ACTTUNE: ACTION-AWARE PRECISION AND GPU OPERATING-POINT ADAPTATION FOR ENERGY-EFFICIENT VISION-LANGUAGE-ACTION INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) policies repeatedly invoke inference to control robots, making graphics processing unit (GPU) energy a recurring cost of task execution. Reducing energy per inference call, however, may not reduce energy per successful task if numerical errors increase failures or slower inference prolongs execution. We therefore target GPU energy per successful task while preserving task success and keeping the inference-latency increase within 10%. Our approach builds on two observations: quantization sensitivity varies across action classes, model layers, and weights versus activations; and numerical precision changes the workload, shifting favorable GPU operating points. We introduce ActTune, an action-aware framework that connects layer-wise precision allocation with workload-dependent GPU operating-point selection over requested frequency–power-cap pairs. A lightweight decision tree learns its splits and leaf precision configurations directly from configuration action errors, then selects precision before each policy call. The controller forecasts the next workload and applies the selected GPU operating point asynchronously using a lookup table calibrated under a latency budget. A shared resident quantized weight bank enables configuration switching without weight reconstruction or additional policy evaluations. On LIBERO, a benchmark for lifelong robot learning, ActTune improves mean task success by up to 2.3% relative to state of the art. Relative to the original BF16 implementations, it delivers up to $2.02\times$ faster inference and, with GPU operating-point adaptation, reduces energy per successful task by up to 76.8%.

1 INTRODUCTION

VLA models translate visual observations and language instructions into robot actions (Kim et al., 2024). For battery-powered robots, repeated onboard inference consumes limited battery energy and can shorten operation between charges. Reducing inference energy is therefore important for extending robot endurance. Dynamic voltage and frequency scaling (DVFS) offers a direct way to pursue this goal by lowering power consumption (Tang et al., 2019). However, lower frequencies prolong inference, which can offset energy savings and delay action updates. The need for timely responses therefore limits how aggressively frequency can be reduced.

Numerical precision and GPU operating-point control offer complementary ways to reduce this cost. Quantization lowers weight and activation precision, changing computation and memory traffic, but the resulting numerical errors can also perturb predicted actions. We define a GPU operating point as a requested frequency–power-cap pair. Adjusting these settings trades execution speed against power consumption. Lower frequencies may reduce power, yet longer inference can offset these savings and delay the robot's response to new observations. Crucially, these two controls are coupled: precision determines the workload on which GPU operating-point control operates. An energy-efficient operating point for one precision configuration may therefore be less effective for another. Exploiting both controls requires connecting the precision needs of robot actions to the hardware characteristics of the resulting inference workload.

Existing work provides useful foundations for this connection. VLA quantization methods use action or task sensitivity to allocate precision (Xu et al., 2026; Akbari et al., 2026; Ranjan & Savakis, 2026), while temporal-dynamic-aware quantization for VLA models (DyQ-VLA) and motion-aware

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.



(a) Coarse approach (b) Final alignment (c) Object transport

Figure 1: ActTune intuition: approach tolerates errors, alignment requires accuracy, and transport precision depends on path and grip stability.

vector quantization (MotionVQ) adapt precision during execution using motion signals (Zheng et al., 2026; Song et al., 2026). However, compression objectives such as model size and bit-operation budgets do not directly capture GPU energy during task execution (Ranjan & Savakis, 2026). Other work studies energy efficiency through quantized robotic policies or device–edge execution placement (Park et al., 2025; Zhou et al., 2026), and PiQi, a framework for partially quantized inference, combines partial quantization, processor placement, and DVFS for general deep neural network (DNN) inference (Aghapour et al., 2024). Building on these directions, we investigate how action-aware precision allocation and workload-dependent GPU operating-point selection can work together to reduce energy per successful robot task.

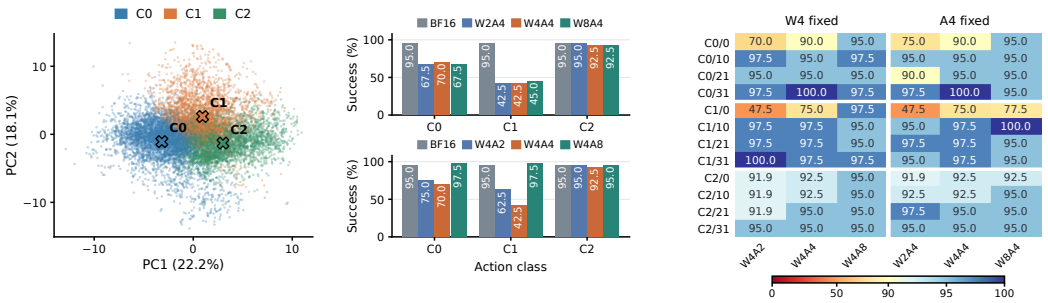
Our approach is motivated by two observations. First, quantization sensitivity depends on both the action being performed and where numerical error is introduced in the model. For example, coarse approach motion may tolerate small deviations, whereas final alignment may require greater accuracy (Figure 1). Using a fixed set of action classes learned from training actions, short quantized interventions reveal sensitivity differences across action classes, Transformer blocks, and weight versus activation precision (Figure 2(b–c)). Second, hardware measurements show that different numerical precisions can favor different GPU operating points (Figure 3). Together, these observations suggest a two-stage adaptation strategy: use action sensitivity to guide precision allocation, then use the resulting workload to guide GPU operating-point selection.

We introduce **ActTune**, an action-aware framework that implements this strategy. Before each action chunk is generated, a lightweight decision tree uses current proprioception and the preceding predicted action chunk to select a layer-wise weight and activation precision configuration. Offline calibration learns the tree splits and leaf configurations directly from action error against the BF16 reference policy under precision and storage budgets. A shared resident quantized weight bank supports configuration switching without reconstructing weights or storing a complete model copy for every leaf. This design enables action-aware precision selection with only one policy inference call per action chunk.

ActTune profiles frequency and power-cap settings on replayed training workloads with the precision selector fixed. It selects the candidate table with the lowest estimated energy while limiting the mean inference-latency increase to 10% relative to the same precision selector at the fixed reference GPU operating point, subject to action equality and memory checks. After each call, the controller forecasts the next workload and applies the selected GPU operating point asynchronously. Precision selection uses the current observation. Because matching actions on replayed inputs does not guarantee unchanged closed-loop behavior, the final assessment must consider complete episodes. We therefore evaluate closed-loop task success and energy per successful task, and check the 10% budget using the measured inference latency. Our code is open source.[1] Our contributions are:

- **Empirical characterization.** We firstly characterize how quantization sensitivity varies across action classes, Transformer blocks, and weight versus activation precision, and show that favorable GPU operating points depend on numerical precision.
- **Action-aware precision and GPU operating-point adaptation.** We propose ActTune, which connects action-conditioned layer-wise precision allocation with workload-dependent GPU operating-point selection. A lightweight decision tree and a shared quantized weight bank enable online adaptation with one policy call per action chunk.
- **Quantitative evaluation.** In the evaluated LIBERO comparisons, ActTune improves mean task success by up to 2.3% relative to QVLA, an action-centric VLA quantization framework, and action-guided quantization (ActQuant). It also reduces static model storage by up to 29.3% relative to the compared general-purpose quantization baselines at W4A8/W4A4 (Table 1). Compared with the

[1] https://anonymous.4open.science/r/Actune-165E



(a) Action clusters (b) Action sensitivity (c) Layer sensitivity

Figure 2: Action grouping and precision sensitivity. PC1 and PC2 denote the first and second principal components. WnAm denotes n-bit weights and m-bit activations. Crosses mark cluster centroids; success rates average all four suites, with all action and layer precision batches complete.

original BF16 implementations, it achieves up to 2.02× inference speedup and up to 76.8% lower energy per successful task with GPU operating-point adaptation.

2 Motivation

We examine two empirical questions that motivate ActTune: (1) whether quantization sensitivity depends on the action and on where precision is reduced in the model, and (2) whether different precision configurations favor different GPU operating points. The first determines how precision should be allocated, while the second determines whether hardware settings should adapt to the resulting workload.

Action- and layer-dependent quantization sensitivity. We first group training action blocks into three fixed classes, C0–C2, using principal component analysis (PCA) followed by K-means (Figure 2(a)). These classes describe patterns in the action data rather than prescribed physical task stages. To measure quantization sensitivity, we insert an eight-action quantized intervention into matched BF16 trajectories and then return to BF16 control. We measure final task success rate using equal-weight averages over Goal, Object, Spatial, and Long. The matched BF16 reference achieves a mean task success rate of 95.0%. Figure 2(b) shows that the same precision configuration can have substantially different effects across action classes. At fixed A4, C1 achieves only 42.5–45.0% task success across the tested weight precisions, whereas C2 maintains 92.5–95.0%. For C1, changing activation precision has a larger effect than changing weight precision, and increasing weight precision alone does not resolve the poor A4 outcomes. Sensitivity is also nonuniform across Transformer layers (Figure 2(c)). For C1, applying W4A4 to layer 0 yields 75.0% task success, compared with 97.5% when applying the same precision to the other tested layers. Increasing layer-0 activation precision from A4 to A8 raises task success to 97.5%. These results show that precision requirements depend jointly on the action class, model location, and quantized component, motivating action-conditioned, layer-wise allocation of weight and activation precision.

Precision-dependent GPU energy–latency tradeoffs. We next test whether numerical precision changes the preferred GPU operating point. On an NVIDIA RTX A6000, we sweep requested GPU frequency at a fixed 300 W power cap and sweep power cap at a requested frequency of 1800 MHz (Figure 3). Measurements for each precision are normalized to its own 1800 MHz / 300 W reference, isolating the effect of hardware tuning from the energy differences introduced by quantization. At the kernel level, we benchmark resident INT4×INT4, INT8×INT8, and BF16×BF16 GEMM workloads. The measurements include the output epilogue but exclude input quantization, packing, and transfers. INT4 and INT8 reach their lowest measured GPU energy at different frequencies (Figure 3(a)), showing that the energy–latency tradeoff depends on numerical precision. An exploratory policy-level sweep shows the same tendency during VLA execution (Figure 3(b)). In this exploratory sweep, among measured operating points that increase total benchmark time by at most 1%, the lowest-energy points reduce GPU energy by 20.4%, 22.7%, and 21.0% for BF16, W4A8, and W4A4, respectively, relative to their corresponding reference operating points. These points use different frequency–power-cap combinations, motivating configuration-specific hardware profiling rather than a common operating point for all precision settings.

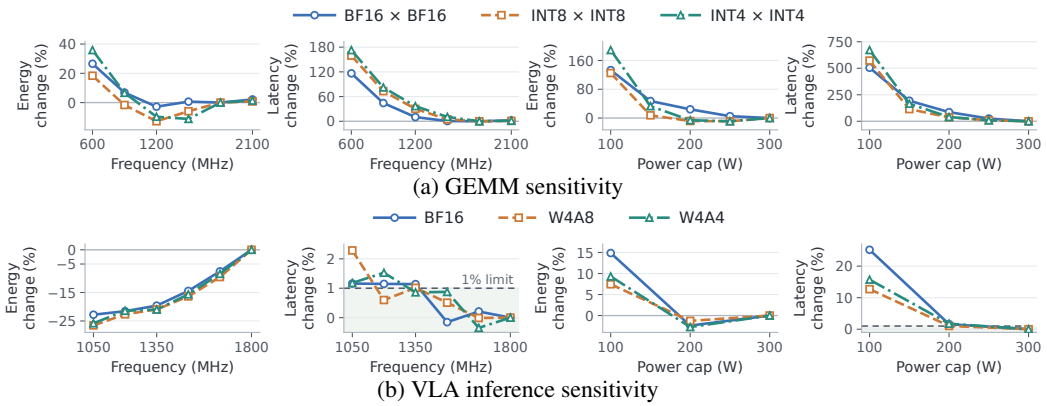


Figure 3: Energy and latency under frequency and power-cap tuning, relative to 1800 MHz / 300 W. Dashed lines mark the 1% total-time threshold used in this exploratory sweep.

The clusters above serve as diagnostics; ActTune learns decision regions from each configuration's action error (Section 3.2). The hardware sweep uses four matched episodes and one measurement per setting, so it provides design evidence rather than a precise estimate of task-level savings. These findings motivate action-aware precision allocation and workload-dependent GPU control.

3 The ActTune Design

3.1 Overview

ActTune adapts precision and GPU operating points to reduce energy per successful task while preserving task success and limiting the inference-latency increase from operating-point adaptation to 10%. Following Section 2, precision selection accounts for action-dependent quantization sensitivity, while GPU frequency and power cap depend on the resulting workload. The controller selects precision before each policy call and prepares GPU operating points asynchronously between calls by forecasting the next workload. At request t, the policy maps observation o_t, comprising images, a language instruction, and proprioception $s_t \in \mathbb{R}^{d_s}$, to an action chunk $A_t \in \mathbb{R}^{H\times 7}$. Each action contains six motion coordinates and a gripper command. A GPU operating point $u_t = (f_t, p_t)$ pairs the requested frequency f_t with power cap p_t. The selected precision configuration q_t and operating point u_t remain fixed throughout inference, including iterative action generation. The predicted chunk provides action-history features for the next request. Figure 4 shows the workflow. Offline calibration learns candidate decision trees from configuration errors and selects their complexity using validation loss and routing cost (Sections 3.2–3.3). Each leaf stores a precision configuration from the shared weight bank. With the tree fixed, GPU profiling selects operating points under a latency budget (Section 3.4). Online execution uses one policy call per request (Section 3.5).

3.2 Learning decision regions for precision selection

ActTune learns when to use each precision configuration from calibration errors. A binary tree partitions action context into decision regions, with one precision configuration stored at each leaf. Calibration learns the split conditions and configuration choices, while validation characterizes the tradeoff between the number of regions and routing cost. The method does not prescribe a fixed number of leaves. The tree forms $x_t = [\phi(A_{t-1}, m_{t-1}); s_t]$ from current proprioception and the preceding predicted chunk, then selects leaf $k_t = g(x_t) \in \{1, \ldots, K\}$ and configuration $q_t = q_{k_t}$. Here ϕ extracts motion and gripper features, m_{t-1} is the preceding gripper margin, $g(x_t)$ is the reached leaf, and q_k is its configuration (Appendix B.5). The selected number of leaves is K. These inputs are available before inference, so selection requires no additional policy call. Regions are learned from configuration losses, not predefined semantic action stages; the tree depth bounds the number of online comparisons.

3.3 Action-conditioned precision with shared resident weights

ActTune allocates backbone precision under average weight and activation budgets, weighted by parameter count and input element count, respectively. Individual matrices may use different pre-

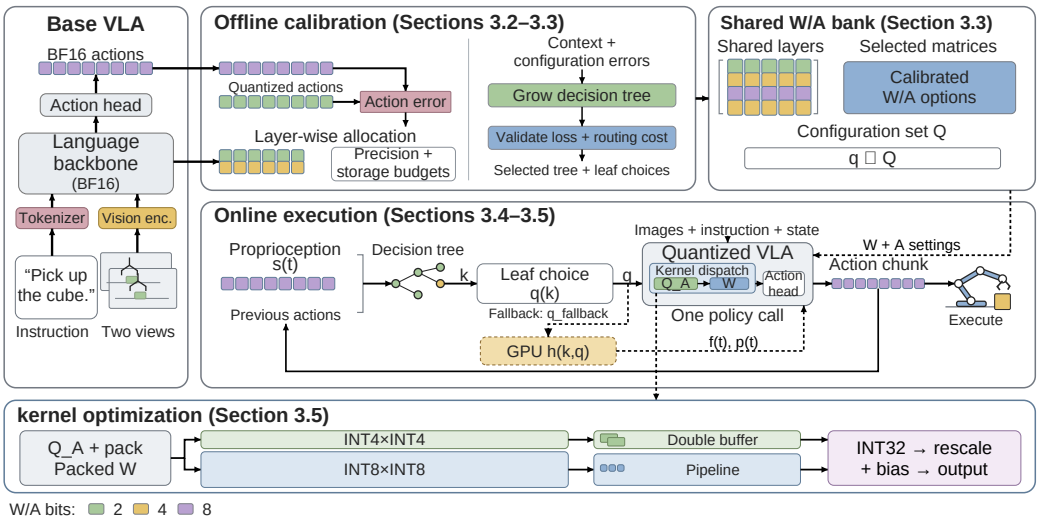


Figure 4: Overview of ActTune.

cisions within these budgets. A separate resident-storage bound counts all retained weight versions and quantization metadata, including inactive alternatives (Appendix B.3).

Shared configurations. Offline calibration selects switchable matrices and their precision options under the precision and resident-storage budgets. These options form a configuration set $\mathcal{Q}$; all other targeted matrices share one resident version. Each tree leaf selects a configuration, and multiple leaves may share it, so leaf count and configuration count need not match. Configurations specify resident weight versions and runtime activation precision. Calibration also determines a fallback configuration $q_{\text{fallback}} \in \mathcal{Q}$.

Sensitivity-guided bank construction. We calibrate layer alternatives against the BF16 checkpoint using activation-guided channel scaling based on activation-aware weight quantization (AWQ) (Lin et al., 2023) and output-row clipping to minimize local reconstruction error. Weighting these errors by the action Jacobian prioritizes matrices whose quantization affects action predictions more strongly. Matrices with low penalties receive fewer weight bits, freeing budget for upgrades with greater error reduction per added bit. We lower shared weight precision as needed to meet the resident-storage bound, prioritizing small error penalties per byte saved. We calibrate candidate weight/activation pairs and select a feasible bank using training action error (Appendix B.4). For each model–budget pair, one tree shares its splits and configuration indices across tasks; each index resolves to the bank associated with the task's checkpoint.

Learning the partition and configuration choices. For each calibration observation i, we evaluate every configuration in $\mathcal{Q}$ against the same BF16 reference. With postprocessed action chunks A_i^q and A_i^{ref}, the loss is:

$$\ell_i(q) = \frac{1}{6H} \sum_{h=1}^{H} \sum_{d=1}^{6} \left| A_{i,h,d}^{q} - A_{i,h,d}^{\text{ref}} \right| + \frac{\lambda_g}{H} \sum_{h=1}^{H} \mathbf{1}\left[A_{i,h,7}^{q} \neq A_{i,h,7}^{\text{ref}} \right]. \tag{1}$$

Here λ_g weights gripper disagreement. For N fitting contexts and each candidate leaf count K, we optimize action agreement:

$$\min_{g,\, \{q_k \in \mathcal{Q}\}} \sum_{i=1}^{N} w_i \, \ell_i\big(q_{g(x_i)}\big) \quad \text{subject to} \quad |\,\text{Leaves}(g)| = K. \tag{2}$$

Weights w_i sum to one and assign equal mass to suites, then trajectories within each suite, then valid contexts within each trajectory. Each leaf chooses the configuration with the lowest weighted fitting loss. We grow a tree by loss-reducing splits, then use bottom-up dynamic programming to obtain the lowest-fitting-loss subtree for each feasible leaf count (Appendix B.6).

Selecting complexity and freezing. We select tree size by comparing validation action loss and measured CPU routing cost across candidate leaf counts, including a single-leaf global selector.

Entire calibration trajectories are held out before fitting the bank and tree; validation leaves the fitted splits and leaf configurations unchanged. For a common leaf count across model–budget pairs, we compare their validation curves jointly. Section 4.3 reports the resulting experimental choice. We freeze the chosen tree before subsequent GPU profiling. Calibration uses the preceding fallback-policy chunk as history; deployment uses the selected configuration's output. The closed-loop evaluations in Section 4 assess this history shift and task success using the frozen tree.

3.4 Selecting GPU frequency and power cap

Precision changes computation and memory traffic, shifting favorable GPU operating points (Section 2). We therefore profile frequency and power cap jointly, with the weight bank and precision selector fixed. The experiments label this joint operating-point control policy "DVFS"; its controls are requested GPU frequency and power cap.

Offline profiling. For each supported region and configuration pair (k, q), we measure energy and latency on training observations at operating points $u = (f, p) \in \mathcal{U}$, relative to $u_{\mathrm{ref}} = (f_{\mathrm{ref}}, p_{\mathrm{ref}})$. Comparisons use the same memory frequency, observations, selector states, and random states for stochastic policies. Closed-loop measurements also include switching overhead. We retain operating points that preserve reference actions and pass memory checks. Uniform settings and a sweep over latency penalties define candidate lookup tables, each mapping a region–configuration pair to a GPU operating point. Using replay measurements, we select the table h^* with the lowest estimated GPU energy whose mean inference latency increases by at most 10% relative to the same precision selector at u_{ref}. Unsupported pairs use calibrated fallbacks. We freeze the table, verify actions and memory on disjoint trajectories, then evaluate closed-loop task success, energy per success, and inference latency on the complete benchmark. The 10% budget applies to mean complete prediction-call latency, measured against the same frozen precision selector at the fixed reference operating point u_{ref}. Figure 5 reports this comparison for all four model–precision settings. Energy per success is total episode GPU energy, including failures and gaps between calls, divided by successes.

Online use. After each inference, the controller predicts the next decision region and precision configuration. It looks up the corresponding GPU frequency and power cap in the frozen table and applies them asynchronously between calls. These settings remain fixed during the next inference.

3.5 Online execution and integer kernels

Each episode starts with empty action history and the fallback operating point. Before inference, the controller waits for any pending GPU update, traverses the frozen tree with x_t, and reads $q_t = q_{g(x_t)}$. Missing history or invalid input features use q_{fallback}. The policy produces $A_t = \pi_{q_t}(o_t)$ at the active operating point. After inference, the controller uses new action features and bounded proprioception extrapolation to forecast the next region and precision configuration. It applies the operating point from h^* asynchronously between calls. Forecast errors leave the prepared operating point active; precision uses the current observation. Episode measurements include switching and remaining waits. Appendix A.2 gives control and runtime details.

Executing the selected precision. ActTune executes low-bit linear layers without reconstructing dense weights. It uses INT4×INT4 or INT8×INT8 Tensor Core kernels with INT32 accumulation (NVIDIA, 2022). Quantization bit width determines represented values and resident weight storage; the kernel determines arithmetic operand width. Extending a code to a wider operand preserves its value and quantization precision.

Layers whose weights and activations both use at most four bits run with INT4 arithmetic, extending two-bit codes to signed INT4 as needed. Layers with eight-bit weights or activations use INT8 arithmetic, extending narrower codes inside the kernel. W2/W4 weights for A8 layers remain packed in device memory and are sign-extended into INT8 tiles during GEMM. Together, these paths cover all nine local weight/activation pairs without changing the calibrated codes. The INT4 kernel, implemented in NVIDIA's Compute Unified Device Architecture (CUDA) programming platform, uses packed operands, shape-dependent tiles, asynchronous copies, double-buffered shared memory, and fragment reuse. The INT8 Triton kernel uses coalesced weight layouts, tile-local unpacking, and software pipelining. Fusing activation quantization and GEMM epilogues for rescaling, bias addition, and output conversion reduces kernel launches and intermediate memory traffic. Policy execution includes these operations; the resident-GEMM measurements in Section 2 exclude input quantization, packing, and transfers. By shortening execution time, these kernel and runtime op-

Table 1: Comparison of quantization methods at W4A16, W4A8, and W4A4 on LIBERO for OpenVLA-OFT and $\pi_{0.5}$. Storage includes all persistent tensors. Parentheses show energy with GPU operating-point adaptation (DVFS). Bold marks the best value per model/precision group; energy is compared without DVFS.

Model	Setting	Method	Spatial	Object	Goal	Long	Mean↑	Δ	Model storage (GB)↓	Energy (J/success)↓
OpenVLA-OFT	BF16	–	97.6	98.4	97.8	94.4	97.1	–	15.42	2830
	W4A16	QVLA	96.0	**98.6**	97.4	**95.2**	96.8	−0.3	4.50	3579.1
		ActQuant	96.0	98.2	97.2	95.0	96.6	−0.5	**4.20**	3457.2
		Ours	**97.2**	98.0	**97.6**	95.0	**97.0**	**−0.1**	4.47	**3369**
	W4A8	SmoothQuant	95.6	97.2	96.8	92.4	95.5	−1.6	6.32	2057.5
		GPTQ	97.4	96.8	**98.2**	92.8	96.3	−0.8	6.01	1944.8
		AWQ	**98.4**	96.8	97.2	93.4	96.5	−0.6	5.91	1986.7
		Ours	97.4	**98.2**	97.6	**94.8**	**97.0**	**−0.1**	**4.47**	**1145.3** (843.1)
	W4A4	SmoothQuant	77.2	70.0	77.8	68.6	73.4	−23.7	6.32	2763.7
		GPTQ	91.4	94.4	92.2	87.8	91.5	−5.6	6.01	1453.9
		AWQ	94.2	90.4	72.4	16.8	68.5	−28.6	5.91	2788.2
		Ours	**96.8**	**97.6**	**96.8**	**90.8**	**95.5**	**−1.6**	**4.47**	**1135.4** (905.0)
$\pi_{0.5}$	BF16	–	98.4	98.4	97.6	93.6	97.0	–	7.47	2836
	W4A16	QVLA	98.0	97.2	96.4	91.8	95.9	−1.2	6.3	3468.5
		ActQuant	98.4	**99.4**	96.8	91.8	96.6	−0.4	6.20	3364.9
		Ours	**99.4**	**99.4**	**98.6**	**95.0**	**98.1**	**+1.1**	**3.42**	**3355**
	W4A8	SmoothQuant	98.8	99.4	98.2	**97.6**	**98.5**	**+1.5**	3.67	1548.8
		GPTQ	99.0	**99.6**	**98.8**	94.6	98.0	+1.0	3.57	1722.6
		AWQ	98.0	99.2	98.2	95.8	97.8	+0.8	3.53	1662.9
		Ours	**99.4**	98.6	98.0	96.2	98.1	+1.1	**3.42**	**825.1** (657.2)
	W4A4	SmoothQuant	93.6	98.8	71.8	58.0	80.6	−16.5	3.67	1902.5
		GPTQ	96.6	**99.6**	91.4	85.6	93.3	−3.7	3.57	**1341.7**
		AWQ	72.4	94.2	49.4	27.0	60.8	−36.3	3.53	2965.3
		Ours	**99.4**	**99.6**	**98.8**	**95.8**	**98.4**	**+1.4**	**3.42**	1527.2 (1221.8)

timizations also help reduce energy consumption, as shown by the energy-saving contributions in Figure 6b.

4 Experiments

4.1 Experimental setup

We evaluate quantized OpenVLA-OFT (Kim et al., 2025) and $\pi_{0.5}$ (Physical Intelligence et al., 2025) on the LIBERO Spatial, Object, Goal, and Long suites (Liu et al., 2023) (Table 1). Our hardware measurements use NVIDIA RTX A6000 GPUs. We compare ActTune with QVLA (Xu et al., 2026) and ActQuant (Akbari et al., 2026) at W4A16, and with SmoothQuant (Xiao et al., 2022), GPTQ (Frantar et al., 2022), and AWQ (Lin et al., 2023) at W4A8 and W4A4. We quantize eligible linear layers in both the vision encoder and language Transformer; other components retain their existing precision. We report task success, inference speedup, compressed-model storage, and GPU-board energy per successful episode. Appendix A.1 specifies the evaluation protocol and selector settings. Appendices A–C give hardware and controller settings, calibration details, and supplementary results.

We use the average-precision budgets in Section 3.3. Two switchable matrices, each with two precision options, give $\mathcal{Q} = \{q_{00}, q_{01}, q_{10}, q_{11}\}$, with $q_{\text{fallback}} = q_{11}$. Appendix B.4 gives the selected matrices, precision options, and calibration procedure. For each model–budget pair, one tree is shared across the four LIBERO suites. OFT uses the matching suite checkpoint and its calibrated bank; $\pi_{0.5}$ shares both checkpoint and bank across suites.

4.2 Results

Table 1 compares task success, static model storage, and energy per successful episode at W4A16, W4A8, and W4A4 on the four LIBERO suites. Figure 5 shows inference latency before and after optimization (left), latency comparisons (middle), and latency overhead and energy savings from GPU operating-point adaptation (right).

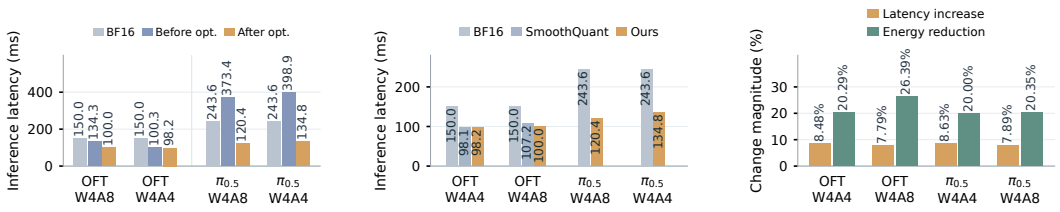


Figure 5: Inference latency before/after optimization (left) and across methods (middle), and DVFS latency overhead and energy-per-success savings relative to 1800 MHz / 300 W (right).

General-purpose quantization comparison. General-purpose methods show larger success losses at W4A4 than at W4A8 (Table 1). At W4A4, ActTune reaches 95.5% on OpenVLA-OFT and 98.4% on $\pi_{0.5}$, exceeding the strongest compared baseline by 4.0 and 5.1 percentage points, respectively. This gap is consistent with preserving precision in action-sensitive computations. ActTune also has the smallest reported storage in these groups (4.47 GB and 3.42 GB, including all resident weight versions) and the lowest W4A8 energy per successful episode on both models at the fixed reference GPU operating point.

VLA quantization comparison. At W4A16, ActTune reaches 97.0% mean success on OpenVLA-OFT and 98.1% on $\pi_{0.5}$, exceeding QVLA (Xu et al., 2026) and ActQuant (Akbari et al., 2026) on both models (Table 1). On $\pi_{0.5}$, it also reduces storage to 3.42 GB, 44.8–45.7% below these baselines, combining higher success with a smaller model. On OpenVLA-OFT, storage is similar: 4.47 GB versus 4.50 GB for QVLA and 4.20 GB for ActQuant. Energy per successful episode is 2.6–5.9% lower on OpenVLA-OFT and 0.3–3.3% lower on $\pi_{0.5}$ than these baselines, but remains above BF16 on both models.

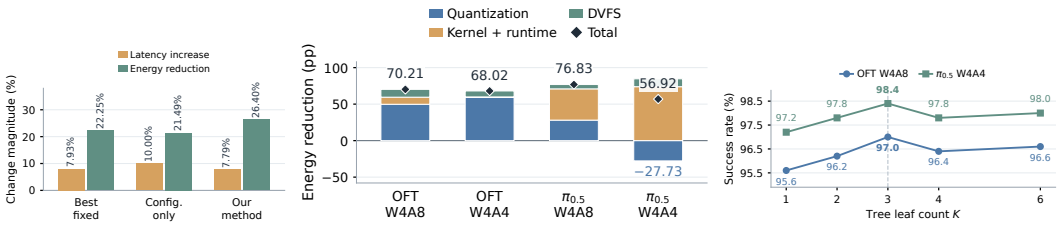
Inference speedup. Figure 5 (left) shows that runtime optimization reduces inference latency across all four configurations, with the largest gains on $\pi_{0.5}$. Its W4A8 and W4A4 implementations change from slower than BF16 to $2.02\times$ and $1.81\times$ faster, while OpenVLA-OFT reaches $1.50\times$ and $1.53\times$, respectively. Figure 5 (middle) compares ActTune inference latency with baseline reference values.

Energy analysis. At the fixed reference GPU operating point, ActTune's W4A8/W4A4 configurations reduce energy per successful episode relative to BF16 by 59.5%/59.9% on OpenVLA-OFT and 70.9%/46.1% on $\pi_{0.5}$ (Table 1). With GPU operating-point adaptation enabled, the parenthesized energy entries in Table 1 correspond to reductions of 70.2%/68.0% and 76.8%/56.9%, respectively. Figure 5 (right) shows that GPU operating-point adaptation increases mean inference latency by 7.79–8.63% relative to the same quantized policies at the fixed 1800 MHz / 300 W reference operating point. All four model–precision settings remain within the 10% inference-latency budget.

4.3 Ablation Study

GPU operating-point selection. We hold the precision selector fixed and compare four GPU operating-point policies on 400 episodes, sampling 100 from each LIBERO suite (Figure 6a). Appendix A.6 describes the sampling procedure. Our method saves 26.40% energy relative to the reference, exceeding the best fixed setting and configuration-only control by 4.15 and 4.91 percentage points, respectively. Its inference latency increase is 7.79%, compared with 7.93% and 10.00% for these alternatives. These results support using decision-region information in hardware selection, while showing that configuration-only adaptation does not necessarily outperform a well-chosen fixed setting.

Energy-saving contributions. Figure 6b reports energy-saving contributions measured through sequential ablations of quantization, kernel and runtime optimization, and GPU operating-point adaptation (DVFS). Each contribution is the measured energy-per-success decrease from the preceding variant, normalized by BF16 energy per success; Appendix C.4 details the ablation and normalization. Quantization contributes 49.82 and 59.27 percentage points on OFT W4A8 and W4A4, while kernel/runtime optimization contributes 42.82 and 73.88 points on $\pi_{0.5}$ W4A8 and W4A4. Lower power does not necessarily mean lower energy: energy integrates power over execution time, so a sufficiently long execution can outweigh a reduction in average power. For $\pi_{0.5}$ W4A4, quantization alone increases energy per success by 27.73%; kernel/runtime optimization and DVFS offset this increase, yielding a net reduction of 56.92%.



(a) GPU settings (b) Energy contributions (c) Closed-loop success

Figure 6: ActTune ablations: (a) GPU operating-point policies; (b) energy-saving contributions in percentage points of BF16 energy per success; (c) closed-loop success versus leaf count K.

Closed-loop effect of tree size. Figure 6c varies the leaf count K of the same action-error-trained selector while keeping the four quantization configurations fixed. Each setting uses the same 2,000 test episodes per model; Appendix A.7 gives the setup. Both models perform best at $K = 3$: 97.0% for OpenVLA-OFT W4A8 and 98.4% for $\pi_{0.5}$ W4A4. These scores exceed $K = 1$, the single-leaf selector with a training-selected fixed configuration, by 1.4 and 1.2 percentage points, and match the corresponding ActTune entries in Table 1. For both models, $K = 4$ and $K = 6$ score 0.6 and 0.4 points below $K = 3$. These closed-loop results support the three-leaf setting; increasing the number of leaves does not improve success in these comparisons.

5 Related Work

Quantization and efficient VLA inference. Quantization and computation reuse reduce the cost of VLA inference. GPTQ and AWQ calibrate low-bit weights (Frantar et al., 2022; Lin et al., 2023), while SmoothQuant and OmniQuant address weight–activation quantization through rescaling and reconstruction (Xiao et al., 2022; Shao et al., 2024). For VLAs, QVLA, ActQuant, and task-evidence-aware mixed-precision quantization of VLA models (Mix-QVLA) use action sensitivity or task evidence to fit precision allocations (Xu et al., 2026; Akbari et al., 2026; Ranjan & Savakis, 2026); DyQ-VLA and MotionVQ adapt precision or quantized representations to motion during execution (Zheng et al., 2026; Song et al., 2026). Other approaches improve decoding and action chunking (Kim et al., 2025), reuse visual tokens or intermediate features (Xu et al., 2025; Yang et al., 2025), or overlap inference with action execution (Black et al., 2025).

Energy-aware inference and GPU operating-point control. Energy-aware deployment must also account for hardware operation and timing constraints (Tang et al., 2019; Stojkovic et al., 2024). Hardware-aware automated quantization (HAQ) uses hardware latency and energy feedback for bit allocation (Wang et al., 2019), and PiQi combines partial quantization, processor placement, and DVFS (Aghapour et al., 2024). DynamoLLM, a system for large language model (LLM) inference, adjusts serving resources and GPU frequency under latency objectives (Stojkovic et al., 2025). For robotic workloads, saliency-aware quantized imitation learning (SQIL) studies quantized imitation policies (Park et al., 2025), while EcoVLA optimizes device-edge execution under latency constraints (Zhou et al., 2026). We combine action-conditioned precision selection with measured GPU operating points and evaluate energy per successful episode, including failed attempts and idle intervals, while keeping the mean inference-latency increase within 10% of the fixed reference operating point.

6 Conclusion

We presented ActTune, a framework for energy-efficient VLA inference through action-aware precision selection and GPU operating-point adaptation. Its decision tree learns splits and leaf configurations directly from configuration action errors, while hardware profiling identifies workload-specific energy–latency tradeoffs. Closed-loop LIBERO evaluation of this controller achieves 97.0% mean success for OpenVLA-OFT W4A8 and 98.4% for $\pi_{0.5}$ W4A4. With GPU operating-point adaptation, ActTune reduces energy per successful task by up to 76.8% relative to the original BF16 implementations, while keeping the inference-latency increase below 10% relative to the same quantized policies at the fixed reference operating point. Validation on physical robots remains future work.

AI USE STATEMENT

Generative AI tools assisted with implementing and revising analysis and visualization scripts, including aggregation of experimental summaries, calculation of speedups and relative energy savings, numerical consistency checks, and generation of figures and LaTeX tables. AI tools also assisted with literature searches, interpretation and presentation of experimental results, cross-reference checks, and language polishing. The authors take responsibility for the final code, experimental results, analyses, and manuscript.

ETHICS STATEMENT

This work studies energy-efficient VLA inference using the LIBERO simulation benchmark. Reducing inference energy may lower the resource cost of robotic policies, but energy efficiency alone does not establish safe robot behavior. Deployment on physical robots requires task-specific safety assessment and validation beyond the simulation experiments reported here.

REPRODUCIBILITY STATEMENT

The experiments can be reproduced using the open-source code provided with this work. The main text and appendices describe the model configurations, calibration procedure, evaluation protocol, and latency and energy measurements needed to reproduce the reported results.

REFERENCES

Ehsan Aghapour, Yixian Shen, Dolly Sapra, Andy D. Pimentel, and Anuj Pathania. PiQi: Partially Quantized DNN Inference on HMPSoCs. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2024. doi: 10.1145/3665314.3670841. URL https://doi.org/10.1145/3665314.3670841.

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. *arXiv preprint arXiv:2605.24011*, 2026. URL https://arxiv.org/abs/2605.24011.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/file/300ccb2187dedd4edcc07f7e76d8e553-Paper-Conference.pdf.

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. URL https://arxiv.org/abs/2210.17323.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An Open-Source Vision-Language-Action Model. *arXiv preprint arXiv:2406.09246*, 2024. URL https://arxiv.org/abs/2406.09246.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. *arXiv preprint arXiv:2502.19645*, 2025. URL https://arxiv.org/abs/2502.19645.

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 LLM Compression and Acceleration. *arXiv preprint arXiv:2306.00978*, 2023. URL https://arxiv.org/abs/2306.00978.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. *arXiv preprint arXiv:2306.03310*, 2023. URL https://arxiv.org/abs/2306.03310.

NVIDIA. *Parallel Thread Execution ISA, Version 7.8*, 2022. URL https://docs.nvidia.com/cuda/archive/11.8.0/parallel-thread-execution/index.html.

Seongmin Park, Hyungmin Kim, Sangwoo Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. Saliency-Aware Quantized Imitation Learning for Efficient Robotic Control. *arXiv preprint arXiv:2505.15304*, 2025. URL https://arxiv.org/abs/2505.15304.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Navin Ranjan and Andreas Savakis. Mix-QVLA: Task-Evidence-Aware Mixed-Precision Quantization of Vision-Language-Action Models. *arXiv preprint arXiv:2606.19565*, 2026. URL https://arxiv.org/abs/2606.19565.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=8Wuvhh0LYW.

Zhuoran Song, Haozhe Jiang, Chunyu Qi, Minnan Pei, Gang Li, Xiaoyao Liang, and Haibing Guan. A Motion-Aware Vector Quantization Framework with Centroid Reuse for Efficient VLA Inference. *arXiv preprint arXiv:2607.24148*, 2026. URL https://arxiv.org/abs/2607.24148.

Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Íñigo Goiri, and Josep Torrellas. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. *arXiv preprint arXiv:2403.20306*, 2024. URL https://arxiv.org/abs/2403.20306.

Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, 2025. doi: 10.1109/HPCA61900.2025.00102. URL https://arxiv.org/abs/2408.00741.

Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The Impact of GPU DVFS on the Energy and Performance of Deep Learning: an Empirical Study. *arXiv preprint arXiv:1905.11012*, 2019. URL https://arxiv.org/abs/1905.11012.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. URL https://openaccess.thecvf.com/content_CVPR_2019/html/Wang_HAQ_Hardware-Aware_Automated_Quantization_With_Mixed_Precision_CVPR_2019_paper.html.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. *arXiv preprint arXiv:2211.10438*, 2022. URL https://arxiv.org/abs/2211.10438.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient Vision-Language-Action Manipulation via Adaptive Token Caching. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.02175.

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. URL https://arxiv.org/abs/2602.03782.

Yantai Yang, Yuhao Wang, Zichen Wen, Zhongwei Luo, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-Free Acceleration and Compression for Vision-Language-Action Models. *arXiv preprint arXiv:2506.10100*, 2025. URL https://arxiv.org/abs/2506.10100.

Zihao Zheng, Hangyu Cao, Sicheng Tian, Jiayu Chen, Maoliang Li, Xinhao Sun, Hailong Zou, Zhaobo Zhang, Xuanzhe Liu, Donggang Cao, Hong Mei, and Xiang Chen. DyQ-VLA: Temporal-Dynamic-Aware Quantization for Embodied Vision-Language-Action Models. *arXiv preprint arXiv:2603.07904*, 2026. URL https://arxiv.org/abs/2603.07904.

Ao Zhou, Bo Dai, Le Yu, Xingyu Liu, Zeyu Hao, Lingkun Long, Chunming Hu, and Jianlei Yang. EcoVLA: Energy-Efficient Device-Edge Co-Inference for Vision-Language-Action Models under Real-Time Constraints. *arXiv preprint arXiv:2608.15502*, 2026. URL https://arxiv.org/abs/2608.15502.

A Appendix

A.1 Evaluation settings and result provenance

All results in Table 1 are obtained from local evaluations, including the BF16 references and every quantization baseline. Success and energy evaluations use the Spatial, Object, Goal, and Long suites of LIBERO, with ten tasks and 50 initial states per task, giving 500 episodes per suite and 2,000 per model–precision setting. We aggregate episodes by task and initial-state identity, counting each episode once even when a run is resumed. The benchmark determines success, and we weight the four suites equally in the reported average. All ActTune results use the action-error-trained decision tree in Sections 3.2–3.3, whose splits and leaf configurations are learned directly from configuration action errors. We use three leaves for the main results and vary this count only in the tree-size ablations. The selector is frozen before closed-loop evaluation and GPU profiling. Table 1 reports closed-loop success and energy for this method; Figure 6c examines closed-loop success as leaf count varies. Appendix C.1 reports the validation action loss and routing cost used to select tree size. OpenVLA-OFT uses eight-action chunks. The evaluated $\pi_{0.5}$ runtime executes ten actions per policy call and retains ten denoising steps. These execution settings are distinct from the eight-action blocks used to define the diagnostic classes in Section 2.

For each model and suite, quantized methods start from the corresponding BF16 checkpoint and use the same 512-trajectory calibration selection described in Appendix B, while retaining their method-specific calibration objectives. The cited papers define the compared methods; the table entries report our local measurements. Each row reports one deployed system. Its success, inference latency, and energy come from the same 2,000 benchmark episodes with the same deployed checkpoint, calibrated configuration, and execution backend. Storage is measured from that deployed model, including all resident alternatives. Cross-method comparisons use the same task–initial-state pairs and hardware settings. The Δ column is measured against the locally evaluated BF16 reference listed for the same model. We compute means and Δ from unrounded suite averages, then round them to one decimal place using half-up rounding.

DyQ-VLA comparison scope. DyQ-VLA (Zheng et al., 2026) is closely related: it studies motion-dependent quantization sensitivity and uses kinematic signals for online precision switching. At the time of this revision (September 2026), it is listed as an arXiv preprint, with no formal conference or journal publication reported. For this reason, we discuss it as related work but do not include it in the quantitative comparison in Table 1.

A.2 Quantization and execution scope

Hardware measurements use NVIDIA RTX A6000 GPUs (Ampere, compute capability 8.6). The selector uses 13 motion and gripper features plus current proprioception (Appendix B.5). We set $\lambda_g = 0.1$ in Equation 1.

W4A8 and W4A4 denote the average precision budgets defined in Section 3.3. Targeted layers can use two-, four-, or eight-bit weights, and W4A4 also allows two-, four-, or eight-bit activations. The action head, proprioception projector, embeddings, normalization, and attention products retain their existing precision. The two switchable matrices reuse resident alternatives, whose storage is included even when they are inactive. The evaluated $\pi_{0.5}$ W4A4 implementation quantizes activations in groups of 64 channels.

Our W4A8/W4A4 inference measurements use the optimized integer backends in Section 3.5. ActTune selects INT4×INT4 or INT8×INT8 kernels according to each layer's weight and activation precision. The W4A4 baselines use INT4×INT4, and the W4A8 baselines use INT8×INT8 with packed W4 weights. These deployed backends produce the reported success, latency, and energy measurements in the same closed-loop runs. Each energy-per-success denominator uses the successes recorded in those runs. The local GPTQ and AWQ W4A4 results and the $\pi_{0.5}$ SmoothQuant W4A4 result use calibration that accounts for A4 activation quantization error.

GPU control and runtime checks. The controller reads back the requested frequency and power cap after each update to verify that they were applied. Operating-point changes are separated by at least two inference requests. Runtime checks verify coverage of the targeted linear layers and compliance with the calibrated precision budgets.

A.3 RETAINED WEIGHT/ACTIVATION PAIRS

Table 2 lists the precision pairs retained in the evaluated banks. For W4A8, the candidate set is $\{\mathrm{W2A8}, \mathrm{W4A8}, \mathrm{W8A8}\}$, and both models retain all three pairs. For W4A4, the candidate space is $\{2, 4, 8\} \times \{2, 4, 8\}$, giving nine possible weight/activation pairs. OpenVLA-OFT retains seven pairs, excluding W2A8 and W8A2; $\pi_{0.5}$ retains six, also excluding W8A8. The retained pairs follow from the layer allocation described in Appendix B.4. Each W4A4 bank uses all three weight widths and all three activation widths, without requiring every pair.

Table 2: Candidate and retained local weight/activation pairs in the frozen evaluated banks. Retained counts are distinct bit-width pairs, not action classes or whole-model configurations.

Model	Budget	Candidates	Retained	Retained pairs
OpenVLA-OFT	W4A8	3	3	W2A8, W4A8, W8A8
OpenVLA-OFT	W4A4	9	7	W2A2, W2A4, W4A2, W4A4, W4A8, W8A4, W8A8
$\pi_{0.5}$	W4A8	3	3	W2A8, W4A8, W8A8
$\pi_{0.5}$	W4A4	9	6	W2A2, W2A4, W4A2, W4A4, W4A8, W8A4

OpenVLA-OFT uses a separately calibrated bank for each LIBERO suite; the retained pair set is the same across all four suites, although the assigned layers differ. Each $\pi_{0.5}$ precision setting uses one shared bank across the four suites. Each of the four precision configurations in a bank contains the reported pair set.

The table counts distinct precision pairs across executed eligible linear matrices, excluding OpenVLA-OFT's unused vocabulary projection. Each pair specifies one matrix's precision; a W8A8 matrix can therefore occur within a W4A4 average budget. A whole-model configuration assigns a pair to every eligible matrix, and the precision selector chooses among four such configurations. The accompanying precision inventory records the source checksums, per-configuration pair counts, weighted bit averages, and every layer's assignment in `layer_assignments.csv`.

A.4 INFERENCE TIMING AND MODEL STORAGE

We record inference latency during the complete closed-loop benchmark runs used for the success and energy comparison. ActTune's selector uses the action history produced by those runs. We use CUDA synchronization to time the complete prediction call, including input processing, configuration selection, policy inference, and action postprocessing. Model loading, compilation, graph capture, remote procedure call (RPC) overhead, and simulation are excluded. Measurements run on an RTX A6000 without competing computation. The no-DVFS comparison uses a fixed 1800 MHz application-clock setting and 300 W power cap; the DVFS comparison enables joint requested-frequency and power-cap adjustment, with the quantized policy fixed.

For suite s, let $\bar{t}_s^{\mathrm{BF16}}$ and $\bar{t}_s^{q}$ denote the mean prediction times of the original BF16 implementation and quantized configuration q. The reported speedup is

$$S(q) = \frac{1}{4} \sum_{s} \frac{\bar{t}_s^{\mathrm{BF16}}}{\bar{t}_s^{q}}. \tag{3}$$

This average of per-suite ratios need not equal the ratio of two rounded overall mean latencies. We use this aggregation formula for all methods. Initial task states match across methods, while subsequent trajectories may differ. The operating-point adaptation budget permits at most a 10% increase in mean complete prediction-call latency relative to the same quantized policy at the fixed reference operating point (no DVFS). We check this budget using the inference timings recorded during the closed-loop benchmark. The BF16 reference is the original implementation used in Table 1; a separately optimized BF16 CUDA Graph implementation is outside this comparison. The runtime optimizations preserve the frozen weights, precision assignments, selectors, and denoising steps. Recorded actions match the reference exactly on the tested inputs, and model tensor bytes remain unchanged. These action-equality checks cover the tested inputs; the full closed-loop runs provide the reported success and energy measurements.

Local model storage is the persistent compressed tensor payload in decimal GB (10^9 bytes). It includes packed weight codes, quantization metadata, retained high-precision tensors, and every

resident weight version, counting shared storage once. For suite-specific checkpoints, our reported value is the maximum across suites. Activation buffers, temporary workspaces, allocator caches, serialization overhead, and nonpersistent positional caches are excluded. For SmoothQuant at W4A8 and W4A4, Table 1 reports the measured local W4 integer-backend storage: 6.32 GB for OpenVLA-OFT and 3.67 GB for $\pi_{0.5}$. The counts include packed weights, scales, offsets, input scales, persistent weight-sum buffers, and unquantized components such as the action head and proprioception projector. The shared packing layout gives the same static storage at both activation precisions. For AWQ and GPTQ, we measure persistent storage from each deployed W4A8/W4A4 model using the same accounting: 5.91 and 6.01 GB for OpenVLA-OFT, and 3.53 and 3.57 GB for $\pi_{0.5}$, respectively. These counts include persistent VLA adapter tensors and unquantized components. The W4A8 and W4A4 deployments have the same reported storage at the table's precision; runtime activation buffers are excluded from this static measure. Static model storage is distinct from sampled GPU memory; the table reports persistent tensor storage.

A.5 Energy measurement

Every energy entry in Table 1, including BF16, W4A16 baselines, and ActTune with and without DVFS, uses the complete LIBERO benchmark: 500 episodes per suite and 2,000 per model–precision setting. Each method and operating-point setting uses the same task–initial-state identities; the subsequent closed-loop trajectories may differ. The unparenthesized local W4A8/W4A4 energy entries use exclusive RTX A6000 runs at a fixed 1800 MHz application-clock setting and a 300 W power cap, with DVFS disabled. Parenthesized ActTune entries report the corresponding results with DVFS enabled. We sample instantaneous GPU-board power through the NVIDIA Management Library (NVML) at approximately 20 Hz and integrate each episode using the trapezoidal rule, from its first inference to its final environment step. The integral includes GPU rendering and intervals between policy calls; model loading, environment resets, explicit warmup, and central processing unit (CPU) energy are excluded. Energy per success is the total energy of all 2,000 attempted episodes, including failures, divided by successes in those same runs. Runs with competing GPU processes are excluded and repeated. This metric measures total GPU-board energy during simulated task execution, including inference, GPU rendering, and intervals between calls. It does not isolate inference energy or measure whole-robot energy. The four-episode exploratory hardware sweep in Section 2 uses a separate cohort and operating settings.

A.6 GPU operating-point ablation cohort

For the GPU operating-point ablation in Section 4.3, we use 400 evaluation episodes per setting: 100 from each of the Spatial, Object, Goal, and Long suites. Within each suite, we randomly sample 100 episodes directly from its pool of 500 task–initial-state pairs (ten tasks with 50 initial states each), without imposing a per-task quota. This subset is distinct from the full 2,000-episode success-rate evaluations in Table 1. The three-leaf precision selector is held fixed across the reference, best fixed, configuration-only, and region-and-configuration GPU operating-point policies.

A.7 Closed-loop leaf-count ablation protocol

We test $K \in \{1, 2, 3, 4, 6\}$ using the calibration data and trajectory-level fitting/validation split in Appendix B. For each model–precision pair, we learn tree splits and leaf configurations directly from the four configuration losses in Equation 1, following Sections 3.2–3.3 and Appendix B.6. Loss-reducing splits grow the initial tree with maximum depth five; bottom-up pruning produces a subtree with each requested leaf count K. One tree shares its splits and configuration indices across the four suites. OFT resolves each index to the matching suite's bank; $\pi_{0.5}$ shares its bank across suites. Several leaves may select the same configuration. Model weights, the four-configuration banks, kernels, and fitting/validation split stay fixed across K. At $K = 1$, the single leaf selects the configuration index with the lowest weighted fitting loss. At $K > 1$, current proprioception and previous-action features route each request to a leaf and its stored configuration. Missing history or invalid features trigger $q_{\text{fallback}} = q_{11}$. Here, K is the number of leaves, as in Figure 6c. The PCA and K-means groups are used only for the Motivation diagnostics and do not supply labels for these trees. Each tree is frozen before evaluation. The $K = 3$ settings use the same selectors as the corresponding ActTune rows in Table 1. Each setting uses the same 500 task–initial-state pairs per suite for closed-loop tests, giving 10,000 episodes per model across all five settings. We report mean success over the four suites.

B CALIBRATION FOLLOWING THE QVLA PROTOCOL

B.1 TRAINING DATA AND HELD-OUT DIAGNOSTICS

Following the training-data source and trajectory-count convention in Appendix F of QVLA (Xu et al., 2026), we sample 512 distinct trajectories from the combined official LIBERO training tasks for calibration. We use a frozen selection from the four suites with sampling seed 20260914. The data snapshot is pinned to revision `f13aa24a3da8c43c7225569f28c562979fa0e35a` of the LIBERO dataset mirror. Forty separate training trajectories are reserved for diagnostics; they and previously held-out diagnostic trajectories are excluded from calibration. Evaluation rollouts are not used to fit quantization parameters or selectors.

Within each selected trajectory, we sample observations from frame zero at stride eight. We exclude the final observation because it lacks a subsequent stored state for alignment checks. This produces 10,996 calibration observations and 882 diagnostic observations (Table 3). An observation at index j is aligned with the next eight demonstration actions; near the trajectory end, the last available action is repeated to complete the stored block. The manifest records trajectory and frame identities, observation hashes, and file checksums. The shared-data loaders check disjoint trajectory identities, disjoint observation hashes, and the fixed frame ordering.

Table 3: Shared training calibration and held-out diagnostic data.

Suite	Calibration trajectories	Calibration frames	Diagnostic trajectories	Diagnostic frames
Spatial	135	2,139	10	162
Object	121	2,341	10	185
Goal	138	2,305	10	174
Long	118	4,211	10	361
Total	512	10,996	40	882

The stride, random seed, padding rule, and diagnostic split are explicit local choices. QVLA does not provide the original trajectory identities or frame schedule, so we follow its data-source and 512-trajectory convention without claiming identical calibration samples. Our shared manifest contains no additional instruction-only subset. Following this calibration convention does not mean using QVLA's channel-wise quantizer. ActTune uses the layer-wise weight banks and precision budgets specified in Section 3.3.

B.2 CALIBRATION TARGETS AND SEPARATION FROM EVALUATION

We use the selected observations to collect activation statistics, measure quantization error, and compare candidate configurations with BF16 reference actions. ActTune calibrates candidate weight/activation settings and fits the configuration mapping using action agreement under the weight, activation, and resident-storage constraints of Section 3. The local baseline adaptations retain their method-specific calibration objectives while using the shared training selection. Layer coverage, quantizers, and execution backends can differ across methods despite the shared calibration set. We freeze the settings before held-out diagnostics and benchmark evaluation, whose outcomes are excluded from fitting.

B.3 PRECISION AND STORAGE BUDGETS

Let $\mathcal{L}$ index the targeted backbone linear matrices, with n_ℓ parameters and v_ℓ input activation elements per request. Each configuration $q \in \mathcal{Q}$ assigns weight bits $b_{\ell q} \in \{2, 4, 8\}$. W4A8 fixes activation bits $a_{\ell q} = 8$, whereas W4A4 allows $a_{\ell q} \in \{2, 4, 8\}$. The budgets are

$$\bar{b}(q) = \frac{\sum_{\ell \in \mathcal{L}} n_\ell b_{\ell q}}{\sum_{\ell \in \mathcal{L}} n_\ell} \le 4, \qquad \bar{a}(q) = \frac{\sum_{\ell \in \mathcal{L}} v_\ell a_{\ell q}}{\sum_{\ell \in \mathcal{L}} v_\ell} \le 4 \quad \text{(W4A4)},$$
$$M_{\text{bank}} = \sum_{\ell \in \mathcal{L}} \sum_{(b,a) \in \mathcal{B}_\ell} \text{bytes}(\ell, b, a) \le M_{\max}, \tag{4}$$

where $\mathcal{B}_\ell = \{(b_{\ell q}, a_{\ell q}) : q \in \mathcal{Q}\}$ contains the distinct resident versions of matrix ℓ. Storage includes packed weights, scales, and stored biases for every version, including inactive alternatives.

The W4A4 activation bound also applies within each group of linear layers with the same invocation shape. Thus, average active precision and full-bank storage are separately constrained; W4A8 and W4A4 are budget labels, not uniform layer assignments. The action head, proprioception projector, embeddings, normalization, and attention products retain their existing precision; the unused vocabulary projection is excluded from $\mathcal{L}$.

B.4 Selecting precision for each layer

We select layer precision offline using reconstruction errors on training data, subject to bit and storage budgets. Let $E_\ell(b, a)$ denote the action-Jacobian-weighted reconstruction error for quantizing eligible matrix ℓ to weight bits b and activation bits a. Calibration uses BF16 reference inputs and actions, activation-guided channel scaling, and row-wise weight clipping. For W4A4, we fit A4 scales and calibrate all nine weight/activation pairs from $\{2, 4, 8\} \times \{2, 4, 8\}$. For $\pi_{0.5}$, the sensitivity calculation includes the ten-step denoising computation and the executed action coordinates. This error is a proxy for ranking assignments; task success is evaluated separately in closed-loop rollouts.

Starting from uniform W4A8 or W4A4, the allocator assigns W2 to a capped set of matrices with the smallest increase in E_ℓ per parameter. For W4A4, it also assigns A2 by ranking the error increase per input activation element within each invocation-shape group. The predeclared W2 parameter fractions are 0.25%, 1%, and 2.5%. For W4A4, these are paired with A2 input-element fractions of 2.5%, 5%, and 10%, respectively. These fractions guide the initial candidate construction; subsequent storage repair can change the final assignments.

These reductions free bit budget for precision upgrades. Writing $N = \sum_\ell n_\ell$ and $V = \sum_\ell v_\ell$, the greedy allocator ranks a feasible upgrade $(b, a) \to (b', a')$ by

$$\frac{E_\ell(b, a) - E_\ell(b', a')}{(b' - b)n_\ell/N + (a' - a)v_\ell/V}. \tag{5}$$

The allocator accepts only upgrades that reduce the error proxy. Every configuration must satisfy the parameter-weighted W4 bound; W4A4 must also satisfy the input-element-weighted A4 bound within each activation shape group. For W4A8, activation precision stays at eight bits throughout allocation. The greedy search enforces these bounds but does not establish a globally optimal layer assignment.

To limit the storage needed for configuration switching, we retain alternatives for two matrices. We choose them from eligible lower-precision alternatives by the smallest reconstruction-error penalty, with a 2 MiB limit per alternative backend at this construction stage. For $\pi_{0.5}$, graph constraints keep vision-branch precision assignments fixed across configurations. The two binary alternatives yield four whole-model configurations; all other matrices share one assignment across them. The fourth configuration combines alternatives already stored for the three base configurations and requires no additional weight version.

The storage check includes all retained alternatives and their quantization metadata. If the bank exceeds its byte limit, we reduce shared W8 matrices to W4, starting with the smallest error penalty per byte saved.

We select among feasible candidate banks by comparing their actions with BF16 on the calibration observations. The score is mean continuous-action absolute error plus 0.1 times gripper disagreement, averaged over the three base configurations.

The deployed W4A4 banks include model-specific refinements. OpenVLA-OFT applies W4-to-W2 reductions ranked using training gradients to meet its compact-storage limit, then recomputes the bit counts and validates loaded storage. The $\pi_{0.5}$ group-64 implementation reuses the calibrated candidate allocations and rescores them with grouped activation quantization before freezing the bank and selector. At inference, the controller selects among the four frozen configurations, and the runtime applies their associated activation precisions.

Table 4 lists the two switchable matrices and their precision options in each evaluated bank; all other targeted matrices retain the same assignment across configurations.

B.5 Decision-tree inputs and diagnostic groups

The tree input is $x_t = [\phi(A_{t-1}, m_{t-1}); s_t]$. Its 13 action features comprise translation and rotation magnitude, variation, curvature, and directional cancellation; combined-motion variation, curvature,

Table 4: Switchable layers in the evaluated banks. OFT denotes OpenVLA-OFT; V, L, and E denote its fused vision encoder, the $\pi_{0.5}$ language model, and the $\pi_{0.5}$ action expert, respectively. Block indices start at zero. Precision options correspond to the listed layers in order.

Model	Budget	Suite	Switchable layers	Precision options
OFT	W4A8	Spatial, Object, Long	`V.1.attn.qkv`	W2A8 / W4A8
			`V.22.mlp.fc1`	W2A8 / W4A8
OFT	W4A8	Goal	`V.19.attn.proj`	W2A8 / W4A8
			`V.22.attn.proj`	W2A8 / W4A8
OFT	W4A4	Spatial	`V.1.attn.qkv`	W2A2 / W4A2
			`V.23.mlp.fc1`	W2A2 / W4A2
OFT	W4A4	Object	`V.17.attn.proj`	W2A2 / W4A2
			`V.23.mlp.fc1`	W2A4 / W4A4
OFT	W4A4	Goal	`V.17.attn.proj`	W2A2 / W4A2
			`V.22.mlp.fc1`	W2A2 / W4A2
OFT	W4A4	Long	`V.22.attn.proj`	W2A4 / W4A4
			`V.23.mlp.fc1`	W2A2 / W4A2
$\pi_{0.5}$	W4A8	All four	`L.16.self_attn.o_proj`	W2A8 / W4A8
			`E.1.self_attn.q_proj`	W2A8 / W4A8
$\pi_{0.5}$	W4A4	All four	`L.15.self_attn.q_proj`	W2A2 / W4A2
			`L.16.self_attn.o_proj`	W2A2 / W4A2

and cancellation; and gripper transitions and margin. Current proprioception completes the input. Missing history or nonfinite features trigger q_{11}.

The C0–C2 groups organize the Motivation diagnostics. They use standardized 8×7 action blocks, nine PCA components (95.97% training variance), and K-means with $K = 3$, seed 7, and 20 initializations. Training counts are 4,212/3,404/3,380. These diagnostic labels are separate from the decision regions learned from configuration errors.

B.6 Adaptive tree fitting and trajectory-level validation

We split calibration trajectories into fitting and validation subsets before calibrating the bank or tree. Fitting contexts provide the causal features above and four configuration losses from Equation 1, measured with identical observations and policy random draws. Contexts without history use the fallback and are excluded from tree fitting. For context i in trajectory τ and suite s, we set $w_i = 1/(4M_s n_\tau)$, where M_s counts fitting trajectories with valid history in suite s, and n_τ counts valid fitting contexts in τ.

For fitting indices I at a node, its configuration and total loss are

$$q_I = \arg\min_{q \in \mathcal{Q}} \sum_{i \in I} w_i \ell_i(q), \qquad R(I) = \sum_{i \in I} w_i \ell_i(q_I). \tag{6}$$

We choose the feature threshold maximizing $R(I) - R(I_L) - R(I_R)$, requiring a positive reduction and at least 20 contexts per child. The initial tree has a depth cap of five. Let $D_v(r)$ be the minimum fitting loss of a subtree rooted at node v with exactly r leaves. Bottom-up pruning sets $D_v(1) = R(I_v)$ and, for $r \geq 2$, minimizes $D_{v_L}(j) + D_{v_R}(r - j)$ over feasible child leaf counts j. Infeasible counts have infinite cost. Backtracking from $D_{\text{root}}(K)$ yields the fitting-optimal subtree for each feasible leaf count K, approximating Equation 2; it is not a global search over all possible trees. The candidate set includes a single leaf and does not impose $K = 3$.

On held-out trajectories, we average loss within trajectories, then equally within and across suites, including the fallback at history-free requests. Validation measures the loss–routing-cost tradeoff across candidate sizes without changing the fitted splits or leaf choices. We use these curves to choose the experimental operating point; automatic one-standard-error selection is a separate exploratory comparison, not the rule used to obtain the final common three-leaf setting. Several leaves may share a configuration. The fixed tree and bank remain frozen for subsequent GPU profiling and evaluation.

C SUPPLEMENTARY RESULTS

C.1 LEAF-COUNT ABLATIONS ACROSS MODELS

OFT W4A8. Figure 7 compares 1–10 leaves for the OFT W4A8 tree shared across four LIBERO suites. Three leaves yield the lowest observed validation action loss, reducing it by 0.182% relative to a single leaf, with a CPU tree-traversal latency of 0.182 μs. Every larger candidate has both higher loss and higher traversal latency. Fewer leaves route faster but incur higher action loss. We therefore adopt three leaves as the common experimental setting based on this comparison and the tradeoffs across models.

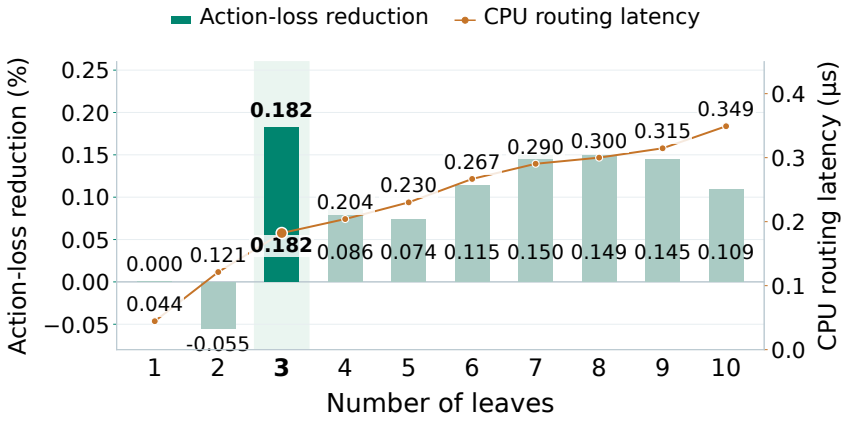


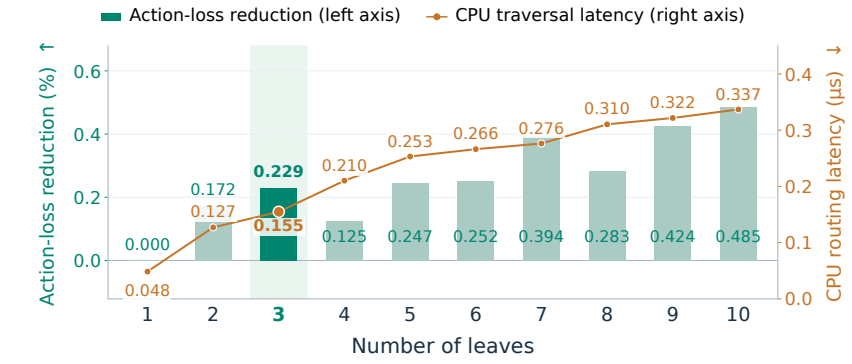
Figure 7: OFT W4A8 leaf-count ablation for one tree shared across four LIBERO suites. Bars show validation action-loss reduction relative to one leaf; the line shows CPU routing latency. Three leaves give the lowest observed validation loss in this setting.

Other model–precision settings. Figure 8 extends Figure 7 to OFT W4A4 and $\pi_{0.5}$ W4A4/W4A8, using one tree shared across four LIBERO suites in each setting. We evaluate every fixed leaf count from 1 to 10 by minimizing fitting loss over subtrees of the grown tree, including counts absent from the cost–complexity path, without refitting on validation data. Action loss is averaged within each of 40 validation trajectories, then equally within and across suites; CPU traversal latency is measured over 21 rounds with the same protocol as Figure 7 and excludes model inference and action-feature extraction. Within this range, the lowest observed validation loss occurs at 10 leaves for OFT W4A4, 3 for $\pi_{0.5}$ W4A4, and 10 for $\pi_{0.5}$ W4A8, with reductions of 0.485%, 0.014%, and 0.003% relative to their single-leaf baselines. These minima show that three leaves do not minimize loss in every setting. Offline loss alone does not establish a closed-loop benefit; Figure 6c reports the separate closed-loop leaf-count comparison for OFT W4A8 and $\pi_{0.5}$ W4A4.

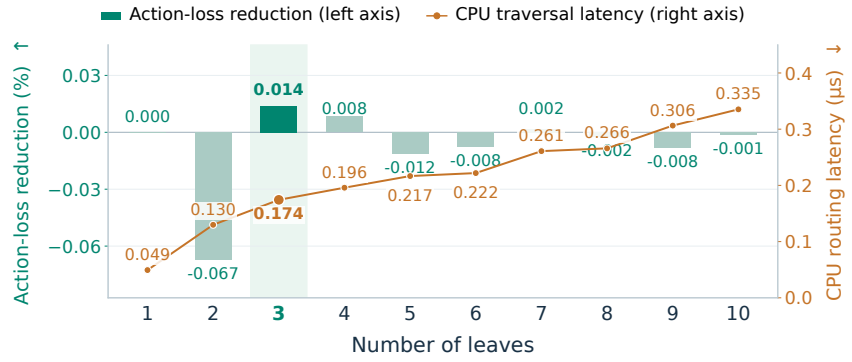
Rationale for a common three-leaf setting. We adopt three leaves as a common experimental setting across model–precision pairs based on these validation curves. This choice retains context-dependent precision selection and requires at most two tree comparisons per decision. For OFT W4A4, three leaves reduce action loss by 0.229% relative to a single leaf; their loss is 0.257% above the ten-leaf minimum, while traversal takes 0.155 rather than 0.337 μs. For $\pi_{0.5}$ W4A4, three leaves attain the lowest observed loss. For $\pi_{0.5}$ W4A8, three leaves increase loss by 0.012% relative to a single leaf and remain within 0.016% of the ten-leaf minimum, with traversal taking 0.130 rather than 0.426 μs. The common three-leaf setting thus trades some models' minimum loss for lower routing cost; single-leaf routing remains faster. This comparison of fixed leaf counts is separate from automatic one-standard-error selection.

C.2 INFERENCE LATENCY BY SUITE

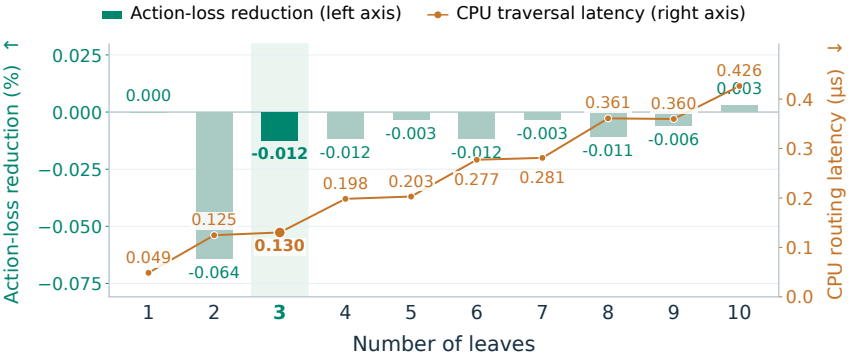
Table 5 gives the per-suite times underlying the four ActTune speedups reported in Section 4.2 and the optimized latencies shown in Figure 5 (left). The speedups include runtime optimization and do not isolate quantization alone.



(a) OFT W4A4



(b) $\pi_{0.5}$ W4A4



(c) $\pi_{0.5}$ W4A8

Figure 8: Shared-tree leaf-count ablations for the remaining model–precision settings. Bars: validation action-loss reduction; lines: CPU routing latency. Three leaves are highlighted for comparison.

Table 5: Per-suite mean inference latency (ms) and speedup relative to the original BF16 runtime.

Model	Setting	Spatial	Object	Goal	Long	Mean	Speedup
OpenVLA-OFT	BF16	149.21	150.69	148.71	151.52	150.03	1.00×
OpenVLA-OFT	W4A8	99.76	99.65	99.47	101.31	100.05	1.50×
OpenVLA-OFT	W4A4	98.50	98.08	98.16	97.87	98.15	1.53×
$\pi_{0.5}$	BF16	239.00	243.90	243.76	247.59	243.56	1.00×
$\pi_{0.5}$	W4A8	119.72	120.38	120.22	121.23	120.39	2.02×
$\pi_{0.5}$	W4A4	133.97	134.15	135.47	135.69	134.82	1.81×

C.3 Effect of inference optimization

Table 6 compares the earlier quantized runtime with the final implementation used in Table 1. The earlier and final timings come from separate measurement campaigns; each reported speedup averages the four per-suite latency ratios. BF16 denotes the original full-policy implementation used in Table 1. The archived records agree on input identities, predicted actions, selected profiles, and model tensor storage.

Table 6: Full-policy inference latency before and after runtime optimization (ms), with speedups relative to the earlier quantized runtime and original BF16 implementation.

Model	Setting	BF16	Earlier	Optimized	Speedup vs. earlier	Speedup vs. BF16
OpenVLA-OFT	W4A8	150.03	134.33	100.05	1.34×	1.50×
OpenVLA-OFT	W4A4	150.03	100.32	98.15	1.02×	1.53×
$\pi_{0.5}$	W4A8	243.56	373.39	120.39	3.10×	2.02×
$\pi_{0.5}$	W4A4	243.56	398.94	134.82	2.96×	1.81×

The comparison measures the combined effects of kernel and runtime optimization, including CUDA Graph replay, launch-geometry tuning, and model-specific execution changes. It does not isolate the contribution of GEMM kernels. The earlier OpenVLA-OFT W4A4 implementation already used language and vision graphs, leaving a smaller additional gain than the other configurations. Matching actions on these recorded inputs is not a new closed-loop success-rate evaluation.

C.4 Energy-saving contribution breakdown

Figure 6b is based on energy measurements from sequential component ablation experiments. For each model–precision setting, we measure GPU-board energy per successful episode for four variants: the BF16 reference, quantization only, quantization with kernel and runtime optimization, and the optimized quantized policy with GPU operating-point adaptation (DVFS). Table 7 reports the resulting contributions. Each contribution is computed from the measured decrease in energy per successful episode between consecutive variants, divided by the corresponding measured BF16 reference energy per success and expressed in percentage points. Thus, the contributions add to the total saving up to rounding; they are not percentage reductions with a different denominator at each stage. Kernel and runtime optimization are reported jointly. DVFS includes requested-frequency and power-cap adjustment, as defined in Section 3.4. Negative contributions denote increased energy. The -27.73-point quantization contribution for $\pi_{0.5}$ W4A4 means that this stage uses 127.73% of BF16 energy per success. The subsequent 73.88-point kernel/runtime contribution offsets that increase, and the additional 10.77-point DVFS contribution yields a total energy saving of 56.92%. Energy depends on both average power and execution time. Lower power can therefore coexist with higher energy if execution takes sufficiently longer. For energy per success, failed attempts and the number of successes also affect the reported value.

Table 7: Energy-saving contributions from sequential ablation measurements, with every component normalized by the corresponding measured BF16 energy per success. Component values are percentage points; total savings are percentages. The displayed components sum to the total saving in each row.

Model	Precision	Quantization	Kernel + runtime	DVFS	Total (%)
OpenVLA-OFT	W4A8	49.82	9.71	10.68	70.21
OpenVLA-OFT	W4A4	59.27	0.61	8.14	68.02
$\pi_{0.5}$	W4A8	28.09	42.82	5.92	76.83
$\pi_{0.5}$	W4A4	-27.73	73.88	10.77	56.92

C.5 Complete action-precision intervention grid

Table 8 expands the selected comparisons in Figure 2(b) to all nine weight/activation combinations. For each of the ten tasks per suite, the experiment uses official initial state 1 and selects the first

complete BF16 action block assigned to each class. Every branch restores the same state, applies quantization for one eight-action block, and then returns to BF16 control until success or the original episode limit. A BF16 sham branch verifies the reference continuation. Each class–precision cell has ten paired branches per suite, except Goal C1 and C2, which have nine and eight because some reference trajectories contain no qualifying class window. Missing windows are not imputed; success rates use the available paired branches within each suite, and the table weights suites equally.

Only the seven linear projections in each of the 32 language Transformer blocks are quantized in this intervention study. Weights use the shared-training GPTQ group-128 calibration, while A2/A4/A8 use the same dynamic per-token affine quantization rule with 4/16/256 codes. The vision modules and action head remain BF16. These BF16 dense quantize–dequantize interventions diagnose sensitivity and are separate from ActTune's native integer-runtime evaluation. Branches from one task share a reference episode and are not independent trials; the grid is a small diagnostic study, not the 500-episode-per-suite benchmark.

Table 8: Final success after one action-block precision intervention (%). Entries average the four suites; the matched BF16 mean is 95.0%.

Intervention	C0	C1	C2
BF16	95.0	95.0	95.0
W2A2	62.5	35.0	91.9
W2A4	67.5	42.5	95.0
W2A8	97.5	97.5	94.4
W4A2	75.0	62.5	95.0
W4A4	70.0	42.5	92.5
W4A8	97.5	97.5	95.0
W8A2	75.0	52.5	95.0
W8A4	67.5	45.0	92.5
W8A8	97.5	97.5	95.0

With this small cohort and fixed quantizer, nonmonotonic entries do not establish a general advantage of lower precision or an ordering of C0–C2 by sensitivity.

C.6 Action-grouping diagnostics

On the 882 held-out blocks, the mean PCA-space silhouette is 0.1541, with 7.26% negative values, indicating class overlap. Across 20 refits that resample whole calibration trajectories within each suite, agreement with the frozen labels on these blocks has mean adjusted Rand index 0.8455 (range 0.7198–0.9829); all refits retain nine components. These are descriptive separation and stability statistics, not classification accuracy or confidence intervals on task success.