far a rounding error in each weighted combination of layer ℓ's input channels moves the action in later denoising steps. **(b)** ASP keeps the most sensitive of these combinations in a 16-bit branch and quantizes the deflated remainder to 4 bits; the action mass decides which experts receive it.

to the weight with a per-channel factor $\gamma \in \mathbb{R}^{d_\ell}$, and a block Hadamard rotation H (Ashkboos et al., 2024) spreads the remaining outliers across coordinates,

$$\tilde{x} = H\,\mathrm{diag}(\gamma)^{-1}x, \qquad \widetilde{W} = H\,\mathrm{diag}(\gamma)\,W, \qquad \widetilde{W}^\top\tilde{x} = W^\top x. \tag{3}$$

A WAM block generates its normalization parameters at run time, so we apply H online with the fast Walsh–Hadamard transform (Fino & Algazi, 1976). The deployed layer computes $Q_4(\widetilde{W})^\top Q_4(\tilde{x})$, and Section 4 works in these transformed coordinates.

4 Method

We propose Q-WAM, a post-training W4A4 quantization method for World Action Models that allocates activation precision by the sensitivity of the generated action. Section 4.1 defines and estimates this sensitivity, the Action Observability Gramian (AOG). Section 4.2 introduces Action-Subspace Protection (ASP), which keeps the activation directions that the AOG marks as most sensitive in 16 bits and quantizes the remaining weights and activations to 4 bits. Section 4.3 aggregates the AOG over the layers of each expert to select the experts that receive this protection.

4.1 The Action Observability Gramian

Within one action prediction, layer ℓ runs once for each of the h action tokens at each of the T steps of Equation 1. We index these token–step pairs by $i \in \mathcal{I}_\ell$, write $x_\ell^{(i)}$ for the input activation at pair i, and write $\delta_\ell^{(i)} = Q_4(x_\ell^{(i)}) - x_\ell^{(i)}$ for its quantization residual. The residual changes with the input, so we treat it as random vector over calibration inputs (o_t, l). Let $\mathcal{A}$ be the action chunk a generated by the full-precision model and $\mathcal{A}_\delta$ the chunk generated when the activations carry the residuals $\delta = (\delta_\ell^{(i)})_{\ell,i}$. A first-order Taylor expansion gives

$$\mathcal{A}_\delta - \mathcal{A} = \sum_\ell \sum_{i \in \mathcal{I}_\ell} J_\ell^{(i)}\,\delta_\ell^{(i)} + O(\|\delta\|^2), \qquad J_\ell^{(i)} = \frac{\partial a}{\partial x_\ell^{(i)}} \in \mathbb{R}^{m \times d_\ell}, \tag{4}$$

The Jacobian $J_\ell^{(i)}$ says how the final action responds to a small change in the input of layer ℓ at pair i. It accounts for everything that happens after that point: the rest of the block, and all remaining

denoising steps. Our goal is that the quantized model generates the same action as the full-precision model, so we minimize the expected squared change of the action caused by the rounding errors. Inserting Equation 4 into this squared change gives one term for every layer and token–step pair, plus cross terms between different pairs. We treat the rounding errors as zero-mean, uncorrelated with each other, and uncorrelated with the Jacobians (Appendix A.2 states these approximations precisely). The cross terms then vanish, and since all token–step pairs of a layer share one error distribution, what remains is a sum with one term per layer:

$$\mathbb{E}\big\|\mathcal{A}_\delta - \mathcal{A}\big\|^2 \;\approx\; \sum_\ell \mathbb{E}\big[\delta_\ell^\top G_\ell\, \delta_\ell\big], \qquad G_\ell = \mathbb{E}\Big[\sum_{i\in\mathcal{I}_\ell} J_\ell^{(i)\top} J_\ell^{(i)}\Big] \in \mathbb{R}^{d_\ell \times d_\ell}. \tag{5}$$

We call G_ℓ the Action Observability Gramian (AOG) of layer ℓ. Equation 5 says what the AOG measures: if the input of layer ℓ carries a rounding error δ, then $\delta^\top G_\ell\, \delta$ approximates the expected squared change this error causes in the final action. This is useful for two reasons. First, it turns a question about the whole model, how far quantization moves the robot's action, into one matrix per layer that we compute once during calibration. Second, because G_ℓ is a matrix and not a single score, it tells how sensitive the action is to errors in every direction of the layer's input, that is, in every weighted combination of its channels. For each layer, this quadratic form is also exact to second order: at the full-precision model, G_ℓ is the Hessian of $\frac{1}{2}\mathbb{E}\|\mathcal{A}_\delta - \mathcal{A}\|^2$ with respect to the layer's input (Appendix A.1).

Estimating the AOG G_ℓ. Computing G_ℓ exactly is expensive: $J_\ell^{(i)}$ has one row per action coordinate, hd_a in total, and each row needs its own backward pass through all denoising steps. We avoid this with random probes $u \sim \mathcal{N}(0, I_m)$. A single backward pass of $\langle a, u\rangle$ returns $J_\ell^{(i)\top} u$ for every layer and token–step pair at once (Baydin et al., 2018), and since $\mathbb{E}[uu^\top] = I_m$, the expectation of $(J^\top u)(J^\top u)^\top$ over probes equals $J^\top J$ for every Jacobian J. Averaging over N calibration inputs with P probes each therefore gives the unbiased estimate

$$\widehat{G}_\ell = \frac{1}{NP}\sum_{n=1}^{N}\sum_{p=1}^{P}\sum_{i\in\mathcal{I}_\ell} \big(J_{\ell,n}^{(i)\top} u_{np}\big)\big(J_{\ell,n}^{(i)\top} u_{np}\big)^\top, \tag{6}$$

where $J_{\ell,n}^{(i)}$ is the Jacobian at input n. Each input then costs P backward passes instead of hd_a; we use P=12, about 37× fewer than the exact computation on Fast-WAM. Appendix A.4 gives the procedure and bounds the error of the trace and of the leading subspace of $\widehat{G}_\ell$.

The AOG predicts quantization damage. We check this prediction on Fast-WAM. For each of the linear layers in its action expert, we quantize that layer alone to W4A4 and measure the normalized root-mean-square error (NRMSE) of the generated action, which is the damage from quantizing the layer. Figure 3a compares this damage with two per-layer criteria. Each point is one layer, placed by its damage and by its score under a criterion, and each score is divided by its median over the layers so that both criteria share one axis. The first is the local error $\|\delta_\ell\|^2$, the squared quantization error of the layer input. Round-to-nearest minimizes it by construction, and widely used post-training methods (Lin et al., 2024; Frantar et al., 2023; Li et al., 2025) calibrate each layer or block from local quantities of this kind, the error or statistics of its own input and output, without looking at the generated action. The second is the AOG prediction $\mathbb{E}[\delta_\ell^\top G_\ell \delta_\ell]$. The AOG prediction tracks the damage with a Pearson correlation ρ=0.95, while the local error reaches ρ=0.04.

4.2 Action-Subspace Protection

The AOG weighs each direction of the activation by its effect on the action. A residual δ_ℓ costs $\mathbb{E}[\delta_\ell^\top G_\ell\, \delta_\ell]$, and we keep the few most costly directions in 16 bits. We find these directions in three steps. First, we apply the smoothing and rotation of Section 3, so the layer quantizes $\tilde{x} = H\,\mathrm{diag}(\gamma)^{-1}x$ instead of x. Second, we express the AOG in these coordinates. Since $x = \mathrm{diag}(\gamma)H\tilde{x}$, a rounding error δ of $\tilde{x}$ shifts x by $\mathrm{diag}(\gamma)H\delta$ and therefore costs $\delta^\top \widetilde{G}_\ell\, \delta$, with $\widetilde{G}_\ell = H\,\mathrm{diag}(\gamma)\, G_\ell\, \mathrm{diag}(\gamma)H$. Third, since $\widetilde{G}_\ell$ is symmetric and positive semidefinite, it has orthonormal eigenvectors v_j, $\widetilde{G}_\ell = \sum_j \lambda_j v_j v_j^\top$, and

$$\delta^\top \widetilde{G}_\ell\, \delta = \sum_j \lambda_j \big(v_j^\top \delta\big)^2, \qquad \lambda_1 \ge \lambda_2 \ge \cdots \ge 0. \tag{7}$$

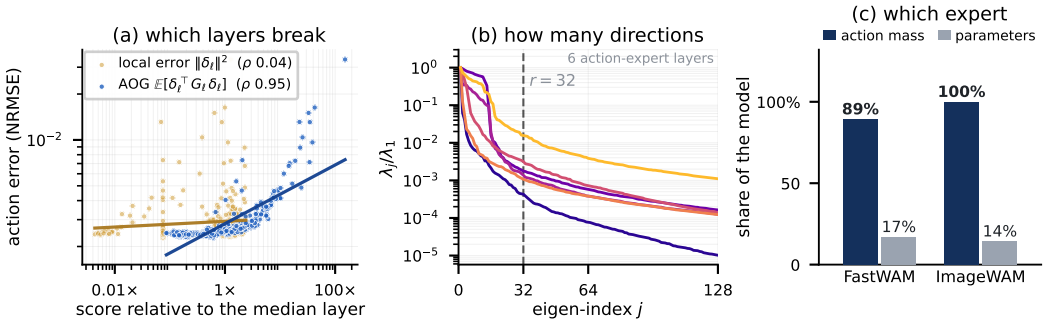

Figure 3: **Deciding what to protect with the AOG. (a)** Action error from quantizing a single layer, against the layer's local error and its AOG score. **(b)** The first 128 eigenvalues of the rotated AOG for several action-expert layers, which fall quickly; the dashed line marks the rank we keep. **(c)** The action expert's share of the action mass and of the parameters in each model.

The cost splits into one term per eigenvector: the component $v_j^\top \delta$ of the residual contributes $\lambda_j (v_j^\top \delta)^2$. A large eigenvalue marks a direction in which rounding changes the action strongly, and a small eigenvalue marks a direction to which the action barely reacts. The eigenvalues fall by orders of magnitude within a few dozen directions (Figure 3b), so only a few directions of each activation are sensitive. Each direction mixes many channels and is set by the action's response to errors, not by activation values, so we protect directions rather than channels. We call the span of the top r eigenvectors the layer's action subspace, keep the component of $\tilde{x}$ inside it in 16 bits, and quantize the rest to 4 bits.

Splitting the layer. Let $V \in \mathbb{R}^{d_\ell \times r}$ hold the top-r eigenvectors of $\widetilde{G}_\ell$, so that $V^\top V = I_r$, and let $\Pi = VV^\top$ project onto the action subspace. We split every activation into a protected component inside the action subspace and a remainder orthogonal to it,

$$\tilde{x} = \Pi\tilde{x} + (I - \Pi)\tilde{x} = \underbrace{V\big(V^\top \tilde{x}\big)}_{\text{protected component}} + \underbrace{\tilde{x}_\perp}_{\text{remainder}}, \qquad \tilde{x}_\perp = (I - \Pi)\tilde{x}. \tag{8}$$

Since $W^\top x = \widetilde{W}^\top \tilde{x}$, multiplying Equation 8 by $\widetilde{W}^\top$ splits the output of the layer into two products. The first, $\widetilde{W}^\top V(V^\top \tilde{x}) = (V^\top \widetilde{W})^\top (V^\top \tilde{x})$, runs over only $r \ll d_\ell$ dimensions. In the second, $I - \Pi$ is a symmetric projection, so it leaves $\tilde{x}_\perp$ unchanged and can be moved onto the weight, $\widetilde{W}^\top \tilde{x}_\perp = \widetilde{W}^\top (I - \Pi)\tilde{x}_\perp = \big((I - \Pi)\widetilde{W}\big)^\top \tilde{x}_\perp = \widetilde{W}_\perp^\top \tilde{x}_\perp$, where $\widetilde{W}_\perp = (I - \Pi)\widetilde{W}$ is the weight with its protected component removed. We keep the first product in 16 bits and quantize both factors of the second to 4 bits,

$$W^\top x = \widetilde{W}^\top \tilde{x} = \big(V^\top \widetilde{W}\big)^\top \big(V^\top \tilde{x}\big) + \widetilde{W}_\perp^\top \tilde{x}_\perp \approx \underbrace{\big(V^\top \widetilde{W}\big)^\top \big(V^\top \tilde{x}\big)}_{\text{low-rank 16-bit branch}} + \underbrace{Q_4(\widetilde{W}_\perp)^\top Q_4(\tilde{x}_\perp)}_{\text{4-bit deflated path}}. \tag{9}$$

We call this construction Action-Subspace Protection (ASP). Projecting $\tilde{x}$ before Q_4 keeps the protected component out of the 4-bit grid, and deflating the weight to $\widetilde{W}_\perp$ lets the rounding error δ reach the output only as $(I - \Pi)\delta$, up to the product of the weight and activation rounding errors. Both paths are dense GEMMs that run efficiently on GPUs, and the thin rank-r branch adds little overhead (Appendix B.2).

By Equation 7, the damage along v_j depends on how strongly the action reacts to that direction (λ_j) and how much rounding error lands on it. After the rotation, the rounding error has roughly the same mean square σ^2 in every direction, i.e., it is *isotropic*, so the expected damage along v_j is $\sigma^2 \lambda_j$. The top-r eigenvectors are therefore the best r directions to protect, which Appendix A.3 proves.

4.3 Where to Protect: The Action Mass of an Expert

Each protected layer carries a 16-bit branch, so in a Mixture-of-Transformers we protect only the experts whose rounding errors damage the action most. We denote an expert by E, the set of layers

Table 1: RoboTwin 2.0 results. BPW is the weight cost; Mem is the targeted-block memory.

Model	Method	Precision	BPW	Clean	Randomized	Avg.	Mem (GB)
Fast-WAM (2-expert MoT, video generation backbone)							
	bf16 (upper bound)	W16A16	16.00	92.28	91.34	91.81	11.85
	SmoothQuant	W4A4	4.00	21.98	16.48	19.23	2.98
	ViDiT-Q	W4A4	4.41	82.00	80.48	81.24	3.61
	SVDQuant	W4A4	4.84	89.12	87.38	88.25	3.60
	Q-WAM (Ours)	W4A4	**4.62**	**91.62**	**89.94**	**90.78**	3.44
ImageWAM (2-expert MoT, image editing backbone)							
	bf16 (upper bound)	W16A16	16.00	92.82	93.70	93.26	9.11
	SmoothQuant	W4A4	4.00	59.04	59.36	59.20	2.36
	ViDiT-Q	W4A4	4.76	78.76	82.08	80.42	3.05
	SVDQuant	W4A4	4.75	84.34	84.22	84.28	2.78
	Q-WAM (Ours)	W4A4	**4.58**	**93.00**	**92.94**	**92.97**	2.69
LingBot-VA (shared backbone, not an MoT)							
	bf16 (upper bound)	W16A16	16.00	90.84	90.20	90.52	10.16
	SmoothQuant	W4A4	4.00	81.68	72.24	76.96	2.80
	ViDiT-Q	W4A4	4.41	80.82	74.04	77.43	3.18
	SVDQuant	W4A4	4.76	85.84	83.12	84.48	3.27
	Q-WAM (Ours)	W4A4	**4.76**	**90.20**	**88.92**	**89.56**	3.27

that process one modality, and score it by the expected damage its 4-bit rounding does to the action. With the isotropic rounding error of Section 4.2, every direction of a layer receives error σ^2, so the layer's term in Equation 5 is σ^2 times the sum of its eigenvalues, $\mathbb{E}[\delta_\ell^\top \widetilde{G}_\ell \delta_\ell] = \sigma^2 \operatorname{tr} \widetilde{G}_\ell$. Taking the same σ^2 for all layers and summing over the layers of E, the expert contributes $\sigma^2 \mu_E$, where

$$\mu_E = \sum_{\ell \in E} \operatorname{tr}\left(\widetilde{G}_\ell\right) \tag{10}$$

is the action mass of E. Protecting a layer removes the damage along its top-r eigenvectors, which is the most that any rank-r branch can remove; Appendix A.3 proves this. As depicted in Figure 3b, the eigenvalues fall steeply, so the protected directions carry nearly all of a layer's damage, and the gain from protecting an expert grows with its action mass. ASP therefore protects the experts with the largest action mass per added 16-bit value. As depicted in Figure 3c, on both Mixture-of-Transformers models the action expert holds under a fifth of the parameters yet carries 89.3% of the action mass on Fast-WAM (Appendix C.2 visualize it per layer) and 99.99% on ImageWAM, so ASP protects the action expert alone. LingBot-VA shares one backbone, so ASP covers all of its layers. Furthermore, table 4 shows that protecting the other expert changes the success rate by less than the run-to-run RoboTwin variation.

5 Experiments

Experimental Setup We evaluate Q-WAM on three WAM designs: Fast-WAM (Yuan et al., 2026) and ImageWAM (Zhang et al., 2026b) add an action expert to a video-generation and an image-editing backbone in a Mixture-of-Transformers, and the released open-sourced LingBot-VA (Li et al., 2026) uses one shared backbone for video and action. We use a protection rank of $r = 32$ and group-wise W4A4 quantization with a group size of 32, retaining the protected branches in 16-bit precision. Calibration runs on NVIDIA H100 GPUs, simulation evaluation on NVIDIA L40S GPUs, and real-world inference on an NVIDIA RTX 5090. Implementation details are provided in Appendix B.

Benchmarks and Baselines We evaluate all three models on RoboTwin 2.0 (Chen et al., 2026) simulation under clean and randomized conditions, with 100 demonstrations. Real-world evaluation covers Fast-WAM and ImageWAM on five tasks across two embodiments (Figure 4). The Unitree G1 performs *tool sorting* and *item classification*, and a bimanual UR3 arm performs *table cleaning*, *cube stacking*, and *drawer manipulation*, with 25 trials per task. We report task success rates and compare against bf16 policies, SmoothQuant (Xiao et al., 2023), ViDiT-Q (Zhao et al., 2025), and SVDQuant (Li et al., 2025) in simulation, and bf16 and SVDQuant on physical robots. Quantized methods use W4A4, with effective bits per weight (BPW) accounting for storage overhead.

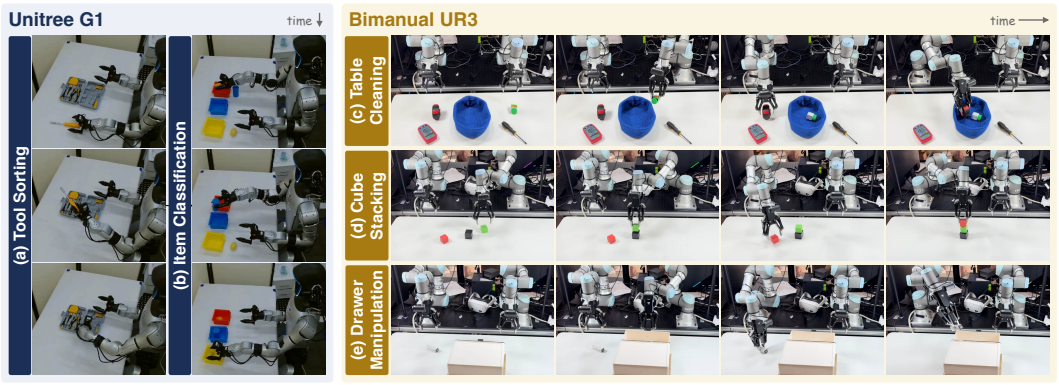

Figure 4: Real-world evaluation tasks. Frames progress from left to right within each task.

Table 2: Real-world results (successes/trials) on two embodiments and five tasks.

		Unitree G1		UR3 Bimanual			
Model	Method	Tool Sort	Item Cls.	Table Clean	Drawer	Cube Stacking	Avg. (%)
Fast-WAM	bf16	19/25	13/25	17/25	20/25	12/25	64.8
	SVDQuant	16/25	10/25	12/25	12/25	4/25	43.2
	Q-WAM (Ours)	**18/25**	**11/25**	**14/25**	**19/25**	**8/25**	**56.0**
ImageWAM	bf16	21/25	17/25	22/25	23/25	21/25	83.2
	SVDQuant	15/25	12/25	19/25	**22/25**	5/25	58.4
	Q-WAM (Ours)	**19/25**	**14/25**	**21/25**	**22/25**	**19/25**	**76.0**

5.1 Main Results

Simulation Results Table 1 shows that Q-WAM achieves the highest success rates among the quantized methods in both clean and randomized conditions across all three models. Its average success reaches 90.78% on Fast-WAM, 92.97% on ImageWAM, and 89.56% on LingBot-VA, trailing the respective bf16 baselines by only 1.03, 0.29, and 0.96 percentage points. Compared with SVDQuant, Q-WAM improves average success by 2.53, 8.69, and 5.08 percentage points, respectively. These improvements come at lower storage cost on the two multi-expert models. BPW decreases from 4.84 to 4.62 on Fast-WAM and from 4.75 to 4.58 on ImageWAM. On LingBot-VA, Q-WAM achieves its gain at the same 4.76 BPW and 3.27 GB of memory as SVDQuant. The improvements persist under randomized conditions, where Q-WAM exceeds SVDQuant by 2.56, 8.72, and 5.80 points across the three models. In comparison, SmoothQuant and ViDiT-Q exhibit substantial model-dependent degradation, particularly on ImageWAM, where their average success rates fall to 59.20% and 80.42%, versus 92.97% for Q-WAM. Overall, Q-WAM preserves near-bf16 task performance across both multi-expert and shared-backbone WAMs while reducing the memory of the targeted blocks by 68–71% (Table 1). Figure 5 reports latency with real 4-bit kernels on one NVIDIA L40S: the quantized layers that process the action tokens run $1.06\times$, $1.56\times$ and $1.41\times$ faster than bf16 on Fast-WAM, ImageWAM and LingBot-VA. These layers handle short action chunks, so their cost is dominated by loading weights, which 4-bit storage reduces.

Real-World Experiments Table 2 evaluates whether these gains extend to physical execution across two embodiments. Q-WAM achieves average success rates of 56.0% on Fast-WAM and 76.0% on ImageWAM, exceeding SVDQuant by 12.8 and 17.6 percentage points, respectively. It matches or exceeds SVDQuant on every model–task pair. The largest improvement occurs on ImageWAM cube stacking, where Q-WAM succeeds in 19/25 trials, compared with 5/25 for SVDQuant and 21/25 for bf16. While gaps of 8.8 and 7.2 points to bf16 remain, the results demonstrate improved closed-loop performance under quantization on physical robots.

Table 3: Component ablation on RoboTwin 2.0 at a fixed weight group size.

Model	Method	BPW	Clean	Randomized	Avg.	Mem (GB)
Fast-WAM	bf16 (upper bound)	16.00	92.28	91.34	91.81	11.85
	per-group W4A4	4.50	82.60	81.80	82.20	3.35
	+ smoothing and rotation	4.50	87.78	87.32	87.55	3.35
	+ **ASP (Ours)**	**4.62**	**91.62**	**89.94**	**90.78**	3.44
ImageWAM	bf16 (upper bound)	16.00	92.82	93.70	93.26	9.11
	per-group W4A4	4.50	21.16	20.14	20.65	2.64
	+ smoothing and rotation	4.50	86.78	88.06	87.42	2.64
	+ **ASP (Ours)**	**4.58**	**93.00**	**92.94**	**92.97**	2.69
LingBot-VA	bf16 (upper bound)	16.00	90.84	90.20	90.52	10.16
	per-group W4A4	4.50	83.44	78.24	80.84	3.11
	+ smoothing and rotation	4.50	87.52	82.64	85.08	3.11
	+ **ASP (Ours)**	**4.76**	**90.20**	**88.92**	**89.56**	3.27

Table 4: Adding ASP to the second expert on top of the action expert. † The expert Q-WAM selects by action mass.

model	ASP applied to	BPW	clean	randomized	Avg.
Fast-WAM	action expert†	4.62	91.62	89.94	90.78
	+ video expert	4.80	91.04	90.68	90.86
ImageWAM	action expert†	4.58	93.00	92.94	92.97
	+ editing expert	4.75	93.52	93.46	93.49

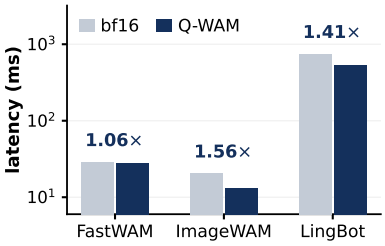

Figure 5: Block-level latency.

5.2 Ablation Analysis

Table 3 adds smoothing and rotation, then ASP, to per-group W4A4 at a fixed group size. Smoothing and rotation raise average success by 5.35, 66.77, and 4.24 points on Fast-WAM, ImageWAM, and LingBot-VA without changing BPW, but still leave gaps of 4.26, 5.84, and 5.44 points to bf16. ASP recovers 3.23, 5.55, and 4.48 of these points (76–95% of the remaining gap) under both clean and randomized conditions, for only 0.08–0.26 additional BPW (0.05–0.16 GB). Removing outliers is therefore not enough; protecting the action-sensitive subspace is what brings W4A4 close to bf16. Appendix C.1 shows that adding SVDQuant's low-rank weight protection on top of ASP costs extra bits without improving success. Moreover, Appendix C.3 provides a further ablation on the sensitivity to the calibration data.

On the two multi-expert models, the results above apply ASP only to the action expert, which the action mass μ_E selects (Section 4.3). Table 4 tests whether the other expert (video or image editing) also needs it. Extending ASP to it adds 0.17–0.18 BPW, more than ASP on the action expert itself costs, yet changes average success by only $+0.08$ and $+0.52$, less than the 1.12-point gap between two calibration sets (Appendix C.3). The action expert alone therefore captures the benefit of ASP, and μ_E correctly identifies where protection is needed.

6 Conclusion

In this work, we present Q-WAM, a post-training W4A4 quantization method for WAMs. To address the gap between a layer's local error and its effect on the action, we propose the Action Observability Gramian, which measures how much a rounding error in each weighted combination of a layer's input channels moves the action. To address the action's sensitivity to a few of these combinations, we propose Action-Subspace Protection, which keeps them in a small 16-bit branch and quantizes the rest to 4 bits. On three WAMs, Q-WAM stays within 1.1 points of bf16 on RoboTwin 2.0, reduces the memory of the quantized blocks by 3.1–3.4×, and outperforms SVDQuant in simulation and on two physical robots.

AI USE STATEMENT

In this work, we used large language models (LLMs) to implement some parts of the experiments. The methodology and the experiments were not designed with generative AI tools, and the structure of the paper was not generated by AI. Additionally, we used LLMs for grammar checking and polishing the writing of the paper, and to find relevant papers for the literature review. We have reviewed all AI-assisted work: two of the authors checked the LLM-assisted implementation, and the literature review was also checked manually. To minimize the risk of factual inaccuracies or citation errors, every model-edited sentence underwent human review, and all references were carefully cross-checked with their primary sources. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Arash Akbari, Arman Akbari, Masih Eskandar, Qitao Tan, Yixiao Chen, Jingwu Luo, Bertha Pangaribuan, Liyun Zhang, Jennifer Dy, Geng Yuan, Xue Lin, Gaowen Liu, Stratis Ioannidis, and Yanzhi Wang. ActQuant: Sub-4-bit action-guided quantization for vision-language-action models, 2026a. URL https://arxiv.org/abs/2605.24011.

Arman Akbari, Ci Zhang, Arash Akbari, Lin Zhao, Yixiao Chen, Weiwei Chen, Xuan Zhang, Geng Yuan, and Yanzhi Wang. Flash-WAM: Modality-aware distillation for world action models, 2026b. URL https://arxiv.org/abs/2606.05254.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2404.00456.

Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM*, 58(2):8:1–8:34, 2011.

Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 35101–35113, 2026. arXiv:2512.13030.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, Deli Zhao, and Hao Chen. WorldVLA: Towards autoregressive action world model, 2025. URL https://arxiv.org/abs/2506.21539.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2506.18088.

Ky Fan. On a theorem of Weyl concerning eigenvalues of linear transformations I. *Proceedings of the National Academy of Sciences*, 35(11):652–655, 1949.

Bernard J. Fino and V. Ralph Algazi. Unified matrix treatment of the fast Walsh–Hadamard transform. *IEEE Transactions on Computers*, C-25(11):1142–1146, 1976.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.17323.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53 (2):217–288, 2011.

Michael F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics–Simulation and Computation*, 18(3):1059–1076, 1989.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos Policy: Fine-tuning video models for visuomotor control and planning. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2601.16163.

Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical Fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Causal world modeling for robot control, 2026. URL https://arxiv.org/abs/2601.21998.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing outliers by low-rank component for 4-bit diffusion models. In *International Conference on Learning Representations (ICLR)*, pp. 97831–97858, 2025. arXiv:2411.05007.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17535–17545, 2023. arXiv:2302.04304.

Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2102.05426.

Weixin Liang, Lili Yu, Liang Luo, Srinivasan Iyer, Ning Dong, Chunting Zhou, Gargi Ghosh, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, and Xi Victoria Lin. Mixture-of-Transformers: A sparse and scalable architecture for multi-modal foundation models. *Transactions on Machine Learning Research (TMLR)*, 2025. arXiv:2411.04996.

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. In *Conference on Machine Learning and Systems (MLSys)*, pp. 87–100, 2024. arXiv:2306.00978.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.02747.

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 (ICLR)*, pp. 92009–92032, 2025. arXiv:2405.16406.

Jing Lyu, Shuanghao Bai, Runze Xiao, Zhenyu Liao, Wenxing Tan, Zihan Tang, Ruochuan Shi, Cheng Peng, Yuheng Ji, Yihao Wang, Badong Chen, Pengwei Wang, Zhongyuan Wang, and Xiaoguang Zhao. DIDO: Distilling interaction-centric dynamics into one-step denoising for world action models, 2026. URL https://arxiv.org/abs/2609.15570.

Liheng Ma, Rui Heng Yang, Zhanguang Zhang, Mateo Clemente, Ziwen Hu, Tongtong Cao, and Yingxue Zhang. Faster-WAM: Do world action models need deep action modules?, 2026. URL https://arxiv.org/abs/2608.02365.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2408–2417, 2015.

Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pp. 142–155. SIAM, 2021.

Farbod Roosta-Khorasani and Uri Ascher. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics*, 15(5):1187–1212, 2015.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Junyi Wu, Haoxuan Wang, Yuzhang Shang, Mubarak Shah, and Yan Yan. PTQ4DiT: Post-training quantization for diffusion transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.16005.

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 (ICML)*, 2023. arXiv:2211.10438.

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. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2602.03782.

Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Hao Li, Hengtao Li, Jie Li, Jindi Lv, Jingyu Liu, Min Cao, Peng Li, Qiuping Deng, Wenjun Mei, Xiaofeng Wang, Xinze Chen, Xinyu Zhou, Yang Wang, Yifan Chang, Yifan Li, Yukun Zhou, Yun Ye, Zhichao Liu, and Zheng Zhu. GigaWorld-Policy: An efficient action-centered world–action model, 2026a. URL https://arxiv.org/abs/2603.17240.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies, 2026b. URL https://arxiv.org/abs/2602.15922.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-WAM: Do world action models need test-time future imagination?, 2026. URL https://arxiv.org/abs/2603.16666.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. QuantVLA: Scale-calibrated post-training quantization for vision-language-action models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39539–39549, 2026a. arXiv:2602.20309.

Yuyang Zhang, Wenyao Zhang, Zekun Qi, He Zhang, Haitao Lin, Jingbo Zhang, Yao Mu, Xiaokang Yang, Wenjun Zeng, and Xin Jin. ImageWAM: Do world action models really need video generation, or just image editing?, 2026b. URL https://arxiv.org/abs/2606.19531.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Rui Wan, Widyadewi Soedarmadji, Enshu Liu, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, Huazhong Yang, Xuefei Ning, and Yu Wang. ViDiT-Q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *International Conference on Learning Representations (ICLR)*, pp. 65811–65841, 2025. arXiv:2406.02540.

Jiacheng Zhou, Jinfan Lv, Ruixuan Li, Longtai Zhang, Yan Wang, Wenqiang Zhang, and Lizhe Qi. QuantWAMs: Calibrating at the right granularity for world action models, 2026. URL https://arxiv.org/abs/2607.28405.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025.

A Derivations for the Action Observability Gramian

Throughout, $\mathcal{A}_\ell$ is the map from layer ℓ's input activation $x_\ell \in \mathbb{R}^{d_\ell}$ to the generated action chunk $a \in \mathbb{R}^m$ through the unrolled T-step denoiser of Equation 1, $J_\ell = \partial a / \partial x_\ell$ is its Jacobian, and $G_\ell = \mathbb{E}[J_\ell^\top J_\ell]$ as in Equation 5. The expectation is over calibration inputs, as in Section 4.1, and the token–step index i is suppressed wherever it plays no role.

A.1 G_ℓ is the Hessian of the action error at the full-precision point

For a fixed perturbation δ_ℓ of layer ℓ's activation, define the action-error objective

$$F_\ell(\delta_\ell) = \tfrac{1}{2}\,\mathbb{E}\big\|\xi(\delta_\ell)\big\|^2, \qquad \xi(\delta_\ell) = \mathcal{A}_\ell(x_\ell + \delta_\ell) - \mathcal{A}_\ell(x_\ell) \in \mathbb{R}^m,$$

so that $\xi(0) = 0$. Differentiating twice with respect to δ_ℓ,

$$\nabla^2 F_\ell(\delta_\ell) = \mathbb{E}\Big[\underbrace{J_\ell(\delta_\ell)^\top J_\ell(\delta_\ell)}_{\text{Gauss–Newton}} + \underbrace{\textstyle\sum_{k=1}^m \xi_k(\delta_\ell)\,\nabla^2 \mathcal{A}_{\ell,k}(x_\ell + \delta_\ell)}_{\text{residual curvature}}\Big],$$

where $J_\ell(\delta_\ell)$ is the Jacobian evaluated at $x_\ell + \delta_\ell$. The second term is weighted by the components of ξ, which vanish at $\delta_\ell = 0$, so

$$\nabla^2 F_\ell(0) = \mathbb{E}\big[J_\ell^\top J_\ell\big] = G_\ell.$$

Section 4.1 uses two consequences. First, G_ℓ is the exact Hessian of the action error at the full-precision point, so the quadratic form of Equation 5 needs no assumption about training convergence, unlike loss-based PTQ, which has to argue that the loss gradient vanishes. Second, $G_\ell \succeq 0$ as a sum of Gram matrices, which is what licenses the eigendecomposition of Section 4.2. If the action head is read as a Gaussian likelihood with fixed isotropic covariance, G_ℓ is also its Fisher information up to the covariance scale, the usual Gauss–Newton/Fisher correspondence (Schraudolph, 2002; Martens, 2020; Kunstner et al., 2019). In either reading it is built from the model's own output and never from ground-truth actions.

A.2 The block-diagonal reduction and the cross terms it discards

Section 4.1 reduces the model-wide curvature G to one term per layer. Write $\Delta = \mathbb{E}\|\mathcal{A}_\delta - \mathcal{A}\|^2$ for the expected squared action change that Equation 5 approximates. Inserting the first-order deviation $J\delta = \sum_\ell \sum_{i \in \mathcal{I}_\ell} J_\ell^{(i)} \delta_\ell^{(i)}$ of Equation 4 and writing it out over the layer and token–step indices gives a double sum,

$$\Delta \approx \underbrace{\sum_{\ell,i} \mathbb{E}\Big[\delta_\ell^{(i)\top} J_\ell^{(i)\top} J_\ell^{(i)} \delta_\ell^{(i)}\Big]}_{\text{same } (\ell,i)\text{: kept}} + \underbrace{\sum_{(\ell,i)\neq(\ell',i')} \mathbb{E}\Big[\delta_\ell^{(i)\top} J_\ell^{(i)\top} J_{\ell'}^{(i')} \delta_{\ell'}^{(i')}\Big]}_{\text{different } (\ell,i)\text{: dropped}}. \tag{11}$$

The additive-noise model of Section 4.1 takes the residuals to be zero-mean, uncorrelated with the Jacobians, and uncorrelated with each other across different (ℓ, i). Under it each cross term factors as $\operatorname{tr}\big(\mathbb{E}[J_\ell^{(i)\top} J_{\ell'}^{(i')}]\,\mathbb{E}[\delta_{\ell'}^{(i')} \delta_\ell^{(i)\top}]\big) = 0$, and each kept term factors as $\operatorname{tr}\big(\mathbb{E}[J_\ell^{(i)\top} J_\ell^{(i)}]\,\Sigma_\ell\big)$ with $\Sigma_\ell = \mathbb{E}[\delta_\ell \delta_\ell^\top]$ the residual covariance shared by the token–step pairs of layer ℓ. Summing the pairs of each layer gives $\Delta \approx \sum_\ell \operatorname{tr}(G_\ell \Sigma_\ell) = \sum_\ell \mathbb{E}[\delta_\ell^\top G_\ell \delta_\ell]$ with $G_\ell = \mathbb{E}[\sum_i J_\ell^{(i)\top} J_\ell^{(i)}]$, which is Equation 5.

The model is approximate. Adjacent denoising steps see similar activations and can leave correlated residuals. It is the block-diagonal reduction that K-FAC makes across layers (Martens & Grosse, 2015) and blockwise PTQ makes across blocks (Li et al., 2021), and Figure 3a is its empirical check, since the kept terms alone predict the measured single-layer W4A4 action damage at ρ=0.95 across all 300 action-expert linears.

The reduction is applied to the curvature of the action map. Layer-wise PTQ (Section 3) applies the same block-diagonal step and then also replaces the objective by each layer's local output error, which removes all downstream information from the curvature. The two steps are independent, and Q-WAM takes only the first.

A.3 Optimality of the top eigenvectors

This appendix states and proves the result that Section 4.2 uses to choose the protected subspace. Let $V \in \mathbb{R}^{d_\ell \times r}$ have orthonormal columns, let $\Pi = VV^\top$, and let $\delta = Q_4(\tilde{x}_\perp) - \tilde{x}_\perp$ be the residual of the 4-bit path in Equation 9. The deflated weight passes $\widetilde{W}^\top(I-\Pi)\delta$ to the output, so the rest of the network sees the residual $(I-\Pi)\delta$. By Equation 5, the expected per-layer distortion of the 4-bit path is

$$\mathbb{E}\big[\delta^\top(I-\Pi)\,\widetilde{G}_\ell\,(I-\Pi)\delta\big] = \operatorname{tr}\big((I-\Pi)\widetilde{G}_\ell(I-\Pi)\,\Sigma_\delta\big), \qquad \Sigma_\delta = \mathbb{E}[\delta\delta^\top].$$

Isotropic rounding error. The residual is isotropic when its covariance is a multiple of the identity, $\Sigma_\delta = \sigma^2 I$. Equivalently, the mean squared error along every unit direction u is the same, $\mathbb{E}[(u^\top\delta)^2] = \sigma^2$, so the rounding error has no preferred direction. The block Hadamard rotation of Section 3 spreads outlier channels over all coordinates, which brings the rounding error close to this state, and Appendix A.5 tests one consequence of it on Fast-WAM. For an isotropic residual, cyclicity of the trace and $(I-\Pi)^2 = I-\Pi$ give

$$\mathbb{E}\big[\delta^\top(I-\Pi)\,\widetilde{G}_\ell\,(I-\Pi)\delta\big] = \sigma^2\operatorname{tr}\big(\widetilde{G}_\ell(I-\Pi)\big) = \sigma^2\big(\operatorname{tr}\widetilde{G}_\ell - \operatorname{tr}V^\top\widetilde{G}_\ell V\big), \tag{12}$$

where the last step uses $\operatorname{tr}(\widetilde{G}_\ell VV^\top) = \operatorname{tr}(V^\top\widetilde{G}_\ell V)$.

Proposition 1 (Optimal protected subspace). *Over all $V \in \mathbb{R}^{d_\ell\times r}$ with orthonormal columns, Equation 12 reaches its minimum when the columns of V span the top-r eigenvectors of $\widetilde{G}_\ell$, and the minimum equals $\sigma^2\sum_{j>r}\lambda_j$.*

Proof. The term $\operatorname{tr}\widetilde{G}_\ell$ does not depend on V, so minimizing Equation 12 over V is the same as maximizing $\operatorname{tr}(V^\top\widetilde{G}_\ell V)$ over matrices with orthonormal columns. The Ky Fan maximum principle (Fan, 1949) states that for a symmetric matrix A with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$,

$$\max_{V^\top V = I_r} \operatorname{tr}\left(V^\top AV\right) = \sum_{j=1}^{r}\lambda_j,$$

attained when the columns of V span an invariant subspace of the r largest eigenvalues. The matrix $\widetilde{G}_\ell = S^\top G_\ell S$ with $S = \operatorname{diag}(\gamma)H$ is symmetric, so the principle with $A = \widetilde{G}_\ell$ gives the minimum distortion $\sigma^2(\operatorname{tr}\widetilde{G}_\ell - \sum_{j\leq r}\lambda_j) = \sigma^2\sum_{j>r}\lambda_j$, attained by the top-$r$ eigenspace. □

Without isotropy, the distortion $\operatorname{tr}\big((I-\Pi)\widetilde{G}_\ell(I-\Pi)\Sigma_\delta\big)$ depends on both $\widetilde{G}_\ell$ and Σ_δ, and the best subspace need not be the top eigenspace of $\widetilde{G}_\ell$.

A.4 The probe estimator: procedure, cost and accuracy

This appendix details the estimator of Equation 6. Throughout, $J_\ell \in \mathbb{R}^{m\times d_\ell}$ denotes the Jacobian of the action chunk $a \in \mathbb{R}^m$ with respect to one token–step activation of layer ℓ, with the index i suppressed.

Why probes. Reverse-mode differentiation computes, in one backward pass, the gradient of one scalar with respect to every intermediate tensor. For the vector output a, one backward pass therefore computes the vector–Jacobian product $J_\ell^\top w$ for one chosen weight vector $w \in \mathbb{R}^m$, by differentiating the scalar $\langle a, w\rangle$. The exact Gram matrix runs w through the m unit vectors $e_1, \ldots, e_m$,

$$J_\ell^\top J_\ell = \sum_{k=1}^{m}\left(J_\ell^\top e_k\right)\left(J_\ell^\top e_k\right)^\top,$$

which costs m backward passes per input, m=448 on RoboTwin. The probe estimator replaces the sweep over unit vectors with random vectors $u \sim \mathcal{N}(0, I_m)$. Since $\mathbb{E}_u[uu^\top] = I_m$,

$$\mathbb{E}_u\big[(J_\ell^\top u)(J_\ell^\top u)^\top\big] = J_\ell^\top\,\mathbb{E}_u[uu^\top]\,J_\ell = J_\ell^\top J_\ell,$$

so each probe gives an unbiased rank-one estimate of the exact sum, and the P-probe average of Equation 6 is unbiased for every $P \geq 1$. The number of probes controls only the variance. The construction is the Hutchinson trace estimator (Hutchinson, 1989) applied to a Gram matrix instead of a trace. G_ℓ is itself an average over inputs, so probe noise and input noise average down along the same sum, since N inputs with P probes each contribute NP rank-one terms for every token–step pair.

Procedure. For each calibration input (o_n, l_n), the estimator runs three steps.

1. Encode the observation and the instruction once into the cached conditioning $\mathbf{z}$, and fix the initial noise $a^{(0)}$ of the sampler to one draw, so that the action chunk a is a deterministic function of the activations.
2. Run the T Euler steps of Equation 1 with gradients enabled, and record the input activation $x_\ell^{(i)}$ of every quantized linear layer at every token–step pair i. Each recorded activation is a separate node of the autograd graph, for the reason the Attribution paragraph below gives.
3. For $p = 1, \ldots, P$, draw $u_{np} \sim \mathcal{N}(0, I_m)$ and backpropagate the scalar $\langle a, u_{np} \rangle$ through the recorded graph. Read the gradient $g_{\ell,np}^{(i)} = J_\ell^{(i)\top} u_{np}$ at every recorded activation, and add $g_{\ell,np}^{(i)} g_{\ell,np}^{(i)\top}$ to a $d_\ell \times d_\ell$ accumulator A_ℓ.

The graph is kept between probes, so one forward pass serves all P backward passes. After the last input, $\widehat{G}_\ell = A_\ell/(NP)$, which equals Equation 6.

Cost. Per input, the estimator needs one forward pass and P=12 backward passes, against m=448 backward passes for exact Jacobians. Each backward pass returns the vector–Jacobian products of every layer, so the cost does not grow with the number of layers. Each layer stores one $d_\ell \times d_\ell$ accumulator, at most 4096×4096 on Fast-WAM, independent of N. With N=10,919 inputs, the whole calibration costs $NP \approx 1.3 \times 10^5$ backward passes and runs once offline. Smoothing and rotation enter afterwards through Equation 7, so changing either transform requires no new calibration. Q-WAM reads three quantities from the estimate, and the next three paragraphs give the guarantee for each.

Trace. $\operatorname{tr} \widehat{G}_\ell = \frac{1}{NP} \sum_{n,p} \|J_\ell(o_n)^\top u_{np}\|^2$ is Hutchinson's estimator of $\operatorname{tr}(J_\ell^\top J_\ell)$ (Hutchinson, 1989). For a fixed symmetric positive semidefinite A and Gaussian probes, the single-probe estimate $u^\top A u$ has variance $2\|A\|_F^2$, so NP probes give a relative standard deviation of $\sqrt{2/NP}\, \|A\|_F / \operatorname{tr} A$, and $O(\varepsilon^{-2} \log(1/\eta))$ probes suffice for relative error ε at confidence $1 - \eta$ (Avron & Toledo, 2011; Roosta-Khorasani & Ascher, 2015). A sharply decaying spectrum, which Figure 3b shows, puts $\|A\|_F / \operatorname{tr} A$ near its maximum of one, the worst case for this bound. Equation 10 uses the trace only to compare experts whose masses differ by one to four orders of magnitude, so the estimator's error is far below the margin it has to resolve, and variance reduction (Meyer et al., 2021) is unnecessary.

Subspace. Collect the P probes of frame n into $U_n \in \mathbb{R}^{m \times P}$ and stack the N Jacobians into $\mathbf{J} = [J_\ell(o_1); \ldots; J_\ell(o_N)] \in \mathbb{R}^{Nm \times d_\ell}$, so that $\mathbf{J}^\top \mathbf{J}/N$ is the empirical G_ℓ. With the block-diagonal test matrix $\Omega = \operatorname{blkdiag}(U_1, \ldots, U_N) \in \mathbb{R}^{Nm \times NP}$,

$$NP\, \widehat{G}_\ell = \sum_n J_\ell(o_n)^\top U_n U_n^\top J_\ell(o_n) = \mathbf{J}^\top \Omega\, \Omega^\top \mathbf{J},$$

so $\widehat{G}_\ell$ is, up to scale, the Gram matrix of the sketch $\mathbf{J}^\top \Omega$. This sketch is the randomized range finder (Halko et al., 2011) applied to $\mathbf{J}^\top$ with NP test vectors. The range finder recovers the dominant rank-r singular subspace once the number of test vectors exceeds r by a small oversampling margin, with an error set by the trailing singular values. The count that has to exceed r is therefore $NP = 10{,}919 \times 12 \approx 1.3 \times 10^5$, against $d_\ell \approx 10^3$, and not P alone; P sets only how much of each frame's Jacobian is seen. The guarantee of the range finder (Halko et al., 2011) is stated for a dense Gaussian test matrix and ours is block-diagonal, so we also rely on the unbiasedness of $\widehat{G}_\ell$ and on $NP \gg d_\ell$. Choosing $P < r$ per frame allocates a fixed backward-pass budget across frames rather than within one, which is the right allocation because G_ℓ averages over frames. Exact per-frame

Jacobians would cost $m/P \approx 37\times$ more per frame for a per-frame precision that the average then dilutes.

Basis. Gaussian probes have a rotationally invariant law, so the estimator's error is not tied to any coordinate system of action space. Rademacher probes have lower variance for the trace and are the usual choice when only a trace is wanted (Avron & Toledo, 2011), but their law singles out the coordinate axes, which is the wrong trade when the estimate is consumed as an eigenbasis.

Attribution. One implementation detail matters. When several layers consume the same tensor, as the q, k and v projections of one attention block do, the gradient stored on that tensor is the sum over all its consumers, and reading it would credit each layer with the others' sensitivity. We therefore clone each layer's input so that it is a distinct node in the autograd graph, and read $J_\ell^\top u$ from the clone. An earlier version of our AOGs omitted this and was recomputed; the shipped AOGs use per-consumer attribution.

A.5 Checking the isotropic noise model

Proposition 1 and the action mass of Equation 10 assume that the rotated 4-bit residual is isotropic, with no preferred direction. An isotropic residual places the fraction r/d_ℓ of its expected squared norm $\mathbb{E}\|\delta\|^2$ inside any r-dimensional subspace. We test this consequence on twelve action-expert layers of Fast-WAM, the ten linear layers of the first block and the query and key projections of the second, and report medians. The residual of Q_4 places 3.1% of its squared norm inside $\mathrm{span}(V)$, which matches the chance rate r/d_ℓ and the share inside a random subspace of the same rank. The same 3.1% of the squared norm carries 97% of the action damage $\delta^\top \widetilde{G}_\ell \delta$, against 2.9% for the random subspace, and deflation removes it at the output up to a second-order product of rounding errors. Removing the protected component also shrinks the dynamic range that sets ς in Equation 2, by a median factor of 0.77 against 0.99 for the random subspace, so the 4-bit residual contracts in every direction.

B Implementation Details

B.1 Quantization settings and kernels

All quantizers are symmetric and round to nearest, with one scale for each group of consecutive input channels. For INT4 we use a group size of 32 for both weights and activations. Each weight group stores 4-bit codes in $[-8, 7]$, and each activation group is quantized at runtime, per token, with a scale equal to its absolute maximum divided by 7. On Blackwell GPUs we use NVFP4 instead, which stores E2M1 values with an FP8 (E4M3) scale for every group of 16 elements, for both weights and activations; the weights carry an additional FP32 scale per tensor.

Before the rotation, we apply SmoothQuant smoothing with the per-channel factor

$$s_j = \frac{\max|X_j|^\alpha}{\max|W_{:,j}|^{1-\alpha}}, \tag{13}$$

where $\max|X_j|$ is the absolute maximum of input channel j over the calibration inputs and $\max|W_{:,j}|$ is the absolute maximum of the matching weight column. We divide the activation by s and multiply the weight by s, so the layer output is unchanged before quantization. In the RoboTwin experiments we use a smoothing factor of $\alpha = 0.5$ for every layer of all three models. The activation maxima are collected on 50 RoboTwin episodes, one per task. The block Hadamard rotation uses the largest power of two that divides the input dimension, up to 1024, and the protected subspace has rank $r = 32$ with its branch kept in 16-bit precision.

The action-block latency reported in Section 5 is measured with 4-bit kernels written in Triton. A quantized layer runs two of them. The first applies the smoothing, the block Hadamard rotation, the projection onto the protected subspace and the activation quantization, and the second multiplies the packed 4-bit weights, adding the protected branch and the bias in its epilogue. The weights stay packed at 4 bits throughout and are the exported codes of the evaluated checkpoint, and the activations are quantized by the same rule as in evaluation; the fused implementation differs from it

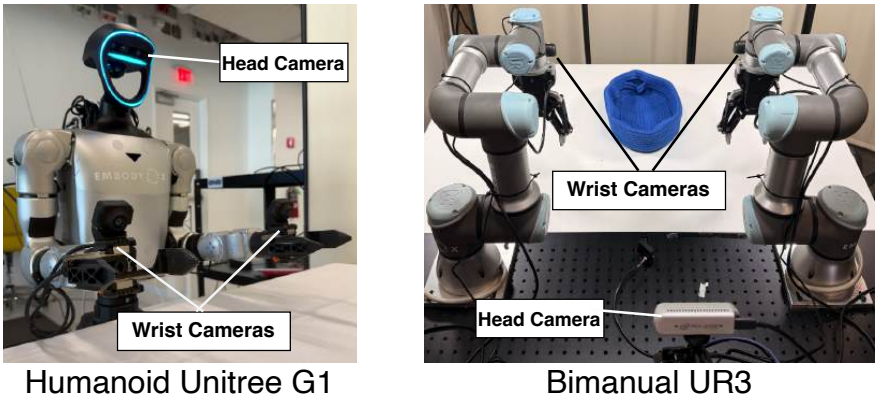

Figure 7: Real-world experiment setup.

only in the order in which it accumulates in floating point. We time the quantized layers of one action chunk since the ASP has been applied to that expert for the corresponding models. The precisions are replayed as CUDA graphs with the same layer layout. Each reported value is the median over repeated runs on one L40S GPU.

B.2 Cost of the 16-bit branch

The 16-bit branch of ASP adds little to the 4-bit model. Figure 6 compares the memory of ImageWAM's quantized blocks: 9.11 GB in bf16, 2.64 GB for the 4-bit model without the branch (smoothing, rotation and group-wise W4A4), and 2.69 GB for Q-WAM. The branch therefore adds 0.05 GB, 1.9% of the 4-bit model, while the model as a whole stays 3.4× smaller than bf16. Across the three models, the branch adds 0.05–0.16 GB and 0.08–0.26 bits per weight (Table 3). Its computation is small as well: both paths are dense matrix multiplications, and with the branch included the quantized action-token layers still run 1.06–1.56× faster than bf16 (Figure 5).

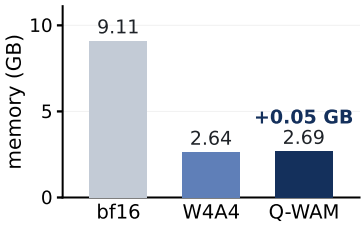

Figure 6: ImageWAM's memory comparison for the ASP overhead.

B.3 Real-world setup

Figure 7 shows the two embodiments of the real-world experiments. The first is a Unitree G1 humanoid with parallel-jaw grippers, and the second is a bimanual UR3 robot. Each embodiment carries four cameras. Table 2 lists the tasks run on each.

C Supplementary Experiments

C.1 SVDQuant as the weight quantizer

SVDQuant protects weights rather than activations. It keeps a rank-32 16-bit branch built from the leading singular vectors of the smoothed weight and quantizes the remainder, which works well on image diffusion models. Table 5 tests whether this weight protection adds to Q-WAM. ASP is held fixed on the activation side, and the weights are quantized either group-wise to INT4, as in Q-WAM, or with SVDQuant's decomposition. The SVDQuant branch costs 0.25 to 0.33 additional BPW and 0.14 to 0.24 GB of memory, but it does not improve the success rate. The average changes by +0.21 points on Fast-WAM and +0.14 points on LingBot-VA, both smaller than the variation between calibration sets (Appendix C.3), and it falls by 2.61 points on ImageWAM. SVDQuant selects the protected directions by the singular values of the weight, a criterion that ignores how the action depends on each layer. In a WAM, this dependence decides which quantization errors

Table 5: Q-WAM with SVDQuant as the weight quantizer. ASP is held fixed on the activation side and only the weight side varies, between group-wise INT4 and SVDQuant's low-rank decomposition.

model	activation	weight	BPW	clean	randomized	Avg.	Mem (GB)
Fast-WAM	ASP	group INT4	**4.62**	91.62	89.94	90.78	**3.44**
	ASP	SVDQuant	4.95	91.44	90.54	90.99	3.68
ImageWAM	ASP	group INT4	**4.58**	93.00	92.94	92.97	**2.69**
	ASP	SVDQuant	4.83	90.56	90.16	90.36	2.83
LingBot-VA	ASP	group INT4	**4.76**	90.20	88.92	89.56	**3.27**
	ASP	SVDQuant	5.03	90.52	88.88	89.70	3.43

matter. Figure 3a shows that per-layer reconstruction error does not predict the damage to the action, whereas the AOG does. Therefore, the weight directions that SVDQuant keeps in 16 bits are not the ones the action depends on, and group-wise INT4 is the cheaper choice for the weights of Q-WAM.

C.2 Where the action mass sits within each expert

Section 4.3 scores whole experts by their action mass μ_E (Equation 10). Figure 8 resolves the same quantity to single layers on Fast-WAM. Each cell is the trace $\mathrm{tr}(\widetilde{G}_\ell)$ of one linear layer as a share of the model's total, arranged by layer type and block, and both experts share one color scale. The action expert holds 89.3% of the total. Within it the mass concentrates in the cross-attention output projections, which carry 57.2% of the expert's mass, and cross-attention as a whole carries 82.0% against 2.0% for self-attention. Across the 300 action layers the trace spans more than six orders of magnitude. The video expert holds the remaining 10.7% and spreads it more evenly, led by the second feed-forward projection at 32.1% and the cross-attention output projection at 25.5%. Eight layers of the video expert's last block carry exactly zero mass, because no later block reads their outputs; the action expert reads that block only through its self-attention key and value projections, which are the two layers of the block that carry mass.

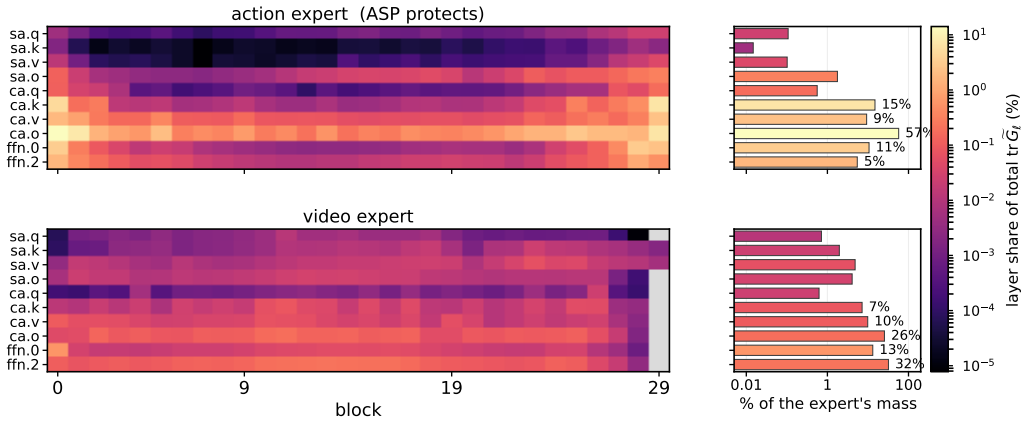

Figure 8: Share of Fast-WAM's action mass carried by each linear layer, for the action expert (top) and the video expert (bottom) on one logarithmic color scale. Rows are layer types and columns are blocks. The bars on the right give each layer type's share of its own expert's mass. Gray cells have exactly zero mass.

C.3 Sensitivity to the calibration set

The AOG and the protected subspaces are estimated from 50 calibration episodes, one per RoboTwin task. Table 6 measures how much the result depends on which episodes are drawn. On ImageWAM, we draw three further sets of 50 episodes with random seeds 7, 13 and 21, in addition to the original

set, which was drawn with seed 42. For each of the four sets, we recompute the AOG and the rank-32 protected subspaces, export a new checkpoint, and evaluate it on all 50 tasks with 100 episodes per task under both the clean and the randomized conditions.

The four checkpoints reach an average success rate of 92.41% with a standard deviation of 0.49 points. The largest gap between two calibration sets is 1.12 points, between seeds 7 and 13. The protected subspaces also barely change between sets. Any two sets share 97–99% of their rank-32 subspace, measured as $\|V_1^\top V_2\|_F^2/r$ and averaged over layers, while a random subspace of the same rank shares about 2%. The protected subspace, and with it the success rate, is therefore insensitive to the choice of calibration episodes.

Table 6: Sensitivity of ImageWAM to the calibration set. Each row recomputes the AOG, the protected subspaces and the checkpoint from a different set of calibration episodes, one per task, and evaluates it on the full RoboTwin suite.

Calibration seed	Clean	Randomized	Avg.
42 (original set)	93.08	92.28	92.68
7	92.62	93.08	92.85
13	91.48	91.98	91.73
21	92.34	92.44	92.39
Mean $\pm$ std.	92.38 ± 0.67	92.44 ± 0.46	92.41 ± 0.49

C.4 Per-task RoboTwin results

RoboTwin 2.0 has 50 tasks. Tables 7, 8 and 9 extend Table 1 task by task for Fast-WAM, ImageWAM and LingBot-VA. They report the success rate of every method on each task under the clean and the domain-randomized conditions.

Table 7: Per-task success rates (%) of Fast-WAM on RoboTwin 2.0 for every method in Table 1, under the clean (Clean) and domain-randomized (Rand.) conditions. The best quantized average is shown in bold.

	bf16		SmoothQuant		ViDiT-Q		SVDQuant		**Q-WAM**	
Task	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.
adjust_bottle	100	100	47	21	100	100	100	100	100	100
beat_block_hammer	97	98	18	10	94	94	98	94	100	96
blocks_ranking_rgb	100	100	1	0	96	97	100	99	100	100
blocks_ranking_size	95	93	1	0	83	90	89	91	90	92
click_alarmclock	99	100	96	92	100	100	100	100	100	100
click_bell	100	100	100	99	100	100	100	100	100	100
dump_bin_bigbin	98	96	35	20	85	87	88	87	96	98
grab_roller	100	100	43	25	100	100	100	100	100	100
handover_block	94	80	0	0	9	16	79	69	94	74
handover_mic	99	100	0	0	72	67	92	91	100	97
hanging_mug	57	58	5	1	44	46	51	49	64	54
lift_pot	99	100	21	6	90	95	98	92	100	99
move_can_pot	91	91	4	0	79	70	94	92	85	88
move_pillbottle_pad	99	99	2	6	84	90	93	97	99	100
move_playingcard_away	100	100	22	12	97	99	100	100	100	100
move_stapler_pad	81	63	0	0	54	35	72	59	72	61
open_laptop	98	100	63	53	98	97	98	98	98	99
open_microwave	80	47	0	1	60	42	56	37	70	33
pick_diverse_bottles	83	86	7	2	82	78	91	90	84	83
pick_dual_bottles	100	96	10	3	95	94	99	98	100	98
place_a2b_left	95	92	22	10	91	91	93	92	96	96
place_a2b_right	97	94	26	15	92	91	96	95	97	88
place_bread_basket	93	94	28	22	90	84	90	92	94	95
place_bread_skillet	93	87	9	1	86	80	95	90	93	94
place_burger_fries	94	98	19	3	90	96	93	95	97	96
place_can_basket	66	65	1	2	62	56	70	66	65	70
place_cans_plasticbox	98	99	1	0	93	93	95	97	100	97
place_container_plate	96	94	38	33	95	97	99	97	98	100
place_dual_shoes	92	89	0	0	93	82	86	88	88	91
place_empty_cup	100	100	59	35	100	99	99	100	100	100
place_fan	95	96	0	0	69	74	76	69	95	94
place_mouse_pad	88	89	6	3	74	75	82	81	84	83
place_object_basket	84	87	9	6	89	89	86	79	80	82
place_object_scale	88	93	7	1	86	85	88	89	87	92
place_object_stand	88	91	16	14	96	93	89	91	90	95
place_phone_stand	98	98	9	7	93	94	96	98	99	98
place_shoe	96	99	20	10	86	96	95	97	95	97
press_stapler	92	97	96	87	96	95	98	97	92	93
put_bottles_dustbin	91	90	0	0	29	33	78	73	88	82
put_object_cabinet	90	88	0	0	34	28	85	88	89	86
rotate_qrcode	91	91	0	2	89	77	90	85	91	86
scan_object	91	89	39	25	81	77	84	85	92	91
shake_bottle	100	100	90	82	100	100	100	100	100	100
shake_bottle_horizontally	100	100	91	84	99	100	100	100	100	100
stack_blocks_three	96	95	0	0	67	80	94	91	95	96
stack_blocks_two	100	100	3	2	100	97	100	100	100	99
stack_bowls_three	85	90	0	0	78	66	76	66	81	83
stack_bowls_two	95	97	2	1	92	89	96	94	94	95
stamp_seal	90	94	12	7	63	55	72	71	89	86
turn_switch	62	64	21	21	65	55	57	60	60	60
Average	92.28	91.34	21.98	16.48	82.00	80.48	89.12	87.38	**91.62**	**89.94**

Table 8: Per-task success rates (%) of ImageWAM on RoboTwin 2.0 for every method in Table 1, under the clean (Clean) and domain-randomized (Rand.) conditions. The best quantized average is shown in bold.

	bf16		SmoothQuant		ViDiT-Q		SVDQuant		**Q-WAM**	
Task	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.
adjust_bottle	100	100	96	97	99	99	100	100	100	100
beat_block_hammer	100	98	42	27	91	97	95	97	98	92
blocks_ranking_rgb	96	99	89	91	73	82	97	98	99	99
blocks_ranking_size	92	97	67	71	60	82	95	93	93	96
click_alarmclock	100	100	100	100	100	99	100	99	99	100
click_bell	100	100	100	100	100	100	99	100	100	100
dump_bin_bigbin	96	93	67	60	94	82	88	86	98	88
grab_roller	100	100	100	100	100	100	100	100	100	100
handover_block	94	96	17	8	97	91	64	64	96	95
handover_mic	100	99	81	85	56	69	96	97	98	98
hanging_mug	74	77	40	29	60	67	56	70	75	76
lift_pot	100	100	35	37	83	89	95	90	100	100
move_can_pot	96	98	20	28	94	98	83	75	96	100
move_pillbottle_pad	99	100	20	25	92	89	81	85	97	99
move_playingcard_away	100	99	92	97	97	96	97	97	100	99
move_stapler_pad	66	67	16	9	36	44	42	34	51	65
open_laptop	97	98	96	96	95	98	95	97	98	99
open_microwave	97	94	27	29	13	14	6	22	82	73
pick_diverse_bottles	87	89	47	43	89	83	89	83	86	89
pick_dual_bottles	99	96	39	41	88	84	95	93	100	98
place_a2b_left	95	96	76	85	87	92	93	94	95	98
place_a2b_right	99	96	89	85	91	96	95	93	99	95
place_bread_basket	94	96	67	61	88	86	89	90	97	93
place_bread_skillet	91	90	59	55	76	86	91	86	93	94
place_burger_fries	100	99	78	78	91	91	91	90	100	99
place_can_basket	77	62	28	20	66	58	56	59	80	71
place_cans_plasticbox	100	97	48	38	80	87	84	75	98	97
place_container_plate	99	97	85	88	92	97	95	95	98	99
place_dual_shoes	77	86	28	40	25	33	65	69	71	83
place_empty_cup	100	100	84	87	100	100	99	100	100	100
place_fan	95	96	26	34	54	65	69	68	93	95
place_mouse_pad	87	93	29	47	56	58	64	61	85	87
place_object_basket	78	82	68	63	80	75	82	81	82	87
place_object_scale	93	97	64	73	82	87	81	85	91	98
place_object_stand	100	98	75	73	95	90	92	96	95	95
place_phone_stand	100	99	86	90	100	99	88	96	100	98
place_shoe	95	99	78	81	79	88	94	97	97	98
press_stapler	95	100	98	98	100	97	97	100	96	100
put_bottles_dustbin	95	92	11	3	67	59	89	66	98	87
put_object_cabinet	88	95	20	16	63	79	70	73	86	87
rotate_qrcode	86	86	53	47	80	83	85	79	90	86
scan_object	93	89	26	22	70	80	86	88	91	95
shake_bottle	100	100	93	97	100	100	99	99	100	100
shake_bottle_horizontally	100	100	93	98	100	100	100	99	100	100
stack_blocks_three	96	98	24	45	64	77	93	84	97	97
stack_blocks_two	100	100	85	74	86	81	99	98	100	100
stack_bowls_three	70	82	35	38	51	70	64	74	84	79
stack_bowls_two	92	97	66	72	82	88	90	93	95	97
stamp_seal	78	83	21	14	41	56	69	68	92	90
turn_switch	75	80	68	73	75	83	75	75	81	76
Average	92.82	93.70	59.04	59.36	78.76	82.08	84.34	84.22	**93.00**	**92.94**

Table 9: Per-task success rates (%) of LingBot-VA on RoboTwin 2.0 for every method in Table 1, under the clean (Clean) and domain-randomized (Rand.) conditions. The best quantized average is shown in bold.

	bf16		SmoothQuant		ViDiT-Q		SVDQuant		**Q-WAM**	
Task	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.
adjust_bottle	96	96	96	90	95	95	100	96	98	94
beat_block_hammer	94	100	90	86	90	83	98	94	100	92
blocks_ranking_rgb	96	94	82	58	76	65	88	88	94	96
blocks_ranking_size	96	82	66	46	81	53	86	62	100	80
click_alarmclock	100	100	100	98	100	100	100	100	100	100
click_bell	100	100	100	100	100	98	100	100	100	100
dump_bin_bigbin	92	94	92	78	90	92	88	98	98	94
grab_roller	100	100	100	100	97	97	100	100	100	100
handover_block	100	90	58	36	56	44	88	80	100	90
handover_mic	94	98	82	64	71	66	74	84	82	90
hanging_mug	18	20	18	16	14	10	22	40	18	34
lift_pot	100	100	98	92	97	92	98	88	100	100
move_can_pot	94	94	82	86	87	74	82	72	96	82
move_pillbottle_pad	100	100	92	82	96	89	96	86	100	100
move_playingcard_away	100	98	100	90	95	91	100	96	98	100
move_stapler_pad	68	70	48	28	40	21	50	42	62	66
open_laptop	100	96	96	88	98	94	94	92	96	94
open_microwave	46	90	20	34	26	71	22	48	28	76
pick_diverse_bottles	94	88	88	56	87	56	80	86	96	88
pick_dual_bottles	100	100	94	74	99	72	98	90	100	96
place_a2b_left	98	90	94	86	89	83	94	84	98	90
place_a2b_right	86	92	90	82	89	76	94	94	100	92
place_bread_basket	94	96	86	84	84	83	84	92	96	92
place_bread_skillet	98	90	90	78	83	74	94	86	90	90
place_burger_fries	98	98	94	78	80	73	98	98	94	100
place_can_basket	82	84	72	50	68	71	74	58	86	72
place_cans_plasticbox	100	98	84	68	96	81	92	88	100	96
place_container_plate	100	98	100	92	96	96	94	90	100	98
place_dual_shoes	94	88	58	50	41	34	74	78	92	86
place_empty_cup	100	100	100	100	99	100	100	100	100	100
place_fan	90	88	72	52	70	62	70	72	100	84
place_mouse_pad	88	92	72	56	76	62	86	82	96	92
place_object_basket	90	76	80	60	84	66	82	86	84	78
place_object_scale	100	96	86	74	87	82	100	88	100	90
place_object_stand	98	92	92	86	98	83	96	80	100	80
place_phone_stand	98	94	96	86	87	86	98	90	98	94
place_shoe	94	98	88	84	95	85	92	90	98	98
press_stapler	78	84	94	96	97	93	100	96	84	90
put_bottles_dustbin	74	74	16	28	26	39	34	50	76	72
put_object_cabinet	92	76	52	54	54	45	84	58	74	80
rotate_qrcode	92	90	90	76	84	82	94	80	96	90
scan_object	96	86	72	74	71	73	96	80	86	92
shake_bottle	98	100	100	96	99	99	96	96	100	100
shake_bottle_horizontally	100	100	98	94	100	99	100	100	100	100
stack_blocks_three	100	96	84	72	91	67	100	88	100	94
stack_blocks_two	100	100	96	90	100	92	98	98	98	100
stack_bowls_three	68	82	64	54	62	56	70	80	66	80
stack_bowls_two	96	92	92	82	89	90	92	92	96	88
stamp_seal	98	90	84	60	84	47	82	68	98	94
turn_switch	54	60	86	68	67	60	60	72	38	62
Average	90.84	90.20	81.68	72.24	80.82	74.04	85.84	83.12	**90.20**	**88.92**