SACQ: STRUCTURE-AWARE CODEBOOK QUANTIZATION FOR ULTRA-LOW-BIT VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models produce remarkably capable robot behavior, but their memory footprint makes deployment on resource-constrained robotic platforms challenging. Weight quantization is a natural remedy, yet post-training quantization (PTQ) recipes inherited from large language models degrade sharply when applied to VLAs at ultra-low precision. Two limitations become particularly important: existing low-bit PTQ methods typically assess quantization error at the layer or projection level without modeling the downstream operation through which it propagates, and allocate precision without directly measuring how reducing a layer's precision changes the policy action. We introduce **SACQ**, an ultra-low-bit weight-only vector-codebook quantization framework for VLAs. SACQ combines (1) **structure-aware codebook fitting**, which fits each layer using a role-specific quadratic metric derived from the operation its projection feeds; (2) **action-aware mixed-bit allocation**, which assigns precision according to decoded-action degradation as individual layers are compressed; and (3) **two-stage post-quantization refinement**, which first reassigns codeword indices with second-order error compensation and then calibrates codebooks and row-group scales against the decoded action. On GR00T N1.7, SACQ retains 93.85% average success across all four LIBERO suites at approximately 1.5 bits per language/action weight, only 0.25 percentage points below the 94.10% BF16 reference, while reducing resident model memory from 5.677 GB to 1.175 GB, a **4.83× reduction**. On SimplER Google Robot, at approximately 2 bits per language/action weight, SACQ achieves 63.76% and 52.27% success under Visual Matching and Variant Aggregation, respectively, compared with 58.90% and 53.45% for BF16.

1 INTRODUCTION

Vision-Language-Action (VLA) models have made rapid progress in robotic manipulation by combining visual perception, language understanding, and action generation within a single policy Kim et al. (2025b); Physical Intelligence et al. (2025); NVIDIA et al. (2025). Recent models perform strongly across diverse manipulation tasks, establishing VLAs as an important approach to generalist robot policies. At the same time, VLA models have grown substantially in scale, creating significant memory demands for resource-constrained robotic platforms.

Post-training quantization (PTQ) is attractive in this setting because it reduces model memory without requiring full retraining. However, ultra-low-bit VLA quantization is especially challenging because the effect of a weight error depends strongly on where it appears in the policy. For example, errors in query and key projections perturb the attention logits that determine token mixing, whereas errors in feed-forward projections alter the hidden representations passed to subsequent operations. As a result, similar layer-local reconstruction errors can have substantially different consequences depending on where they occur. A related challenge is precision allocation: a layer that appears insensitive under a local reconstruction objective can still have a large effect on the final action.

Related Work. Representative PTQ methods such as GPTQ (Frantar et al., 2023) optimize layer-wise output reconstruction using approximate second-order information. GPTVQ (Van Baalen

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.

et al., 2024) extends this reconstruction-based formulation to vector quantization by incorporating the Hessian of the layer-output reconstruction objective into codeword assignment. Other vector-quantization methods, including AQLM (Egiazarian et al., 2024) and QuIP# (Tseng et al., 2024), further demonstrate the effectiveness of codebook-based representations for extreme LLM compression. However, these methods primarily optimize quantization with respect to layer-local reconstruction and do not explicitly account for the operation in which each projection output is subsequently used. For VLAs, recent quantization methods increasingly incorporate properties of action generation into the quantization process. QVLA (Xu et al., 2026) uses action-related sensitivity to guide mixed-precision allocation, while ActQuant (Akbari et al., 2026) combines action-sensitive inter-tensor bit allocation with intra-tensor scale optimization. HBVLA (Yan et al., 2026) extends VLA weight quantization to the 1-bit regime, while QuantVLA (Zhang et al., 2026) explicitly considers the structure of the diffusion action transformer in its quantization design.

Contributions. We introduce SACQ, a weight-only vector-codebook framework for ultra-low-bit VLA quantization with three components. First, **structure-aware codebook fitting** uses projection-specific metrics that reflect each projection's functional role, rather than relying solely on layer-local reconstruction. Second, **action-aware mixed-bit allocation** assigns precision according to the increase in final-action error per bit saved when each layer is moved to a lower-precision state. Third, **two-stage post-quantization refinement** first reassigns codeword indices using second-order error compensation and then, with the updated indices fixed, calibrates codebooks and row-group scales against the decoded action. On GR00T N1.7, SACQ retains 93.85% average success across the four LIBERO suites at approximately 1.5 bits per language/action weight, only 0.25 percentage points below the 94.10% BF16 reference, while reducing resident model memory from 5.677 GB to 1.175 GB, a 4.83× reduction. On SimplER Google Robot, SACQ further achieves 63.76% and 52.27% success under Visual Matching and Variant Aggregation, respectively, at approximately 2 L/A bpw, remaining close to the corresponding BF16 references.

2 Background

2.1 Vision-Language-Action Models

A VLA policy maps a visual observation and a language instruction to an action or a chunk of actions, which is executed on the robot before the next observation is taken. Its components can be broadly grouped into four modules: a vision encoder that turns visual inputs into feature representations, a projector that maps those features into the embedding space of the language model, a language backbone that processes vision and language jointly, and an action head that decodes the backbone representation into an executable action.

In both GR00T N1.7 and OpenVLA-OFT, the language backbone consists of decoder blocks containing the attention projections q, k, v, and o, and the feed-forward projections gate, up, and down. The action head, however, differs across the two model families. In GR00T N1.7 (NVIDIA, 2025), it is a diffusion transformer that denoises an action chunk over a fixed number of steps; each block contains attention projections, feed-forward projections, and an adaptive layer normalization projection that produces per-channel modulation parameters from the diffusion timestep embedding. In OpenVLA-OFT (Kim et al., 2025a), a lightweight MLP-based action head directly predicts a chunk of continuous-valued actions in a single forward pass.

2.2 Vector Codebook Quantization

Weight quantization replaces each weight matrix $W \in \mathbb{R}^{O\times I}$ by a low-precision representation $\widehat{W}$, reducing the memory needed to store and load the model. Post-training quantization does this without full model retraining, typically using a small calibration set drawn from the deployment distribution.

Scalar quantization encodes each weight independently, whereas vector quantization represents a d-dimensional weight vector by a single index into a shared codebook $C = \{c_1, \ldots, c_M\} \subseteq \mathbb{R}^d$, at a nominal index rate of

$$b(M) = \frac{\log_2 M}{d}. \tag{1}$$

We consider a block-scaled vector-codebook representation (Van Baalen et al., 2024). The elements of W are first partitioned into blocks of g consecutive entries, and each block is normalized by its own scale s, typically the maximum absolute value within the block, so that codewords are fit to a common range across the matrix. The normalized entries are then grouped into d-dimensional vectors v, each of which is assigned an index $a(v) \in \{1, \dots, M\}$ and reconstructed as

$$\hat{v} = s\, c_{a(v)}, \tag{2}$$

so a layer stores its codebook, one index per vector, and one scale per block.

2.3 Codebook Optimization

Let $\mathcal{V}$ denote the set of d-dimensional vectors covered by one codebook, obtained from the normalized weights as in Section 2.2. Fitting the codebook means choosing the codewords C and the assignment a that minimize the total distortion over $\mathcal{V}$.

The simplest way to measure that distortion is Euclidean distance, $\sum_v \|v - c_{a(v)}\|^2$, which corresponds to ordinary k-means. This measure treats all d coordinates alike. In general, however, an error along one coordinate may cost more than an error along another, and the cost of two coordinates being wrong together may differ from the sum of their individual costs. Capturing this requires assigning each vector a positive semidefinite matrix $G(v) \in \mathbb{R}^{d\times d}$ and measuring the error as a quadratic form, which generalizes the objective to

$$\min_{C,\,a} \sum_{v\in\mathcal{V}} \big(v - c_{a(v)}\big)^\top G(v) \big(v - c_{a(v)}\big), \tag{3}$$

where v and $c_{a(v)}$ are points in $\mathbb{R}^d$. The diagonal entries of $G(v)$ state the cost of an error along each of the d coordinates, and its off-diagonal entries state how errors along different coordinates combine. Setting $G(v) = I$ for every v returns to the Euclidean case above. Note that Equation (3) assumes that the distortion decomposes into independent per-vector terms, so that the errors of different codewords do not interact.

The choice of G determines where the codewords are placed. A codebook covers far more vectors than it has codewords, so most vectors are represented only approximately, and fitting is necessarily a matter of which errors to accept and which to avoid. G is what encodes that ordering. A common data-aware choice derives G from layer-output reconstruction. For calibration inputs X and quantization error $E = W - \widehat{W}$,

$$\big\|XE^\top\big\|_F^2 = \mathrm{tr}\big(EX^\top XE^\top\big)\,. \tag{4}$$

GPTVQ (Van Baalen et al., 2024) uses local Hessian blocks of this reconstruction objective as the metric for vector assignment. Given $G(v)$, we optimize equation 3 by alternating between codeword assignment and codeword updates; the complete updates are provided in Appendix A.1.

Second-order information can also be used to compensate for errors introduced during sequential quantization. GPTQ (Frantar et al., 2023) sequentially quantizes weights and uses second-order information to adjust the remaining unquantized weights so as to reduce the resulting layer-output error. GPTVQ (Van Baalen et al., 2024) extends this principle to vector quantization, applying the same form of error compensation after quantizing weight vectors.

3 SACQ: Structure-Aware Codebook Fitting Quantization

SACQ consists of three components: structure-aware codebook fitting, action-aware mixed-bit allocation, and two-stage post-quantization refinement. First, we fit each codebook using a metric derived from the downstream computation in which its projection output is used. We then allocate precision based on the increase in final-action error as individual layers are compressed. Finally, we refine the quantized model by reassigning codeword indices with second-order error compensation and, with the updated indices fixed, calibrating the codebooks and row-group scales against the decoded action. We use the vector-codebook representation of Section 2.2 with $g = 64$, $d = 4$, and a single codebook per layer; row-group scales are initialized by their maximum absolute values.

3.1 Structure-Aware Metric Construction

The layer-local objective in Section 2.3 measures quantization error at the output of each individual projection. Because its metric is derived from input statistics, the same input-side weighting is shared across output rows and does not reflect how a particular output channel is subsequently used.

A projection output, however, is consumed by different downstream operations depending on its role. For example, errors in query and key projections perturb the attention logits that determine token mixing, while errors in value projections are weighted by the attention probabilities and passed through the output projection. The output of a gate projection passes through a nonlinearity and is multiplied elementwise with that of the up projection. Thus, projection-output errors of similar magnitude can induce substantially different downstream distortions.

We therefore derive the metric from the distortion of the downstream computation in which each projection output is used, rather than from the projection output itself. The following paragraphs derive this metric for each projection role.

Attention query and key projections. The output of a query projection is multiplied with the keys to form the attention logits. Ignoring the attention scaling factor,

$$Q = XW_Q^\top, \qquad K = XW_K^\top, \qquad L = QK^\top. \tag{5}$$

Let $E_Q = W_Q - \widehat{W}_Q$ denote the quantization error of the query projection. For a calibration input X, this changes the queries by

$$\Delta Q = XE_Q^\top.$$

Holding the keys K fixed, the corresponding change in the attention logits is

$$\Delta L = \Delta QK^\top = XE_Q^\top K^\top. \tag{6}$$

The squared Frobenius norm of this logit perturbation is therefore

$$\|\Delta L\|_F^2 = \|XE_Q^\top K^\top\|_F^2 = \mathrm{tr}\big(E_Q X^\top X E_Q^\top K^\top K\big) = \mathrm{tr}\big(E_Q A E_Q^\top B\big), \tag{7}$$

where $A = X^\top X$ and $B = K^\top K$.

Here, A is the Gram matrix of the calibration inputs, capturing the second-order statistics of the input features. Similarly, B is the Gram matrix of the keys. Its diagonal entry B_{rr} measures the energy of key feature r, while its off-diagonal entries capture interactions between quantization errors from different query output channels. Compared with the layer-local reconstruction objective in Section 2.3, the output-side weighting B is the new term, and it arises because the error is measured at the attention logits rather than at the projection output.

The key projection follows by exchanging the roles of queries and keys. Holding the queries fixed and expanding in $E_K = W_K - \widehat{W}_K$ gives $\Delta L = QE_K X^\top$, with the same input-side Gram $A = X^\top X$ and an output-side weighting $B = Q^\top Q$.

Separable local objective. The loss in equation 7 is not separable across codewords. Errors from different output channels can interact through B, and errors on weights connected to different input features can interact through A. The loss therefore cannot be optimized independently for each codeword.

To fit each codeword independently, we keep only the diagonal of B and restrict A to the 4×4 block corresponding to the four input features spanned by that codeword. For a codeword v belonging to output row $r(v)$ and occupying input coordinates $I(v)$, this gives

$$G(v) = B_{r(v),\,r(v)}\, A_{I(v),\,I(v)}. \tag{8}$$

The block $A_{I(v),I(v)}$ retains correlations among the four input features within the codeword, while $B_{r(v),r(v)}$ acts as a scalar weight for the output channel containing it. Cross-channel terms in B and cross-codeword correlations in A are dropped, yielding a separable, block-diagonal approximation to the full quadratic objective. This approximation enables independent codebook fitting, while cross-vector quantization errors are subsequently addressed by the second-order refinement in Section 3.3.

Attention value and output projections. The output of a value projection is not used on its own: it is weighted by the attention probabilities and then passed through the output projection. Writing the attention probabilities as P,

$$V = XW_V^\top, \qquad C = PV, \qquad Y = CW_O^\top, \tag{9}$$

where C is the attention context and Y is the block output. Let $E_V = W_V - \widehat{W}_V$ denote the quantization error of the value projection. The values change by $\Delta V = XE_V^\top$, and holding P fixed at the BF16 calibration trajectory, the corresponding change in the block output is

$$\Delta Y = P\,\Delta V\,W_O^\top = PXE_V^\top W_O^\top, \tag{10}$$

whose squared Frobenius norm is

$$\|\Delta Y\|_F^2 = \mathrm{tr}\big(E_V\,X^\top P^\top PX\,E_V^\top\,W_O^\top W_O\big), \tag{11}$$

of the same form as equation 7, with $A = X^\top P^\top PX$ and $B = W_O^\top W_O$. The input-side Gram is thus the input Gram weighted by the attention probabilities, and the output-side weighting is the Gram of the output projection.

Applying the same separable approximation, we retain only the diagonals of $P^\top P$ and of $W_O^\top W_O$. Diagonalizing $P^\top P$ weights each source token by its squared attention probability and excludes cross terms between different source tokens. For source token s, that diagonal entry is the sum over all attention heads h and query positions t,

$$\omega_s = \sum_{h,t} P_{h,t,s}^2, \tag{12}$$

which measures how often and how strongly token s is attended to. Diagonalizing $W_O^\top W_O$ gives output channel j the weight $\|W_O[:,j]\|_2^2$ and excludes coupling between channels. The input-side Gram and output-side weighting are therefore

$$A = \mathbb{E}\Big[\sum_s \omega_s\,x_s x_s^\top\Big], \qquad B_{jj} = \big\|W_O[:,j]\big\|_2^2. \tag{13}$$

The input to an output projection is the context C itself, so its input-side Gram is $A = \mathbb{E}[C^\top C]$. We do not model a downstream operator for its output and target the reconstruction of that output directly, so $B = I$.

Feed-forward projections. The feed-forward block is given by

$$g = hW_g^\top, \qquad u = hW_u^\top, \qquad z = \mathrm{SiLU}(g) \odot u, \qquad y = zW_d^\top. \tag{14}$$

For the down projection, whose input is the post-activation representation z, we use

$$A = \mathbb{E}[zz^\top], \qquad B = I. \tag{15}$$

The gate and up projections interact nonlinearly through SwiGLU before passing through the down-stream projection W_d. For a four-dimensional weight vector v in output channel r, let h_v denote the input activations corresponding to its four weights, and let g_r and u_r denote the corresponding gate and up activations. We use the diagonal metrics

$$G_{\mathrm{gate}}(v) = \|W_d[:,r]\|_2^2\,\mathrm{Diag}\big(\mathbb{E}\big[(\mathrm{SiLU}'(g_r)u_r)^2(h_v \odot h_v)\big]\big), \tag{16}$$

$$G_{\mathrm{up}}(v) = \|W_d[:,r]\|_2^2\,\mathrm{Diag}\big(\mathbb{E}\big[\mathrm{SiLU}(g_r)^2(h_v \odot h_v)\big]\big). \tag{17}$$

The full first-order derivation is provided in Appendix A.2.

Action-head projections. The DiT action head follows the same structure-aware principle, while accounting for its model-specific components: separate query- and key/value-side input statistics in cross-attention, GELU-based feed-forward sensitivities, and output-side weighting for AdaLN modulation. The complete role-specific metrics and derivations are provided in Appendix A.3.

Fitting. With $G(v)$ determined for each projection role, we apply the metric-weighted codebook optimization described in Section 2.3. Codebook fitting is performed on the scale-normalized weight vectors defined in Section 2.2, without additionally weighting the metric by the squared normalization scale. For each layer, we fit the codebook on a sampled subset of vectors and then, with the codebook fixed, recompute assignments over all vectors in the layer. Further fitting details are provided in Appendix A.5.

3.2 Action-Aware Mixed-Bit Allocation

The codebook fitting in Section 3.1 is performed independently for each layer and does not determine how many bits each layer receives. This section determines that assignment under a global storage budget.

Quantization states. Each candidate layer takes one of a set of states ordered by precision,

$$\mathrm{GPTQ4} \to \mathrm{GPTQ3} \to \mathrm{VC2.75} \to \mathrm{VC2.50} \to \cdots \to \mathrm{VC1.00}, \tag{18}$$

where VCb denotes a vector-codebook state of nominal rate b, whose codebook size follows from equation 1 as $M = 2^{db}$. For the 1-bit operating point, we additionally allow the VC0.75 state. Only transitions between adjacent states are allowed.

Allocation by action error. Precision is allocated according to the effect on the final decoded action rather than the layer-wise reconstruction error. For each layer ℓ, we measure the increase Δ_ℓ in final-action relative error caused by moving it one state down, averaged over the calibration set with equal weighting across suites. This is measured by actually running the model with that layer at lower precision, rather than relying on a layer-local sensitivity proxy.

Storage accounting. For allocation, we compute the calculated bits per weight of the language backbone and action head as the total stored bits of each component divided by its parameter count, including metadata such as codebooks, indices, and row-group scales. This overhead is not negligible at low rates, so the allocation is driven by this quantity rather than by the nominal rate. We write $\mathrm{cost}_\ell(s)$ for the storage cost of layer ℓ in state s.

Selection rule. At each step, we select the component with the higher calculated bits per weight and, within it, choose the layer with the smallest increase in action error per bit saved,

$$\ell^\star = \arg\min_\ell \frac{\Delta_\ell}{\mathrm{cost}_\ell(s_\ell) - \mathrm{cost}_\ell(s'_\ell)}, \tag{19}$$

where s_ℓ is the current state of layer ℓ and s'_ℓ is its next lower-precision state. This component-balancing rule prevents either the language backbone or the action head from being compressed disproportionately relative to the other. Further details of the allocation procedure are provided in Appendix B.

3.3 Post-Quantization Refinement

After structure-aware fitting and mixed-bit allocation, we refine the quantized policy in two stages. We first reassign the discrete codeword indices using second-order compensation and then, with the updated indices fixed, calibrate the codebooks and row-group scales against the decoded action.

Index reassignment. Following the second-order compensation mentioned in Section 2.3, we refine the discrete assignments obtained from structure-aware fitting. For each target layer, we run the frozen quantized policy on the calibration data and collect the input second-moment matrix

$$H_\ell = \frac{1}{N_\ell} \sum_{t=1}^{N_\ell} x_t^\top x_t. \tag{20}$$

Although H_ℓ is collected from the quantized policy, the reconstruction target for reassignment is the original BF16 weight. Let $\widetilde{H}_\ell$ denote the damped version of H_ℓ, and write

$$\widetilde{H}_\ell^{-1} = U_\ell^\top U_\ell. \tag{21}$$

Let J denote the coordinates of the current d-dimensional vector and R the remaining coordinates, with U_{JJ} and U_{JR} denoting the corresponding blocks of U. We use $\bar{w}$ for the current reconstruction target after compensation from previously processed vectors, and q_k for the candidate reconstruction using codeword k under the fixed row-group scale. We reassign the current vector as

$$a_J^* = \arg\min_k \left\| (\bar{w}_J - q_k) U_{JJ}^{-1} \right\|_2^2 . \tag{22}$$

The remaining target is then updated as

$$\bar{w}_R \leftarrow \bar{w}_R - (\bar{w}_J - q_{a_J^*}) U_{JJ}^{-1} U_{JR}. \tag{23}$$

Subsequent vectors are reassigned using this compensated target, allowing their assignments to absorb the preceding quantization error. Codebooks, row-group scales, and the mixed-bit allocation remain fixed during this stage; only the codeword indices are updated.

Action-level calibration. After index reassignment, we freeze the updated codeword indices a^* and optimize only the codebooks and row-group scales to match the BF16 action. Following prior codebook fine-tuning Egiazarian et al. (2024), we denote the codebooks and row-group scales by C and s, respectively. For a calibration sample x and fixed a^*, we minimize

$$\min_{C,s} \mathbb{E}_x \left[\left\| a_Q(x; C, s, a^*) - a_{\mathrm{BF16}}(x) \right\|_2^2 \right] . \tag{24}$$

This directly calibrates the codebooks and scales so that the quantized policy matches the BF16 action. Keeping the indices fixed preserves the storage rate during calibration. Further optimization and validation details are provided in the Appendix C.2.

4 Experiments

4.1 Experimental Setup

We evaluate GR00T N1.7 (NVIDIA, 2025), quantizing 338 linear layers: 112 in the language backbone and 226 in the DiT action head. Outside this scope, 128 additional linear layers and the token embedding are fixed at 4-bit in the quantized models, while activations remain BF16. BF16 serves as the full-precision reference. We compare against HQQ (Badri & Shaji, 2023), GPTQ (Frantar et al., 2023), ActQuant (Akbari et al., 2026), and TWLA (Zhao et al., 2026) under the same language/action quantization scope. We evaluate on LIBERO (Liu et al., 2023), using the Spatial, Object, Goal, and Long suites, and on SimplER (Li et al., 2025), using the Google Robot Visual Matching (VM) and Variant Aggregation (VA) protocols. Full evaluation and quantization settings are provided in Appendix D, with baseline implementation details in Appendix E.

4.2 Results on LIBERO

Table 1 compares SACQ with existing quantization methods across the four LIBERO suites. SACQ achieves 93.85% average success at 1.4998 L/A bpw, only 0.25 percentage points below the 94.10% BF16 reference, while reaching 94.25% and 94.10% at approximately 2 and 3 bpw, respectively. The gap to competing methods becomes pronounced at lower precision: at approximately 2 bpw, TWLA reaches 93.05% compared with 94.25% for SACQ, while at approximately 1.5 bpw ActQuant and GPTQ-Ternary fall to 30.05% and 31.95%, respectively, compared with 93.85% for SACQ.

This low-bit performance also translates into a substantial reduction in model footprint. At 1.4998 L/A bpw, SACQ reduces resident model memory from 5.677 GB to 1.175 GB, a 4.83× reduction, while remaining within 0.25 percentage points of BF16.

4.3 Results on SimplER

Table 2 compares SACQ with existing quantization methods on the SimplER Google Robot benchmark. At approximately 2 L/A bpw, SACQ achieves average success rates of 63.76% on VM and 52.27% on VA, compared with the BF16 references of 58.90% and 53.45%, respectively. SACQ also achieves higher success than the evaluated quantized baselines under both protocols. Further reducing SACQ to 1.4999 L/A bpw decreases success to 57.20% on VM and 44.67% on VA.

Table 1: **Comparison of SACQ with existing quantization methods on GR00T N1.7 across the four LIBERO suites.** We report success rates (%) against HQQ, GPTQ, ActQuant, and TWLA at comparable language/action bit budgets. L/A bpw denotes the parameter-weighted average bits per weight over the 338 language-backbone and action-head candidate layers.

Method	L/A bpw	Spatial	Object	Goal	Long	Avg.	Memory (GB)
BF16	16.0000	94.2	97.6	95.2	89.4	94.10	5.677
HQQ	3.0000	75.0	93.6	76.4	73.8	79.70	1.660
GPTQ	3.0000	**95.2**	**97.4**	**96.6**	85.6	93.70	1.982
ActQuant	3.0011	94.8	96.2	96.4	**90.4**	**94.45**	1.562
SACQ-3b	2.9990	94.8	96.8	**96.6**	88.2	94.10	1.902
HQQ	2.0000	0.0	0.0	0.4	0.0	0.10	1.368
GPTQ	2.0000	57.2	90.2	49.8	60.2	64.35	1.528
ActQuant	2.0006	92.2	96.6	88.6	67.8	86.30	1.306
TWLA	2.0489	93.6	**97.4**	96.4	84.8	93.05	2.772
SACQ-2b	1.9997	**93.8**	96.6	**97.0**	**89.6**	**94.25**	1.310
GPTQ-Ternary	1.5850	39.4	44.4	10.6	33.4	31.95	1.337
ActQuant	1.4986	48.4	35.4	26.6	9.8	30.05	1.169
SACQ-1.5b	1.4998	**93.2**	**97.0**	**96.2**	**89.0**	**93.85**	1.175
SACQ-1.25b	1.2487	92.0	96.8	94.4	87.6	92.70	1.114
SACQ-1b	1.0000	86.8	93.6	86.2	65.6	83.05	1.069

Table 2: **Comparison of SACQ with existing quantization methods on the SimplER Google Robot benchmark using GR00T N1.7.** We report success rates (%) under the Visual Matching (VM) and Variant Aggregation (VA) protocols against GPTQ, ActQuant, and TWLA at comparable language/action bit budgets. L/A bpw denotes the parameter-weighted average bits per weight over the 338 language-backbone and action-head candidate layers.

Method	L/A bpw	Pick	Move	Drawer	Apple	Avg.	Memory (GB)
Visual Matching (VM)							
BF16	16.0000	86.00	91.25	45.37	12.96	58.90	5.677
GPTQ	2.0000	70.33	25.00	6.94	0.00	25.57	1.524
ActQuant	1.9998	90.67	62.92	32.87	0.00	46.61	1.331
TWLA	2.0635	84.00	72.08	47.22	7.41	52.68	2.782
SACQ-2b	1.9995	**91.33**	**89.17**	**49.54**	**25.00**	**63.76**	1.342
SACQ-1.5b	1.4999	84.67	73.75	38.89	31.48	57.20	1.191
Variant Aggregation (VA)							
BF16	16.0000	83.27	81.83	21.69	26.98	53.45	5.677
GPTQ	2.0000	67.88	23.00	2.12	0.00	23.25	1.524
ActQuant	1.9998	76.97	64.33	11.38	2.12	38.70	1.331
TWLA	2.0635	79.88	68.83	14.29	29.63	48.16	2.782
SACQ-2b	1.9995	**80.73**	**77.83**	**17.72**	**32.80**	**52.27**	1.342
SACQ-1.5b	1.4999	75.39	74.17	4.76	24.34	44.67	1.191

5 Analysis and Ablations

We analyze the roles of the three components of SACQ and further examine when structure-aware codebook fitting becomes most important. Unless otherwise stated, all ablations are conducted on LIBERO using the same evaluation protocol as in Section 4.

Table 3 ablates action-aware mixed-bit allocation, structure-aware codebook fitting, and post-quantization refinement. At approximately 1.5 L/A bpw, Euclidean VQ with mixed-bit allocation fails to preserve the policy, whereas structure-aware fitting raises the pre-refinement average success to 47.50%. Post-quantization refinement then recovers the full SACQ model to 93.85%. Remov-

Table 3: **Component and structure-aware ablations of SACQ on LIBERO.** We evaluate the three components at approximately 1.5 L/A bpw and structure-aware fitting at approximately 1.25 L/A bpw.

Setting	Alloc.	Struct.	Refine.	Spatial	Object	Goal	Long	Avg.
BF16	–	–	–	94.2	97.6	95.2	89.4	94.10
1.5 L/A bpw								
Uniform Euclidean VQ	–	–	–	0.0	0.0	0.0	0.0	0.00
+ Mixed-bit allocation	✓	–	–	0.0	0.0	0.0	0.0	0.00
+ Structure-aware fitting	✓	✓	–	44.8	73.4	31.4	40.4	47.50
Full SACQ-1.5b	✓	✓	✓	93.2	97.0	96.2	89.0	93.85
w/o Struct. (Euclidean)	✓	–	✓	95.2	95.8	93.8	85.6	92.60
w/o Allocation (Uniform)	–	✓	✓	94.2	96.8	95.8	86.2	93.25
1.25 L/A bpw								
w/o Struct. (Euclidean)	✓	–	✓	91.8	94.4	91.2	73.2	87.65
Full SACQ-1.25b	✓	✓	✓	92.0	96.8	94.4	87.6	92.70

ing structure-aware fitting or mixed-bit allocation reduces the final success to 92.60% and 93.25%, respectively.

The effect of structure-aware fitting becomes larger under more aggressive compression. At approximately 1.25 L/A bpw, it improves the average success from 87.65% to 92.70%, with the largest gain on LIBERO-Long from 73.2% to 87.6% (+14.4 percentage points). Additional analysis under limited post-quantization refinement is provided in Appendix G.

6 Conclusion

We introduced SACQ for ultra-low-bit VLA quantization. SACQ combines **structure-aware codebook fitting**, which accounts for the functional role of each projection, **action-aware mixed-bit allocation**, which assigns precision according to its effect on the final decoded action, and **two-stage post-quantization refinement** for recovering the quantized policy.

On GR00T N1.7, SACQ maintains performance close to the BF16 reference on LIBERO even at 1.5 L/A bpw, while reducing resident model memory from 5.677 GB to 1.175 GB, a 4.83× reduction. SACQ also maintains strong performance on SimplER at 2 L/A bpw. Our analysis further shows that structure-aware fitting becomes particularly effective under more aggressive compression and limited post-quantization refinement (Appendix G).

Additional experiments demonstrate that this behavior extends beyond the main setting. On OpenVLA-OFT evaluated on LIBERO, SACQ achieves 94.05% average success at 2 V/L bpw, compared with 95.80% for the BF16 reference (Appendix F.1). On GR00T N1.7 evaluated on LIBERO-Pro Zhou et al. (2025), SACQ achieves 43.62% at 2 L/A bpw, compared with 43.67% for BF16 (Appendix F.2). These results indicate that SACQ remains effective across different VLA architectures and under a broader range of perturbations.

Limitations. SACQ is weight-only and retains BF16 activations, with our current implementation focusing on memory efficiency rather than latency optimization. Realizing runtime speedups from the codebook representation requires dedicated optimized kernels, which we leave for future work. Our evaluation is also limited to simulation benchmarks, leaving real-robot validation for future work.

AI Assistance Disclosure

During manuscript preparation, we used large language models for language polishing, proofreading, and improving the clarity of the presentation. All technical content, including research ideas, methodology, experiments, and analyses, was developed and verified by the authors.

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. *arXiv preprint arXiv:2605.24011*, 2026. 2, 7

Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, November 2023. URL https://dropbox.github.io/hqq_blog/. 7

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12284–12303. PMLR, 2024. 2, 7

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 3, 7

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025a. 2

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R 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. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b. 1

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728, 2025. 7, 15

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023. 7, 15

NVIDIA. Isaac gr00t n1.7: Open foundation model for generalized humanoid control, 2025. URL https://huggingface.co/nvidia/GR00T-N1.7-3B. 2, 7

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025. 1

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 1

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 48630–48656, 2024. 2

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024. 1, 3, 18

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 *The Fourteenth International Conference on Learning Representations*, 2026. 2

Xin Yan, Zhenglin Wan, Feiyang Ye, Xingrui Yu, Hangyu Du, Yang You, and Ivor Tsang. HB-VLA: Pushing 1-bit post-training quantization for vision-language-action models. *arXiv preprint arXiv:2602.13710*, 2026. 2

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 39539–39549, 2026. 2

Zhixiong Zhao, Zukang Xu, Zhixuan Chen, Xing Hu, Zhe Jiang, and Dawei Yang. TWLA: Achieving ternary weights and low-bit activations for llms via post-training quantization, 2026. URL https://arxiv.org/abs/2606.13054. 7

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. LIBERO-PRO: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025. 9, 17

A Structure-Aware Codebook Fitting Details

A.1 Metric-Weighted Codebook Optimization

As summarized in Section 2.3, given $G(v)$, we optimize the objective in equation 3 by alternating between two steps, as in k-means. With the codewords fixed, each vector is assigned to the codeword nearest under its own metric,

$$a(v) = \arg\min_j (v - c_j)^\top G(v)(v - c_j), \tag{25}$$

and with the assignments fixed, each codeword is the solution of a weighted least-squares problem over the vectors assigned to it,

$$c_j = \Big(\sum_{v:a(v)=j} G(v) \Big)^+ \sum_{v:a(v)=j} G(v)\, v, \tag{26}$$

where $(\cdot)^+$ denotes the pseudoinverse. Under these exact updates, both steps are non-increasing in equation 3, yielding convergence.

A.2 Language-Backbone Feed-Forward Metric Derivation

We provide the first-order derivation underlying the feed-forward metrics summarized in Section 3.1.

The feed-forward block is given by

$$g = hW_g^\top, \qquad u = hW_u^\top, \qquad z = \mathrm{SiLU}(g) \odot u, \qquad y = zW_d^\top. \tag{27}$$

Here, z is the input to the down projection and y is the FFN output. For the down projection, the input-side Gram is $A = \mathbb{E}[zz^\top]$, and we use $B = I$ as for the output projection.

The gate and up projections require different treatment because their outputs interact nonlinearly through the SwiGLU operation before being passed through the down projection W_d. We therefore approximate their local perturbations to first order. Consider a four-dimensional weight vector v in output channel r, and let $h_v \in \mathbb{R}^4$ denote the input activations corresponding to its four weights. For the gate projection, the change in z_r under a small perturbation δv is approximated as

$$\delta z_r \approx \mathrm{SiLU}'(g_r)\, u_r\, h_v^\top \delta v. \tag{28}$$

This change propagates through the down projection to the FFN output as

$$\delta y = \delta z_r W_d[:, r]^\top, \tag{29}$$

and therefore

$$\|\delta y\|_2^2 = \|W_d[:, r]\|_2^2 (\delta z_r)^2. \tag{30}$$

Substituting the first-order approximation above and taking the expectation over calibration samples gives the local metric

$$G_{\text{gate}}(v) = \|W_d[:, r]\|_2^2 \mathbb{E}\left[(\text{SiLU}'(g_r)\, u_r)^2 h_v h_v^\top\right]. \tag{31}$$

Similarly, for the up projection,

$$G_{\text{up}}(v) = \|W_d[:, r]\|_2^2 \mathbb{E}\left[\text{SiLU}(g_r)^2 h_v h_v^\top\right]. \tag{32}$$

In practice, we retain only the diagonal entries of these 4×4 local metrics to simplify codebook fitting:

$$\widetilde{G}(v) = \text{Diag}\left(G_{11}(v), G_{22}(v), G_{33}(v), G_{44}(v)\right), \tag{33}$$

where $G(v)$ denotes the corresponding gate or up metric above. This approximation retains the downstream sensitivity induced by the down projection while discarding cross-coordinate correlations within each weight vector.

A.3 Action-Head Metric Derivations

The projection roles in the DiT action head follow the structure-aware construction in Section 3.1, with three model-specific differences.

First, each block contains both self-attention and cross-attention, and in cross-attention the query-side input does not coincide with the key/value-side input. We therefore collect the two separately, with Grams

$$A_q = \mathbb{E}[x_q x_q^\top], \qquad A_{kv} = \mathbb{E}[x_{kv} x_{kv}^\top]. \tag{34}$$

The query projection uses A_q as its input-side Gram and the key and value projections use A_{kv}; in self-attention the two coincide. The output-side weighting is obtained from the second moment of the opposite projection, namely $\text{diag}(W_K A_{kv} W_K^\top)$ for the query projection and $\text{diag}(W_Q A_q W_Q^\top)$ for the key projection. For grouped-query attention, each key-side energy is repeated for the query heads mapped to that key/value head, whereas query-side energies are summed over the query heads sharing the same key/value head.

Second, the feed-forward block uses a GELU activation rather than the SwiGLU structure of the language backbone:

$$p = h W_{\text{up}}^\top, \qquad z = \text{GELU}(p), \qquad y = z W_{\text{down}}^\top. \tag{35}$$

Consider a four-dimensional weight vector v in output channel r, and let h_v denote the input activations corresponding to its four weights. Under a small perturbation δv,

$$\delta p_r = h_v^\top \delta v, \qquad \delta z_r \approx \text{GELU}'(p_r)\, h_v^\top \delta v. \tag{36}$$

This change propagates through the down projection as

$$\delta y = \delta z_r W_{\text{down}}[:, r]^\top, \tag{37}$$

and therefore

$$\|\delta y\|_2^2 \approx \|W_{\text{down}}[:, r]\|_2^2 \text{GELU}'(p_r)^2 (h_v^\top \delta v)^2. \tag{38}$$

In our implementation, we use the separable approximation

$$\mathbb{E}\left[\text{GELU}'(p_r)^2 h_v h_v^\top\right] \approx \mathbb{E}\left[\text{GELU}'(p_r)^2\right] \mathbb{E}\left[h_v h_v^\top\right]. \tag{39}$$

The metric used for codebook fitting is therefore

$$G_{\text{up}}(v) = \|W_{\text{down}}[:, r]\|_2^2 \mathbb{E}\left[\text{GELU}'(p_r)^2\right] \mathbb{E}\left[h_v h_v^\top\right]. \tag{40}$$

This approximation discards correlations between the activation sensitivity and input covariance. The down projection uses the Gram of its post-activation input z with $B = I$, as in the language backbone.

Third, the adaptive layer normalization projection requires its own output-side weighting. Let t_{emb} denote the diffusion timestep embedding. The projection receives the post-activation conditioning input $c = \text{SiLU}(t_{\text{emb}})$ and produces the modulation parameters in (scale, shift) order, which are applied as

$$y = \text{LayerNorm}(x) \odot (\mathbf{1} + \text{scale}) + \text{shift}. \tag{41}$$

An error in the scale branch is multiplied by the normalized hidden state as it propagates, whereas an error in the shift branch propagates with unit coefficient. Writing

$$\ell_j = \mathbb{E}\big[\text{LayerNorm}(x)_j^2\big], \tag{42}$$

the output-side weighting is $[\ell; \mathbf{1}]$ and the input-side Gram is $A = \mathbb{E}[cc^\top]$.

A.4 Complete Metric Definitions

For completeness, we summarize the final structure-aware metrics used for all projection roles in the language backbone and DiT action head.

Table 4: **Metric construction for the seven language backbone roles.** Head and group indices are omitted for readability; the actual computation applies the corresponding normalization, RoPE, and GQA repetition or aggregation.

Projection	Input-side Gram A	Output-side weighting B_{rr}
q proj	$\mathbb{E}[xx^\top]$	$\mathbb{E}[K_r^2]$
k proj	$\mathbb{E}[xx^\top]$	$\mathbb{E}[Q_r^2]$
v proj	$\mathbb{E}\Big[\sum_s \omega_s x_s x_s^\top\Big], \quad \omega_s = \sum_{h,t} P_{h,t,s}^2$	$\|W_O[:, r]\|_2^2$
o proj	$\mathbb{E}[C^\top C], \quad C = PV$	1
down proj	$\mathbb{E}[zz^\top], \quad z = \text{SiLU}(g) \odot u$	1
Direct diagonal metric		
gate proj	$G(v) = \|W_d[:, r]\|_2^2 \text{Diag}\big(\mathbb{E}\big[(\text{SiLU}'(g_r)u_r)^2 (h_v \odot h_v)\big]\big)$	
up proj	$G(v) = \|W_d[:, r]\|_2^2 \text{Diag}\big(\mathbb{E}\big[\text{SiLU}(g_r)^2 (h_v \odot h_v)\big]\big)$	

Table 5: **Metric construction for the seven action head roles.** Here, x_q and x_{kv} are the query-side and key/value-side attention inputs, which coincide in self-attention. The two output projections, proj_out_1 and proj_out_2, are not among these roles and use Euclidean vector k-means.

Projection	Input-side Gram A	Output-side weighting B_{rr}
Q	$A_q = \mathbb{E}[x_q x_q^\top]$	$\text{diag}(W_K A_{kv} W_K^\top)_r$
K	$A_{kv} = \mathbb{E}[x_{kv} x_{kv}^\top]$	$\text{diag}(W_Q A_q W_Q^\top)_r$
V	$\mathbb{E}\Big[\sum_s \omega_s x_{kv,s} x_{kv,s}^\top\Big], \quad \omega_s = \sum_{h,t} P_{h,t,s}^2$	$\|W_O[:, r]\|_2^2$
O	$\mathbb{E}[C^\top C], \quad C = PV$	1
FF-up	$\mathbb{E}[hh^\top]$	$\mathbb{E}\Big[\text{GELU}'\Big((hW_{\text{up}}^\top)_r\Big)^2\Big] \|W_{\text{down}}[:, r]\|_2^2$
FF-down	$\mathbb{E}[zz^\top], \quad z = \text{GELU}(hW_{\text{up}}^\top)$	1
AdaLN	$\mathbb{E}[cc^\top], \quad c = \text{SiLU}(t_{\text{emb}})$	$[\ell; \mathbf{1}]_r, \quad \ell_j = \mathbb{E}\big[\text{LayerNorm}(x)_j^2\big]$

A.5 Fitting Configuration

All codebook fitting is performed on the scale-normalized weight vectors defined in Section 2.2. We apply the structure-aware metric in this normalized space without additionally weighting it by the squared block scale s^2. For each layer, we fit the codebook on a sampled subset of weight vectors and, once the codebook is fixed, recompute the assignments over all vectors in that layer.

We use different fitting budgets for the language backbone and DiT action head. The language-backbone codebooks use at most 32 calibration requests, with up to 2,048 cached tokens per layer, 50,000 sampled weight vectors, and 8 Lloyd iterations. The DiT codebooks use 32 calibration requests per suite, 200,000 sampled weight vectors, and 12 Lloyd iterations.

Codewords are initialized from sampled weight vectors. We add a ridge term of 10^{-4} to each codeword update for numerical stability and clip the updated codewords to $[-1, 1]$.

B Action-Aware Mixed-Bit Allocation Details

B.1 Iterative Allocation Procedure

Algorithm 1 Action-aware mixed-bit allocation

1: Initialize every candidate layer at GPTQ4.
2: **while** the calculated bits per weight exceed the target budget **do**
3: Periodically measure the one-step action-error increases Δ_ℓ, averaged over the calibration set with equal weighting across suites.
4: Compute the calculated bits per weight of each component.
5: Select the component with the higher calculated bits per weight.
6: Within it, choose $\ell^\star$ by equation 19.
7: Move $\ell^\star$ to its next lower-precision state.
8: **end while**

The measurements Δ_ℓ are refreshed after a fixed number of accepted moves, making the procedure a periodically refreshed greedy search rather than a globally optimal allocation.

B.2 Allocation Configuration

We evaluate allocation sensitivity using 10 fixed calibration requests from each of the four LIBERO suites, giving 40 requests in total. The BF16 reference action for each request is generated using the corresponding suite-specific checkpoint. Let A_i denote the final decoded action tensor produced by the quantized policy for request i, and let A_i^{BF16} denote its BF16 reference. We define the request-level relative action error as

$$e_i = \frac{\|A_i - A_i^{\mathrm{BF16}}\|_F}{\max\left(\|A_i^{\mathrm{BF16}}\|_F, 10^{-9}\right)}. \tag{43}$$

The norm is taken over the full decoded action tensor of each request. The current allocation error is then

$$E = \frac{1}{40} \sum_{i=1}^{40} e_i. \tag{44}$$

Since each suite contributes the same number of requests, this is equivalent to equally weighting the four suites.

Let $E_\ell^{\downarrow}$ denote the average relative error when only layer ℓ is moved to its next lower-precision state. We define its sensitivity as

$$\Delta_\ell = E_\ell^{\downarrow} - E. \tag{45}$$

Thus, Δ_ℓ measures the increase in final decoded-action relative error caused by a one-step precision reduction of layer ℓ. We use the same policy-noise seed when computing these action errors.

Sensitivities are refreshed every 100 accepted moves. Allocation checkpoints are saved when nominal-bpw thresholds spaced by 0.10 are crossed, and candidate layers transition only between adjacent quantization states.

C Post-Quantization Refinement Details

C.1 Second-Order Index Reassignment

For each LIBERO suite, we use all 576 training requests to collect layer-input statistics, excluding the 64 validation requests. With the quantized policy frozen after structure-aware fitting and mixed-bit allocation, we collect the full input second-moment matrix H_ℓ from all input rows of each target layer. For the DiT action head, inputs from all denoising calls are included.

For numerical stability, we use

$$\widetilde{H}_\ell = \frac{1}{2}(H_\ell + H_\ell^\top) + 0.1\,\frac{\mathrm{tr}(H_\ell)}{n_\ell} I, \qquad \widetilde{H}_\ell^{-1} = U_\ell^\top U_\ell. \tag{46}$$

where n_ℓ denotes the input dimension of layer ℓ. The reconstruction target is the original BF16 weight. During index reassignment, weight vectors are processed sequentially, while the codebooks, row-group scales, mixed-bit allocation, and all other model parameters remain fixed. Only the codeword indices are updated.

C.2 Action-Level Calibration

After index reassignment, we freeze the updated codeword indices and optimize only the codebooks and row-group scales. As described in Section 3.3, training minimizes the MSE between the final decoded actions of the quantized policy and the BF16 reference. For stochastic action generation, we use the same policy-noise seed for the quantized and BF16 policies when computing the action loss.

We use the same calibration configuration for GR00T-LIBERO and OFT-LIBERO. We use AdamW with codebook and scale learning rates of 10^{-4} and 10^{-5}, respectively, zero weight decay, and no learning-rate scheduler or warm-up. For each suite, we use 576 training requests and 64 validation requests with a global batch size of 6. Calibration runs for 12 epochs, corresponding to 1,152 optimizer updates, with gradient-norm clipping at 1.0.

For SimplER Google, we use the same optimization procedure with codebook and scale learning rates of 5×10^{-5} and 5×10^{-6}, respectively. We use 512 training requests and 128 validation requests with a global batch size of 8, and train for 12 epochs, corresponding to 768 optimizer updates. In all settings, only the codebooks and scales are optimized; the updated indices and all other model parameters remain fixed.

We select the checkpoint with the lowest mean relative action error on the validation set,

$$\mathcal{P}_{\mathrm{val}} = \frac{1}{|\mathcal{D}_{\mathrm{val}}|} \sum_{i \in \mathcal{D}_{\mathrm{val}}} \frac{\|\widehat{y}_i - y_i\|_F}{\max(\|y_i\|_F, 10^{-9})}, \tag{47}$$

where $\mathcal{D}_{\mathrm{val}}$ denotes the validation set, and y_i and $\widehat{y}_i$ are the final decoded actions of the BF16 and quantized policies, respectively.

D Experimental and Quantization Details

D.1 Evaluation Protocols

We use LIBERO (Liu et al., 2023) and SimplER (Li et al., 2025) to evaluate the quantized policies and follow the official evaluation protocols and action-execution settings.

For LIBERO, we evaluate the Spatial, Object, Goal, and Long suites, each consisting of 10 tasks with 50 official initial states per task, resulting in 500 episodes per suite and 2,000 episodes per model configuration. All evaluations use environment seed 17, and at each policy inference step we execute the first eight actions of the predicted action chunk.

For SimplER, we evaluate the Google Robot embodiment using the Visual Matching (VM) and Variant Aggregation (VA) protocols. VM comprises 864 evaluation conditions and VA 1,992, with both protocols covering four task categories: Pick, Move, Drawer, and Apple. All evaluations use seed 7, and at each policy inference step we execute only the first action of the predicted action chunk.

D.2 Quantization Scope and Bit Reporting

For GR00T N1.7, the quantization candidate set contains 338 linear layers: 112 in the language backbone and 226 in the DiT action head. Outside this candidate scope, 128 additional linear layers and the token embedding are fixed at 4-bit precision in the quantized models. Of these 128 linear layers, 104 belong to the vision encoder/projector. We observed that the vision encoder is sensitive to sub-4-bit quantization and therefore keep it at 4-bit throughout our experiments. The vision encoder is quantized to 4-bit using GPTQ, while the remaining fixed components use 4-bit affine quantization. Activations remain BF16, and BF16 serves as the full-precision reference.

All baselines are applied to the same language/action candidate scope. The L/A bpw reported in the main text and tables denotes the parameter-weighted average bits per weight over these 338 candidate layers and excludes metadata such as codebooks and scales. In contrast, the reported memory footprint is measured for the full loaded policy.

E Baseline Implementation Details

HQQ. We apply uniform HQQ to all 338 language/action candidate layers with group size 64, evaluating W3 and W2 configurations. Activations and computation remain in BF16.

GPTQ. We apply GPTQ to the same 338 candidate layers with group size 64 and block size 128. For an additional ultra-low-bit comparison, we extend the same GPTQ port to symmetric ternary codes $\{-1, 0, +1\}$ with group-64 scaling and GPTQ-style error compensation. This setting has a nominal rate of $\log_2(3) \approx 1.585$ bpw and is a custom ternary extension rather than a standard GPTQ mode.

TWLA-compatible. We port the W1.58A16 configuration of TWLA to all 338 candidate layers while keeping activations in BF16. However, our PyTorch VLA port does not directly reproduce the compact llama.cpp inference path and some of the storage/runtime optimizations described in the original work. Consequently, the resident memory reported for our TWLA port may be higher than what a more optimized TWLA implementation could achieve. In addition, one language-backbone projection produces non-finite values during quantization, for which we use an FP64 fallback. We therefore refer to this baseline as TWLA-compatible.

ActQuant. We apply ActQuant's inter-tensor bit allocation and intra-tensor scale optimization to the same 338 language/action candidate layers. Since ActQuant does not directly target GR00T's DiT action head, we apply its procedure to the DiT layers to match the quantization scope used in our comparisons.

F Additional Experiments

We conduct two additional evaluations beyond the main experiments. First, we apply SACQ to OpenVLA-OFT and evaluate the quantized model on LIBERO to examine whether its low-bit behavior extends to a different VLA architecture. Second, we evaluate SACQ-quantized GR00T N1.7 on LIBERO-Pro to examine whether the quantized policy remains close to its BF16 reference under a broader range of perturbations.

F.1 Cross-Architecture Evaluation on OpenVLA-OFT

We evaluate OpenVLA-OFT on the four LIBERO suites using 50 official initial states per task, resulting in 500 episodes per suite. We use program and policy seed 17, environment seed 17, and execute an action chunk of length 8. Unlike the GR00T setting, the quantization scope contains 420 linear layers: 196 in the vision encoder and 224 in the language backbone, while the action head is excluded. We therefore report V/L bpw for OpenVLA-OFT, which is not directly comparable to the L/A bpw reported for GR00T in the main experiments.

Table 6: **Cross-architecture evaluation on OpenVLA-OFT.** Success rates (%) are reported across the four LIBERO suites. V/L bpw denotes the quantization rate over the 420 vision and language linear layers. Bold indicates the best result within each comparable bit-budget group.

Method	V/L bpw	Spatial	Object	Goal	Long	Avg.	Memory (GB)
BF16	16.0000	93.8	98.2	97.2	94.0	95.80	15.203
SACQ-3b	2.9988	91.8	97.4	97.8	95.2	**95.55**	5.128
GPTQ	2.0000	67.6	55.8	73.6	49.6	61.65	3.625
ActQuant	2.0000	79.6	60.8	77.2	24.8	60.60	2.953
TWLA	2.1115	91.0	**96.4**	96.2	**93.8**	**94.35**	8.960
SACQ-2b	2.0000	**92.0**	96.0	**98.0**	90.2	94.05	2.926
GPTQ-Ternary	1.5850	0.0	0.0	0.0	0.0	0.00	3.026
ActQuant	1.5000	61.6	31.2	35.2	0.0	32.00	2.433
SACQ-1.5b	1.4989	**89.4**	**95.0**	**92.8**	**65.6**	**85.70**	2.463

As shown in Table 6, SACQ achieves 95.55% average success at 3 V/L bpw, only 0.25 percentage points below the 95.80% BF16 reference. At 2 V/L bpw, SACQ retains 94.05% average success, similar to the 94.35% obtained by TWLA and substantially higher than the evaluated GPTQ and ActQuant variants. These results show that the low-bit quantization performance of SACQ extends beyond GR00T N1.7 to a different VLA architecture.

F.2 Additional Evaluation on LIBERO-Pro

We additionally evaluate SACQ on LIBERO-Pro Zhou et al. (2025) using GR00T N1.7. LIBERO-Pro includes five perturbation families: Language, Object, Swap, Task, and Environment. Across the four LIBERO suites, each model is evaluated over 10,000 episodes in total.

Table 7: **Evaluation of GR00T N1.7 on LIBERO-Pro.** Success rates (%) are aggregated over the five perturbation families for each LIBERO suite.

Method	L/A bpw	Spatial	Object	Goal	Long	Avg.
BF16	16.0	44.60	41.28	45.52	43.28	43.67
SACQ-3b	3.0	44.44	41.04	45.56	44.00	43.76
SACQ-2b	2.0	44.60	41.08	44.48	44.32	43.62
SACQ-1.5b	1.5	43.64	40.72	43.24	42.88	42.62

As shown in Table 7, BF16 achieves 43.67% average success, while SACQ-3b, SACQ-2b, and SACQ-1.5b achieve 43.76%, 43.62%, and 42.62%, respectively. The 3-bit and 2-bit models remain nearly identical to the BF16 reference in overall success, while the 1.5-bit model also maintains competitive performance. These results show that SACQ maintains performance close to the BF16 reference even under a range of perturbations.

G Structure-Aware Fitting under Limited Refinement

We further examine the role of structure-aware codebook fitting when post-quantization refinement is performed under a limited refinement budget. At approximately 1.5 L/A bpw, we compare three codebook-fitting objectives while keeping the mixed-bit allocation fixed: structure-aware fitting, layer-local reconstruction fitting, and Euclidean fitting. The layer-local baseline fits codebooks according to the reconstruction error at the output of each individual projection,

$$\mathcal{L}_{\text{local}} = \left\| X(W - \widehat{W})^\top \right\|_F^2, \tag{48}$$

without modeling the downstream operation in which that projection output is subsequently used. This layer-local objective follows the reconstruction criterion commonly used in prior vector-

quantization methods such as GPTVQ (Van Baalen et al., 2024). The remaining quantization and refinement pipeline is kept identical across all settings, and refinement is performed for 12 epochs.

Table 8: **Codebook fitting under limited post-quantization refinement at approximately 1.5 L/A bpw.** We vary the calibration data per LIBERO suite while fixing refinement to 12 epochs. Δ denotes the difference in four-suite average success relative to structure-aware fitting under the same calibration budget.

Calib. data	Fitting	Spatial	Object	Goal	Long	Avg.	Δ vs. Struct.
–	BF16	94.2	97.6	95.2	89.4	94.10	–
90	Structure-aware	**76.2**	94.6	**88.0**	**77.8**	**84.15**	0.00
	Layer-local	61.8	**95.0**	71.8	69.6	74.55	-9.60
	Euclidean	42.2	48.6	35.0	33.2	39.75	-44.40
180	Structure-aware	**86.0**	95.8	**90.6**	**83.0**	**88.85**	0.00
	Layer-local	78.8	**96.4**	87.6	73.4	84.05	-4.80
	Euclidean	48.2	87.4	52.0	41.0	57.15	-31.70
360	Structure-aware	**91.2**	96.6	**94.0**	**81.0**	**90.70**	0.00
	Layer-local	87.0	**97.2**	88.2	80.0	88.10	-2.60
	Euclidean	78.4	95.2	77.2	63.4	78.55	-12.15
720	Structure-aware	**91.8**	**96.2**	93.0	84.0	91.25	0.00
	Layer-local	90.6	95.8	**94.2**	**85.2**	**91.45**	+0.20
	Euclidean	87.8	94.2	88.4	75.4	86.45	-4.80
1440	Structure-aware	93.2	**97.6**	93.8	85.6	92.55	0.00
	Layer-local	92.6	97.4	**95.0**	**86.8**	**92.95**	+0.40
	Euclidean	**93.6**	**97.6**	91.6	79.6	90.60	-1.95
2880	Structure-aware	93.2	**97.0**	**96.2**	**89.0**	**93.85**	0.00
	Layer-local	92.6	**97.0**	95.0	85.8	92.60	-1.25
	Euclidean	**95.2**	95.8	93.8	85.6	92.60	-1.25

As shown in Table 8, differences among the codebook-fitting objectives are most pronounced at smaller calibration budgets. With only 90 calibration observations per suite, structure-aware fitting achieves 84.15% average success, compared with 74.55% for layer-local reconstruction fitting and 39.75% for Euclidean fitting. These differences narrow rapidly as the calibration data increase. At 720 and 1,440 observations, layer-local reconstruction fitting achieves averages only 0.20 and 0.40 percentage points higher than structure-aware fitting, respectively, leaving the two methods numerically close. With the full 2,880 observations, structure-aware fitting achieves the highest average success of 93.85%. On LIBERO-Long, it recovers to 89.0%, close to the 89.4% BF16 reference, whereas layer-local reconstruction and Euclidean fitting reach 85.8% and 85.6%, respectively.

Because refinement is fixed to 12 epochs in all settings, reducing the calibration data also reduces both the number of optimization updates and the amount of input statistics available for index reassignment. Under this setting, the results indicate that the main benefit of structure-aware fitting is not uniformly higher final average performance at every calibration budget, but a stronger low-bit initialization when subsequent refinement is limited. As the refinement budget increases, the differences in overall average success become smaller; however, in the full-budget setting, structure-aware fitting recovers LIBERO-Long to 89.0%, closest to the 89.4% BF16 reference. This suggests that even when the overall averages become similar, structure-aware fitting can preserve LIBERO-Long performance closer to the BF16 reference.