SPHQuant: Efficient Extreme Low Bit Weight Quantization for Vision-Language Models

Kewei Zhang[1] **Zheng Chen**[1] **Haotong Qin**[2] **Yulun Zhang**[1,*]
[1]Shanghai Jiao Tong University
[2]The Hong Kong Polytechnic University

Abstract

Recent foundation models are moving toward native multimodal Vision-Language Models (VLMs), making VLMs a central form of next-generation foundation models. However, their large language backbones make edge deployment difficult due to high memory footprint and memory-bound autoregressive decoding. Weight-only post-training quantization is a practical solution, but pushing VLMs to extreme low bit-widths remains challenging: existing rotation-free methods suffer from outliers at 2–3 bits, while rotation-based methods improve accuracy at the cost of additional runtime overhead. We propose SPHQuant, a rotation-free spherical weight-only quantization framework for VLMs. Instead of quantizing weights directly in Cartesian coordinates, SPHQuant decomposes each 8D weight vector into coordinate signs, radius, and a positive unit direction. This representation isolates outlier magnitude into the radius while keeping directions bounded and statistically regular. Based on this insight, SPHQuant allocates extra precision to the radius to mitigate accuracy degradation induced by outliers. It further uses a compact positive-direction codebook and fine-tunes codebook entries through angular parameterization to preserve the unit-sphere constraint. We also design a hardware-friendly GEMV kernel that keeps the direction codebook small enough for shared-memory lookup and packs radial bits efficiently. Experiments show that SPHQuant matches the performance of state-of-the-art extreme low-bit quantization methods while improving decode throughput over QTIP by **30.3%** on RTX A6000. Code will be released at `https://github.com/Pushazf/SPHQuant`.

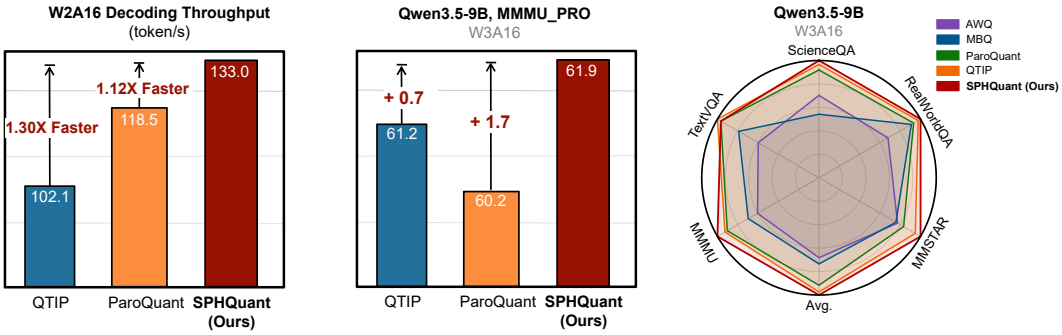
1 Introduction

Recent foundation models are moving beyond text-only language models toward unified vision-language foundation models [11, 21, 1]. Rather than treating vision modules as add-ons to text-only LLMs, recent models jointly learn from vision and text tokens during training, making multimodal understanding a native capability. This shift makes VLMs an important model family and creates a growing need for edge deployment. Many VLM applications, such as mobile assistants [6, 7], on-device OCR, document understanding, embodied agents, and real-time visual reasoning, require low latency, privacy preservation, and reduced dependence on network connectivity. However, deploying VLMs on edge devices remains challenging: their billion-scale parameters require large device memory, while memory-bound autoregressive inference limits decode throughput.

Quantization has been explored across visual tasks, including image demoiréing [5] and video super-resolution [2, 29]. For VLM deployment, weight-only post-training quantization (PTQ) provides a practical way to reduce weight storage and memory traffic during autoregressive decoding. Although 4-bit weight-only quantization has been widely adopted [15], pushing VLMs to more aggressive

*Corresponding author: Yulun Zhang, `yulun100@gmail.com`.

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



(a) Decoding Throughput **(b)** Long Reasoning Accuracy **(c)** Standard Task Accuracy

Figure 1: Performance of SPHQuant. (a) W2A16 decoding throughput. (b) Long-reasoning accuracy. (c) W3A16 task accuracy, normalized by the per-task maximum; radial range: 0.60–1.00.

2 or 3 bit settings remains necessary for fitting larger multimodal models on memory-constrained devices. Existing methods, however, still struggle to balance accuracy and efficiency at such extreme bit-widths. QTIP achieves state-of-the-art 2 bit weight-only quantization with randomized-Hadamard incoherence processing [25], but its transforms introduce additional inference operations; ParoQuant reduces rotation overhead through pairwise Givens rotations [14], but its accuracy rapidly degrades at 2–3 bits due to the limitations of linear quantization. Preserving outlier weights in FP16 can improve accuracy but requires hardware-unfriendly sparse-dense computation.

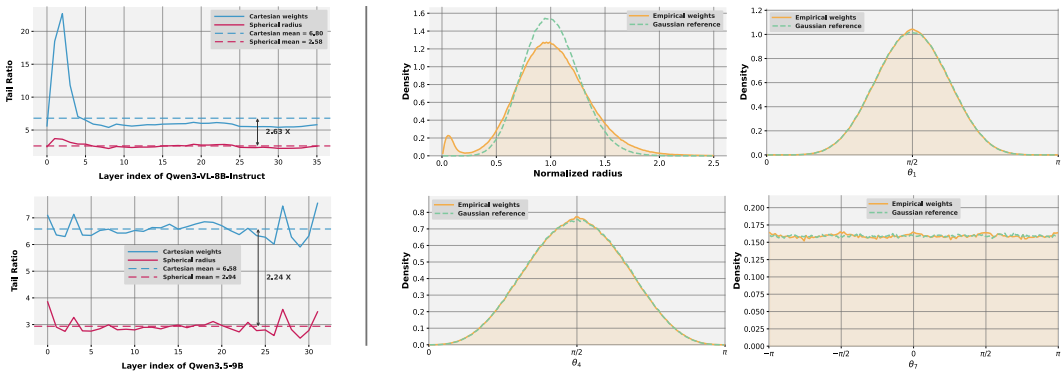
This motivates a question: **can we handle weight outliers with low overhead under extreme low-bit VLM quantization?** We argue that the difficulty is not only caused by the limited bit-width, but also by the coordinate system in which weights are represented and quantized.

Most existing weight-only quantization methods operate directly in Cartesian coordinates. In this representation, a single weight outlier can expand the quantization range of the entire group, reducing the effective information density assigned to non-outlier weights. This effect is tolerable at moderate bit-widths but especially damaging at 2 or 3 bits.

Instead of quantizing weights directly in Cartesian space, we propose to represent each 8D weight vector in spherical coordinates, where the radius and direction are explicitly decoupled. Given a weight vector $\mathbf{w} = \{w_i \mid i = 1, \ldots, k\} \in \mathbb{R}^k$ with $k = 8$, we transform it into its spherical representation $\mathbf{w}' = \{\phi_1, \phi_2, \ldots, \phi_{k-1}, r\}$, where

$$r = \|\mathbf{w}\|_2, \quad \phi_i = \arccos\left(\frac{w_i}{\sqrt{\sum_{j=i}^{k} w_j^2}}\right), \quad i = 1, \ldots, k-2, \quad \phi_{k-1} = \operatorname{atan2}(w_k, w_{k-1}). \tag{1}$$

In this representation, the large values caused by outliers are captured by the radial component, while the directional component is constrained to a unit-sphere manifold. Two empirical observations motivate quantization in spherical coordinates and lead to a radial bit-allocation strategy. **First,** the radius is smoother than the original weights. For example, $(1, 1, 1, 1, 1, 1, 1, 1)$ is transformed into $(2\sqrt{2}, \theta_1, \ldots, \theta_7)$, while $(1, 1, 1, 1, 5, 1, 1, 1)$ is transformed into $(4\sqrt{2}, \theta'_1, \ldots, \theta'_7)$; the coordinate range increases from 1 to 5, but the radius only doubles. We quantify the tail ratio of weights in Cartesian coordinates and radii in spherical coordinates by dividing the 99.9th-percentile value by the 50th-percentile value. As shown in Fig. 2a, across different layers, the radius distribution is consistently smoother than the raw weight distribution, making spherical quantization more robust to outliers. **Second,** even without rotation, the direction is already close to uniform. We decompose each 8D weight vector into seven angles and compare their distributions with the angle distributions of random Gaussian vectors, which correspond to uniformly sampled unit directions. As shown in Fig. 2b, the angle of empirical directions closely match that of uniform-sphere reference, suggesting that directions of weights are close to uniform, and a direction codebook is sufficient for direction quantization. **Consequently,** since outlier magnitudes are isolated in the radius, we can allocate more bits to the radial component. For example, under 2-bit quantization, assigning 8 sign bits, a



(a) Layer-wise Tail Ratios

(b) Empirical and Gaussian Reference Distributions

Figure 2: Observations that support the advantages of quantizing weights in spherical coordinates. (a) Layer-wise Tail Ratios. Spherical radius has significantly smaller tail ratios than Cartesian weights across all layers. (b) Empirical and Gaussian Reference Distributions. The weights exhibit an angular distribution that closely matches the Gaussian reference distribution, while their radial distribution deviates substantially.

6-bit positive-direction index, and 4 radius bits to each 8D vector adds 12.5% to the encoded weight payload while further reducing the impact of outliers.

Based on these insights, we propose SPHQuant, a rotation-free spherical weight-only quantization framework for VLMs. We further design an efficient codebook fine-tuning method and a hardware-friendly CUDA kernel. Specifically, direct fine-tuning of 8D unit vectors in direction codebook requires projecting updates back to the unit sphere, which can discard useful update components and reduce optimization efficiency. We therefore parameterize direction codebook entries by angles to preserve the unit-sphere constraint by construction, and densely pack extra radial bits across 8D vectors for regular CUDA loads and fused lookup, dequantization, and matrix multiplication.

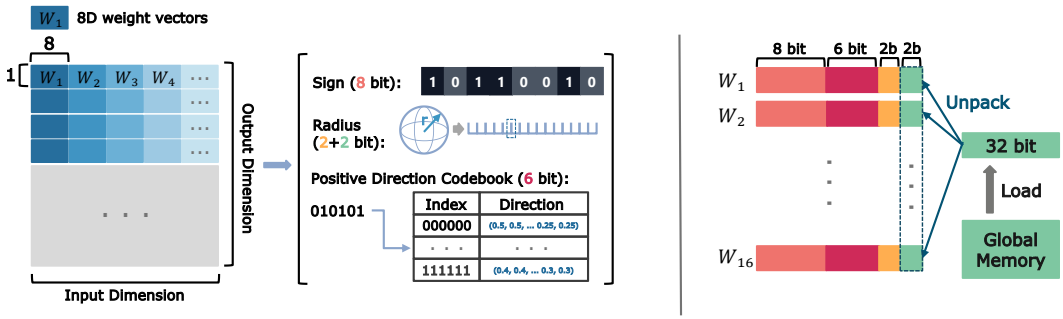
As shown in Fig. 1, SPHQuant achieves competitive accuracy on multiple recent VLMs under extreme low-bit weight-only quantization, while improving decode throughput over QTIP by 30.3% on RTX A6000. In summary, our contributions include:

- We identify a rotation-free spherical representation for extreme low-bit VLM quantization, isolating outlier magnitude in the radius while keeping directions close to uniform.
- We propose SPHQuant, which decouples each 8D weight vector into radius and direction, compactly quantizes directions, and allocates extra bits to the radius for outlier robustness.
- We design angle-parameterized codebook fine-tuning and a CUDA-friendly kernel with dense radial-bit packing and fused dequantization for efficient inference.
- We demonstrate competitive extreme low-bit accuracy across recent VLMs and a 30.3% decoding-throughput improvement over QTIP on RTX A6000.

2 Related work

Vision-language models. Modern LLM-based VLMs combine visual perception with language backbones for multimodal understanding and reasoning [27, 22]. As VLMs evolve from adapter-based architectures toward native multimodal foundation models such as Qwen3.5 [21], Gemma4 [8], they enable applications including mobile assistants, OCR, document understanding, embodied agents, and real-time visual reasoning. However, their language backbones dominate parameter storage and decode-stage memory traffic, making weight-only PTQ important for deployment. Extreme 2- and 3-bit VLM quantization remains challenging because it must preserve multimodal accuracy while achieving real speedups with efficient low-bit kernels.

Weight-only post-training quantization. Weight-only PTQ reduces model memory and decode traffic without full retraining. The simplest baseline is round-to-nearest (RTN) linear quantization,



(a) Quantization in Spherical Coordinates **(b)** Packing extra bits of Radius

Figure 3: Overview of SPHQuant. (a) In **W2A16** setting, each 8D weight vector is quantized in spherical coordinates by decomposing it into signs, radius, and direction. The signs are stored exactly with 8 bits, the radius is linearly quantized with 4 bits including 2 extra bits, and the direction is quantized using a codebook of positive unit vectors. (b) Packed extra radius bits. The extra 2 radius bits from every 16 8D weight vectors are packed and loaded as one 32-bit word.

which maps a weight group to b-bit integer codes with a uniform scale and zero point:

$$Q(\mathbf{w}) = \text{clip}\left(\left\lfloor \frac{\mathbf{w}}{s} \right\rceil + z, 0, 2^b - 1\right), \quad s = \frac{\max(\mathbf{w}) - \min(\mathbf{w})}{2^b - 1}, \quad z = -\left\lfloor \frac{\min(\mathbf{w})}{s} \right\rceil. \tag{2}$$

Scalar methods such as GPTQ [9] and AWQ [15] are hardware-friendly at 4 bits, but become fragile near 2 bits due to outliers and limitation of linear quantization. Transform and vector-code methods improve ultra-low-bit accuracy: QuIP/QuIP# use incoherence processing [3, 24], QTIP uses trellis-coded quantization [25], and ParoQuant reduces rotation overhead through pairwise rotations [14]. Direction–magnitude decomposition has also been explored in PVQ [26] and PCDVQ [33]. PVQ combines Hadamard coherence processing with an implicit integer-lattice representation of directions and separate gain quantization. PCDVQ uses randomized Hadamard regularization and distribution-aligned direction and magnitude codebooks. SPHQuant instead quantizes the original vectors without rotation, adapts its direction codebook through angular fine-tuning, and allocates extra precision to the radius to handle large magnitudes. Our contribution lies in this combination of representation, bit allocation, optimization, and efficient decoding rather than the direction–magnitude decomposition alone. VLM-specific methods such as MBQ [13] and VEQ [20] model modality sensitivity or expert heterogeneity, but they mainly improve calibration objectives under conventional linear quantization.

3 Method

In this section, we introduce SPHQuant, a novel weight-only quantization framework for VLMs. We first describe how SPHQuant performs spherical weight quantization and fine-tunes the quantization parameters. Then, we explain how to implement an efficient CUDA kernel for SPHQuant.

3.1 Quantization in spherical coordinates

SPHQuant applies spherical quantization to the weights of selected linear layers in the VLM language backbone. Given a weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, we split each row along the input dimension into contiguous 8D vectors $\mathbf{w} \in \mathbb{R}^8$. Each vector is decomposed into coordinate signs, radius, and nonnegative unit direction:

$$\mathbf{s} = \text{sign}(\mathbf{w}), \quad \mathbf{w}_+ = |\mathbf{w}|, \quad r = \|\mathbf{w}_+\|_2, \quad \mathbf{d} = \frac{\mathbf{w}_+}{\max(r, \epsilon)}. \tag{3}$$

Here $\mathbf{w}_+$ denotes the positive-orthant version of $\mathbf{w}$, $\mathbf{s} \in \{-1, +1\}^8$, $r \geq 0$, and $\mathbf{d} \in \mathbb{S}^7_+$, where $\mathbb{S}^7_+ = \{\mathbf{u} \in \mathbb{R}^8_{\geq 0} : \|\mathbf{u}\|_2 = 1\}$ is the positive orthant of the 8D unit sphere. Reconstruction gives

$$\hat{\mathbf{w}} = \hat{r} \cdot \mathbf{s} \odot \hat{\mathbf{d}}. \tag{4}$$

where $\hat{r}$ is the dequantized radius, $\hat{\mathbf{d}}$ is the dequantized positive unit direction, and $\mathbf{s}$ is the sign vector extracted during quantization. For each 8D vector, $\mathbf{s}$ can be stored exactly with 8 sign bits and remains unchanged during quantization. We next describe radius and positive-direction quantization and dequantization, and the efficiency benefit of storing signs separately.

For the positive unit direction, SPHQuant uses a codebook $\mathcal{C} = \{\mathbf{c}_k\}$ with $\mathbf{c}_k \in \mathbb{S}^7_+$. Let B be the target weight bit-width. Since the spherical direction has seven degrees of freedom, we allocate $7B$ bits to direction-related information. The coordinate signs are stored separately with 8 bits per 8D vector, leaving a $K_{\text{dir}} = 7B - 8$ bit index for the positive-direction codebook. Thus $|\mathcal{C}| = 2^{K_{\text{dir}}} = 2^{7B-8}$. Separating signs is important for efficient lookup: with sign separation, the codebook only needs to cover positive unit directions, whereas a signed-direction codebook would need an additional factor of 2^8 entries to represent all coordinate sign patterns. In the 2-bit setting, i.e., $B = 2$, the direction codebook has 2^6 entries. Since each entry stores an 8D unit vector in fp16, the sign-separated codebook requires 1 KiB, while the signed-direction codebook requires 2^8 times more storage, or 256 KiB. Since modern GPUs provide only a few hundreds of KiB of shared memory per SM, and prior work also emphasizes avoiding large lookup tables [25], sign separation makes the direction codebook small enough for shared-memory lookup in the 2-bit setting.

Because the directions are empirically close to uniformly distributed on the unit sphere, spherical k-means can be viewed as a data-adaptive refinement of a uniform spherical codebook: it preserves broad directional coverage while adapting the centers to the actual layer weights. We sample normalized directions with probability proportional to r_i^2, so high-magnitude vectors have stronger influence on the codebook, and assign each direction index by cosine similarity:

$$z_i = \arg \max_{k \in \{1,\dots,2^{K_{\text{dir}}}\}} \mathbf{d}_i^\top \mathbf{c}_k. \tag{5}$$

Here z_i is the direction index assigned to the i-th 8D weight vector, and $\hat{\mathbf{d}}_i = \mathbf{c}_{z_i}$ is its dequantized positive unit direction. Since both $\mathbf{d}_i$ and $\mathbf{c}_k$ are unit vectors, maximizing their inner product is equivalent to choosing the codebook direction with the smallest angular distance. Given a fixed direction index, SPHQuant does not directly quantize the original radius r_i. Instead, it first fits a pre-quantization radius $\tilde{r}_i$ along the assigned direction:

$$\tilde{r}_i = r_i^{\text{LS}} = \arg\min_{\alpha \geq 0} \|(\mathbf{w}_i)_+ - \alpha \mathbf{c}_{z_i}\|_2^2 = \langle (\mathbf{w}_i)_+, \mathbf{c}_{z_i} \rangle, \tag{6}$$

where $\langle \cdot, \cdot \rangle$ is the Euclidean inner product; the last equality holds because $\mathbf{c}_{z_i}$ is a unit vector. Thus $\tilde{r}_i$ is the input to radius quantization, while $\hat{r}_i$ is its dequantized value for reconstruction.

SPHQuant stores $\tilde{r}_i$ with group-wise uniform quantization. Radius groups are formed along the input dimension of each weight row: each group contains 128 consecutive scalar weights, or equivalently 16 consecutive 8D weight vectors. Let $\mathcal{G}$ denote one such group, and let $i \in \mathcal{G}$ index an 8D weight vector in the group. Suppose each radius is allocated V bits. We initialize $\tilde{r}^{\mathcal{G}}_{\min} = \min_{i \in \mathcal{G}} \tilde{r}_i$, $\tilde{r}^{\mathcal{G}}_{\max} = \max_{i \in \mathcal{G}} \tilde{r}_i$, and $\Delta_{\mathcal{G}} = (\tilde{r}^{\mathcal{G}}_{\max} - \tilde{r}^{\mathcal{G}}_{\min})/(2^V - 1)$. We quantize and dequantize each radius as

$$q_i = \text{clip}\left(\left\lfloor \frac{\tilde{r}_i - \tilde{r}^{\mathcal{G}}_{\min}}{\Delta_{\mathcal{G}}} \right\rceil, 0, 2^V - 1 \right), \quad \hat{r}_i = \tilde{r}^{\mathcal{G}}_{\min} + q_i \Delta_{\mathcal{G}}, \quad i \in \mathcal{G}. \tag{7}$$

The i-th vector is reconstructed as $\hat{\mathbf{w}}_i = \hat{r}_i \cdot \mathbf{s}_i \odot \mathbf{c}_{z_i}$.

3.2 Outlier-aware radial bit allocation

As discussed in Sec. 1, spherical coordinates isolate the effect of weight outliers into the radius. Compared with quantizing weights in Cartesian coordinates, spherical quantization concentrates outliers in the radius only, so we can mitigate the outlier problem by allocating extra bits to the radius. To verify that the accuracy gain from increasing the radius bit-width comes from mitigating outliers rather than simply using more bits, we introduce a matched-bit control experiment on layer 0 of Qwen3-VL-8B-Instruct.

For SPHQuant, we quantize the layer with the 2-bit direction setting in Sec. 3.1.
We vary the radius bit-width $V \in \{2, 3, 4\}$, and measure the output reconstruction loss against the full-precision layer after the fine-tuning procedure introduced in Sec. 3.3. We report the relative loss reduction using each method's own 2-bit result as the zero point.
As a Cartesian baseline, we use RTN quantization and randomly assign higher precision to $1/8$ of the weights.

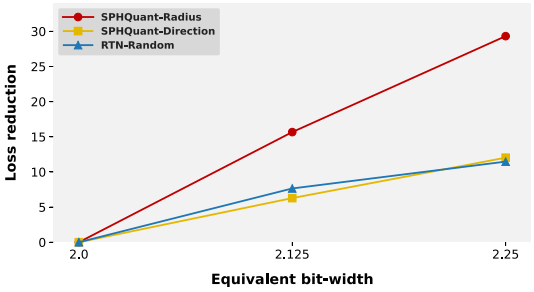


Figure 4: Loss reduction under matched average bit-widths. Higher is better.

The three matched settings are: all weights use 2-bit RTN for 2.00 equivalent bits; randomly selected $1/8$ weights use 3 bits and the rest use 2 bits for 2.125 equivalent bits; randomly selected $1/8$ weights use 4 bits and the rest use 2 bits for 2.25 equivalent bits. These match SPHQuant's average bit-width $2 + (V - 2)/8$ for $V = 2, 3, 4$, respectively.

From Fig. 4, as the equivalent bit-width increases from 2.00 to 2.25, SPHQuant improves much faster than the random mixed-precision RTN baseline. This indicates that the gain is not merely due to using more bits, but comes from allocating the extra precision to the radial component where outlier magnitude is concentrated. We further discuss how allocating extra bits to the radius or direction affects final accuracy in Sec. 4.3.

3.3 Efficient layer-wise optimization with angular codebooks

After the spherical quantization step, the trainable parameters are the continuous radius parameters $\boldsymbol{\rho}$ and the direction codebook parameters $\boldsymbol{\phi}$. The codebook is initialized by radius-weighted spherical k-means, while the pre-quantization radii are initialized by projection onto the assigned directions.

Inspired by the layer-wise reconstruction strategy in ParoQuant [14], SPHQuant performs layer-wise reconstruction with two activation streams for each decoder layer. The full-precision stream provides the teacher output, and the quantized stream provides the student input produced by the already quantized preceding layers. The optimization objective is

$$\min_{\boldsymbol{\rho},\boldsymbol{\phi}} \mathbb{E}_{\mathbf{x}_q} \left[\left\| F_l^{\mathrm{SPH}}(\mathbf{x}_q; \boldsymbol{\rho}, \boldsymbol{\phi}) - F_l^{\mathrm{FP}}(\mathbf{x}_{\mathrm{fp}}) \right\|_2^2 \right], \tag{8}$$

where F_l^{SPH} denotes the l-th decoder layer with selected linear modules replaced by SPHQuant pseudo-quantized modules, and F_l^{FP} denotes the original full-precision layer. Here $\mathbf{x}_{\mathrm{fp}}$ is the input activation to the full-precision layer, while $\mathbf{x}_q$ is the activation produced by the already quantized preceding layers. Optimizing the quantized layer on $\mathbf{x}_q$ allows later layers to adapt to errors introduced by earlier quantized layers, reducing error propagation in the full model. For the radius, we map each continuous parameter to a positive value by $r = \mathrm{softplus}(\rho)$ and apply V-bit group-wise fake quantization with a straight-through estimator during training.

Angular codebook tuning. The key design is to fine-tune directions through angles rather than directly updating 8D unit vectors. A Euclidean gradient on a unit codebook vector can contain components normal to the sphere. To keep the vector on the unit sphere, direct vector tuning must project the gradient onto the sphere or renormalize the vector after each update, which removes part of the update and reduces the effective gradient. SPHQuant avoids this inefficiency by parameterizing each codebook vector with seven hyperspherical angles:

$$[\mathbf{c}]_1 = \cos\theta_1, \quad [\mathbf{c}]_i = \left(\prod_{j=1}^{i-1} \sin\theta_j \right) \cos\theta_i \ (2 \le i \le 7), \quad [\mathbf{c}]_8 = \prod_{j=1}^{7} \sin\theta_j. \tag{9}$$

To keep the optimization unconstrained while ensuring nonnegative unit directions, we store angle logits $\boldsymbol{\phi} \in \mathbb{R}^7$ and map them to

$$\theta_j = \epsilon + \left(\frac{\pi}{2} - 2\epsilon \right) \sigma(\phi_j). \tag{10}$$

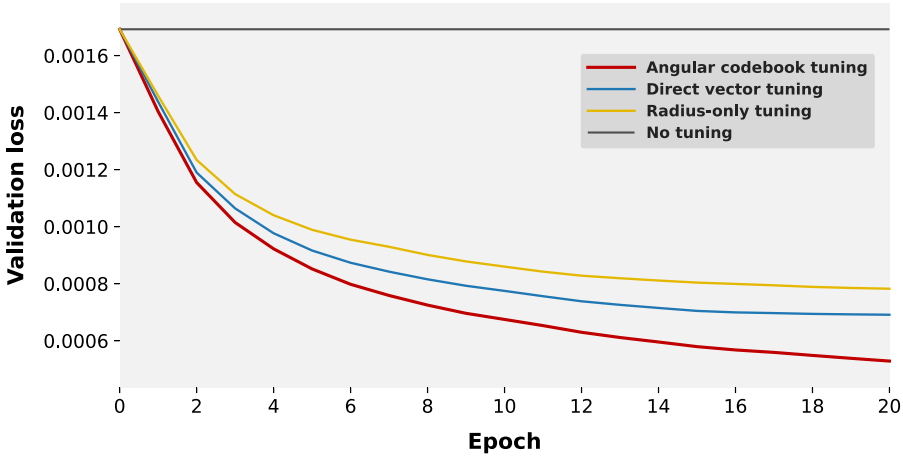


Figure 5: Loss curve of layer 0 of Qwen3-VL-8B-Instruct under different tuning strategies.

Thus every forward pass reconstructs a valid codebook $\mathcal{C}(\phi) \subset \mathbb{S}_+^7$ by construction. The codebook indices z_i are fixed after initialization; fine-tuning only moves the angular codebook and radii. As illustrated in Fig. 5, angular codebook tuning reduces reconstruction loss more effectively in this experiment than radius-only tuning or direct vector tuning.

3.4 Efficient kernel implementation

We implement efficient dequantization in a fused CUDA GEMV kernel [19]. The following packed layout and 64-entry codebook apply to W2 with $B = 2$ and $V = 4$.

Packed extra radius bits. Along the input dimension, every 128 weights form one quantization group, corresponding to 16 8D weight vectors. For each 8D vector, SPHQuant stores its packed quantized code in one 16-bit word: the lower 8 bits encode signs, the next 6 bits encode the direction codebook index, and the upper 2 bits encode the low bits of the radius code. Since the radius receives extra precision, each 128-weight group additionally stores one 32-bit word that packs the extra 2 radius bits of all 16 vectors. This word-aligned layout enables coalesced global-memory access.

Shared-memory direction lookup. We preload the 64-entry direction codebook into shared memory and store each 8D direction as four `half2` pairs. During GEMV, each lane loads the direction by codebook index, restores coordinate signs from the packed sign bits, and multiplies the signed direction by the dequantized radius.

Fused dequantization and accumulation. For each 8D vector, SPHQuant reconstructs the weight block only in registers. The reconstructed block is immediately multiplied with the fp16 activation values and accumulated in fp32. By fusing radius dequantization, direction lookup, sign restoration, and GEMV accumulation in one kernel, SPHQuant avoids writing dequantized weights to global memory while preserving the accuracy benefit of extra radial precision.

Sec. A summarizes the full SPHQuant layer-wise quantization procedure, and Sec. D provides additional CUDA implementation details.

4 Experiments

4.1 Experimental settings

Models and tasks. We evaluate SPHQuant on recent VLMs covering different model scales and architectures: Qwen3-VL-8B-Instruct [1], Qwen3.5-9B [21], and Gemma4-26B-A4B-it [10]. For standard multimodal evaluation, we use benchmarks covering college-level multimodal understanding: MMMU_val [31]; scene text understanding: TextVQA [23]; science question answering: ScienceQA [17]; real-world visual understanding: RealWorldQA [30]; and general multimodal per-

Table 1: Comparison on standard VLM benchmarks. Bit denotes nominal quantization settings. SPHQuant W2 uses 2.25 encoded bits per weight; its Qwen3-VL-8B-Instruct W3 row uses 3.125. These payloads exclude group parameters and codebooks. See Tab. 12 for matched-bit comparisons.

Model	Bit	Method	Tasks					Avg.
			MMMU_val	TextVQA	ScienceQA	RealWorldQA	MMSTAR	
Qwen3-VL-8B-Instruct	BF16	BF16	52.9	82.4	93.1	69.7	68.0	73.2
	W2	QTIP	42.2	**77.2**	80.4	57.1	**57.6**	62.9
		SPHQuant (Ours)	**42.2**	76.8	**81.0**	**57.4**	57.5	**63.0**
	W3	RTN	46.9	75.4	84.2	63.1	60.8	66.1
		AWQ	49.8	78.9	87.3	65.9	63.7	69.1
		MBQ	49.3	79.8	88.4	65.2	63.7	69.3
		ParoQuant	50.5	79.5	88.2	68.3	64.6	70.2
		QTIP	**52.1**	80.5	**92.0**	**70.7**	66.8	72.4
		SPHQuant (Ours)	51.7	**81.0**	91.8	69.9	**67.6**	**72.4**
Qwen3.5-9B	BF16	BF16	47.4	69.1	92.5	67.1	64.8	68.2
	W2	QTIP	**41.7**	63.0	**84.7**	**61.5**	61.1	**62.4**
		SPHQuant (Ours)	41.4	**63.7**	83.0	61.3	**61.7**	62.2
	W3	RTN	38.3	44.5	78.8	56.3	50.5	53.7
		AWQ	36.8	56.4	79.3	57.8	57.3	57.5
		MBQ	38.4	61.6	73.5	63.9	56.9	58.9
		ParoQuant	42.0	66.3	87.1	64.5	58.8	63.7
		QTIP	42.4	**67.2**	88.8	65.7	61.7	65.2
		SPHQuant (Ours)	**43.7**	66.3	**90.0**	**66.3**	**63.0**	**65.9**
Gemma4-26B-A4B	BF16	BF16	53.8	69.5	91.3	65.8	64.2	68.9
	W3	RTN	46.7	62.8	83.2	59.1	59.1	62.2
		AWQ	48.3	66.1	84.7	58.7	62.3	64.0
		MBQ	49.0	64.2	87.2	60.5	**63.5**	64.9
		SPHQuant (Ours)	**51.4**	**67.0**	**88.7**	**64.6**	63.3	**67.0**

ception: MMStar [4]. For reasoning-oriented evaluation, we enable the thinking mode of Qwen3.5-9B on MMMU-Pro [32] to assess long-form multimodal reasoning accuracy.

Baselines. We compare SPHQuant with two categories of weight-only PTQ baselines. Linear quantization methods: AWQ [15], MBQ [13], and ParoQuant [14]. AWQ uses activation-aware scaling to protect salient weights, MBQ improves VLM quantization with modality-balanced calibration, and ParoQuant applies pairwise rotations to reduce rotation overhead. Vector quantization methods: QTIP [25], which uses trellis-coded quantization with incoherence processing to improve extreme low-bit accuracy. Since the novel architecture of Gemma4-26B-A4B depends on a specialized training procedure, we do not include ParoQuant or QTIP results on this model.

Implementation details. We use 2048 COCO samples [16] for calibration and reserve 128 samples for validation. Quantization is performed with a group size of 128. The direction codebook is initialized by spherical k-means with 65,536 sampled vectors and 20 iterations. We optimize the spherical quantization parameters with layer-wise reconstruction using MSE loss, where the student layer takes activations from the quantized stream. Joint optimization is run for 20 epochs with Adam [12], using learning rates of 1×10^{-3} for both radius parameters and codebook parameters. We select the best checkpoint according to validation loss. Layer-wise calibration is conducted on a single RTX PRO 6000 GPU with 96GB memory. For W2 evaluations, we use $V = 4$ radius bits, yielding 2.25 encoded bits per weight; the Qwen3-VL-8B-Instruct W3 results in Tab. 1 use 3.125. Both payloads exclude group-wise parameters and direction codebooks; Sec. B details the storage accounting. For Qwen3-VL-8B-Instruct, calibration also preserves DeepStack visual features to match the real multimodal inference path; details are provided in Sec. C.

Table 2: Accuracy of Qwen3.5-9B under long-reasoning generation.

Bit	Method	MMMU-Pro
BF16	BF16	63.6
W2	QTIP	52.7
	SPHQuant (Ours)	**53.1**
W3	RTN	43.1
	AWQ	54.8
	MBQ	57.1
	ParoQuant	60.2
	QTIP	61.2
	SPHQuant (Ours)	**61.9**

Table 3: Decoding throughput (tokens/s) of Qwen3-VL-8B-Instruct on RTX A6000. SPHQuant W2 uses 2.25 encoded bits per weight.

Bit	Method	Throughput	Speedup
BF16	BF16	43.45	1.00
W2	AWQ	140.57	3.24
	ParoQuant	118.47	2.73
	QTIP	102.05	2.35
	SPHQuant (Ours)	132.96	3.06

Table 4: Ablation on bit allocation.

Variant	Sign	Dir.	Radius	Total	Eq. Bits	MMMU
W2 SPHQuant	8	6	2	16	2.000	37.2
+1 Radius Bit	8	6	3	17	2.125	40.9
+1 Direction Bit	8	7	2	17	2.125	38.8
+2 Radius Bits	8	6	4	18	2.250	**42.2**
+2 Direction Bits	8	8	2	18	2.250	39.5

Table 5: Calibration and epoch ablation.

Calib. Samples	Epochs	MMMU
512	20	40.9
1024	20	41.7
2048	5	41.2
2048	10	42.0
2048	20	**42.2**

4.2 Comparison with state-of-the-art methods

Task accuracy. We evaluate task accuracy on Qwen3-VL-8B-Instruct, Qwen3.5-9B, and Gemma 4-26B-A4B-it, on standard multimodal benchmarks with thinking disabled. We also evaluate the quantized Qwen3.5-9B model on MMMU-Pro with `enable_thinking=True`. At 3 bits, we compare SPHQuant with all baselines. At 2 bits, since linear quantization methods largely collapse, we only compare SPHQuant with QTIP. As shown in Tab. 1, SPHQuant outperforms linear quantization baselines and achieves accuracy comparable to the state-of-the-art extreme low-bit method QTIP.

Decoding throughput. We test decoding throughput of Qwen3-VL-8B-Instruct under the W2A16 setting on an RTX A6000 GPU. We compare AWQ, ParoQuant, QTIP, and SPHQuant at 2-bit weight quantization. For a fair comparison, all methods are evaluated with the same Transformers inference pipeline [28], and we only replace the quantized linear modules with the corresponding method-specific CUDA kernels. The results in Tab. 3 show that SPHQuant improves decoding throughput by 30.3% over QTIP and 12.2% over ParoQuant, while being only 5.4% slower than AWQ. Sec. B discusses encoded bit-width and optimization cost.

Sec. E reports additional baseline, matched-budget, and deployment evaluations.

4.3 Ablation study

Unless otherwise specified, all ablation experiments are conducted on Qwen3-VL-8B-Instruct under the W2A16 setting. We use 8D weight vectors, a group size of 128, 2048 COCO calibration samples, and 20 epochs of layer-wise fine-tuning.

Ablation on outlier-aware radial bit allocation. Starting from the 2-bit SPHQuant setting, we compare two matched-budget allocation strategies: adding bits to the radial component, or using the same additional budget to enlarge the direction codebook. As shown in Tab. 4, allocating extra bits to the radius yields larger accuracy gains than assigning the same budget to direction indices. This supports our observation that outlier magnitude is mainly concentrated in the radius.

Calibration cost of SPHQuant. Tab. 5 reports sensitivity to calibration set size and fine-tuning epochs. Increasing the calibration set from 512 to 2048 samples at 20 epochs improves MMMU accuracy from 40.9 to 42.2, while 10 epochs with 2048 samples reach 42.0.

Table 6: Effect of angular codebook tuning.

Variant	Radius Tune	Direction Tune	Parameterization	Valid Loss	MMMU	TextVQA
No Tuning	No	No	Fixed codebook	0.00169	23.4	62.9
Radius-only Tuning	Yes	No	Fixed codebook	0.00078	39.7	73.8
Radius + Vector Tuning	Yes	Yes	Vector + renorm	0.00069	40.5	74.2
Radius + Angular Tuning	Yes	Yes	Hyperspherical angles	**0.00052**	**42.2**	**76.8**

Effect of angular codebook tuning. We compare the initialization-only quantizer, radius-only tuning, and the full SPHQuant optimization that jointly tunes radius parameters and direction codebook angles. We also compare with direct vector tuning followed by renormalization. Tab. 6 shows that angular tuning improves over radius-only tuning, indicating that adapting the direction codebook on the unit sphere is important for improving tuning efficiency.

Choice of vector dimension. We compare the evaluated 4D, 8D, and 16D configurations on Qwen3-VL-8B-Instruct in Tab. 7. Moving from 4D to 8D improves MMMU and TextVQA accuracy by 0.8 and 1.1 percentage points, respectively, while the 16D configuration adds only 0.2 points on each metric over 8D. The direction-codebook storage grows substantially: under the W2 direction allocation, the FP16 codebooks occupy 32 B, 1 KiB, and 512 KiB, respectively. The 16D codebook exceeds the shared-memory capacity of an RTX 4090 SM [18], while the 8D codebook is compact enough for our shared-memory lookup. These observations motivate 8D as a practical default for the current implementation. This does not establish optimality at a fixed storage budget.

Table 7: Accuracy and direction-codebook storage across vector dimensions on Qwen3-VL-8B-Instruct. Codebooks use FP16 directions under the W2 direction allocation.

Vector size	Direction codebook	MMMU_val ↑	TextVQA ↑
4D	32 B	41.4	75.7
8D	1 KiB	42.2	76.8
16D	512 KiB	**42.4**	**77.0**

5 Conclusion

In this paper, we propose SPHQuant, a spherical weight-only quantization framework for VLMs. By representing weights in spherical coordinates, SPHQuant decouples outlier magnitudes from bounded directional information and enables more effective bit allocation under extreme low-bit constraints. We further develop efficient optimization and hardware-friendly inference support to preserve accuracy while reducing runtime overhead. Experiments on recent VLMs show that SPHQuant achieves competitive extreme low-bit accuracy and improves decoding throughput by 30.3% over QTIP on RTX A6000. SPHQuant combines outlier handling, codebook design, and kernel efficiency through a spherical representation for practical extreme low-bit VLM deployment.

Acknowledgments and Disclosure of Funding

This work was supported by the CAAI-Tencent Rhino-Bird Open Research Fund.

Competing interests. The authors declare no additional competing interests beyond the affiliations and funding disclosed above.

References

[1] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL Technical Report, 2025. URL https://arxiv.org/abs/2511.21631.

[2] Bowen Chai, Zheng Chen, Libo Zhu, Wenbo Li, Yong Guo, and Yulun Zhang. QuantVSR: Low-Bit Post-Training Quantization for Real-World Video Super-Resolution. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(4):2689–2697, 2026. doi: 10.1609/aaai.v40i4.37257. URL https://ojs.aaai.org/index.php/AAAI/article/view/37257.

[3] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-Bit Quantization of Large Language Models With Guarantees, 2024. URL https://arxiv.org/abs/2307.13304.

[4] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and Feng Zhao. Are We on the Right Way for Evaluating Large Vision-Language Models?, 2024. URL https://arxiv.org/abs/2403.20330.

[5] Zheng Chen, Kewei Zhang, Xiaoyang Liu, Weihang Zhang, Mengfan Wang, Yifan Fu, and Yulun Zhang. QuantDemoire: Quantization with Outlier Aware for Image Demoiréing. *arXiv preprint arXiv:2510.04066*, 2025. URL https://arxiv.org/abs/2510.04066.

[6] Xiangxiang Chu, Limeng Qiao, Xinyang Lin, Shuang Xu, Yang Yang, Yiming Hu, Fei Wei, Xinyu Zhang, Bo Zhang, Xiaolin Wei, and Chunhua Shen. MobileVLM: A Fast, Strong and Open Vision Language Assistant for Mobile Devices, 2023. URL https://arxiv.org/abs/2312.16886.

[7] Xiangxiang Chu, Limeng Qiao, Xinyu Zhang, Shuang Xu, Fei Wei, Yang Yang, Xiaofei Sun, Yiming Hu, Xinyang Lin, Bo Zhang, and Chunhua Shen. MobileVLM V2: Faster and Stronger Baseline for Vision Language Model, 2024. URL https://arxiv.org/abs/2402.03766.

[8] Clement Farabet and Olivier Lacombe. Gemma 4: Byte for byte, the most capable open models, April 2026. URL https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate Post-training Compression for Generative Pretrained Transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[10] Google DeepMind. Gemma 4 Model Card, 2026. URL https://ai.google.dev/gemma/docs/core/model_card_4. Last updated 2026-04-17.

[11] Yizhang Jin, Jian Li, Yexin Liu, Tianjun Gu, Kai Wu, Zhengkai Jiang, Muyang He, Bo Zhao, Xin Tan, Zhenye Gan, Yabiao Wang, Chengjie Wang, and Lizhuang Ma. Efficient Multimodal Large Language Models: A Survey. *Visual Intelligence*, 3(27), 2025. doi: 10.1007/s44267-025-00099-6. URL https://arxiv.org/abs/2405.10739.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[13] Shiyao Li, Yingchun Hu, Xuefei Ning, Xihui Liu, Ke Hong, Xiaotao Jia, Xiuhong Li, Yaqi Yan, Pei Ran, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. MBQ: Modality-Balanced Quantization for Large Vision-Language Models, 2024. URL https://arxiv.org/abs/2412.19509.

[14] Yesheng Liang, Haisheng Chen, Zihan Zhang, Song Han, and Zhijian Liu. ParoQuant: Pairwise Rotation Quantization for Efficient Reasoning LLM Inference. In *International Conference on Learning Representations (ICLR)*, 2026.

[15] 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. In *MLSys*, 2024.

[16] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context, 2015. URL https://arxiv.org/abs/1405.0312.

[17] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering, 2022. URL https://arxiv.org/abs/2209.09513.

[18] NVIDIA Corporation. NVIDIA Ada GPU Architecture Tuning Guide, 2026. URL https://docs.nvidia.com/cuda/ada-tuning-guide/index.html.

[19] NVIDIA Corporation. CUDA C++ Programming Guide, 2026. URL https://docs.nvidia.com/cuda/cuda-c-programming-guide/.

[20] Guangshuo Qin, Zhiteng Li, Zheng Chen, Weihang Zhang, Linghe Kong, and Yulun Zhang. VEQ: Modality-Adaptive Quantization for MoE Vision-Language Models. *arXiv preprint arxiv:2602.01037*, 2026.

[21] Qwen Team. Qwen3.5: Towards Native Multimodal Agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[22] Qwen Team. Qwen3.6-35B-A3B: Agentic Coding Power, Now Open to All, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

[23] Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards VQA Models That Can Read, 2019. URL https://arxiv.org/abs/1904.08920.

[24] Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even Better LLM Quantization with Hadamard Incoherence and Lattice Codebooks, 2024. URL https://arxiv.org/abs/2402.04396.

[25] Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. QTIP: Quantization with Trellises and Incoherence Processing. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=7sdkLVuYCU.

[26] Tycho F. A. van der Ouderaa, Maximilian L. Croci, Agrin Hilmkil, and James Hensman. Pyramid Vector Quantization for LLMs, 2024. URL https://arxiv.org/abs/2410.16926.

[27] Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. InternVL3.5: Advancing Open-Source Multimodal Models in Versatility, Reasoning, and Efficiency. *arXiv preprint arXiv:2508.18265*, 2025. URL https://arxiv.org/abs/2508.18265.

[28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6/.

[29] Tianxing Wu, Zheng Chen, Cirou Xu, Bowen Chai, Yong Guo, Yutong Liu, Linghe Kong, and Yulun Zhang. LSGQuant: Layer-Sensitivity Guided Quantization for One-Step Diffusion Real-World Video Super-Resolution. *arXiv preprint arXiv:2602.03182*, 2026. URL https://arxiv.org/abs/2602.03182.

[30] xAI Team. RealWorldQA: A Real-World Multimodal Question Answering Benchmark, 2024. URL https://huggingface.co/datasets/xai-org/RealworldQA.

[31] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI, 2024. URL https://arxiv.org/abs/2311.16502.

[32] Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. MMMU-Pro: A More Robust Multi-discipline Multimodal Understanding Benchmark, 2025. URL https://arxiv.org/abs/2409.02813.

[33] Yuxuan Yue, Zukang Xu, Zhihang Yuan, Dawei Yang, Jianlong Wu, and Liqiang Nie. PCDVQ: Enhancing Vector Quantization for Large Language Models via Polar Coordinate Decoupling, 2025. URL https://arxiv.org/abs/2506.05432.

Appendix

A SPHQuant quantization procedure

The following pseudocode summarizes the complete layer-wise quantization pipeline used by SPHQuant. The procedure follows the method described in Sec. 3.1–3.3: each selected linear layer is quantized in spherical coordinates, the radius and angular codebook parameters are optimized by layer-wise reconstruction, and the final representation is packed for efficient inference.

Pseudocode: SPHQuant layer-wise quantization

Input: full-precision VLM M, calibration set $\mathcal{D}$, selected linear layers, vector dimension $k = 8$, target bit-width B, radius bit-width V, group size $G = 128$, fine-tuning epochs T.

Output: quantized VLM: packed signs, direction indices, radius codes, group scales, and tuned codebooks.

Run the full-precision model on calibration samples and collect the input activations of the first language layer. Split calibration samples into training and validation subsets.
Initialize two activation streams: the full-precision stream $\mathbf{x}_{\mathrm{fp}}$ and the quantized stream $\mathbf{x}_q$.
For each decoder layer l **do**: compute the full-precision teacher output $F_l^{\mathrm{FP}}(\mathbf{x}_{\mathrm{fp}})$.
  For each selected weight matrix W in layer l **do**: split W row-wise into contiguous 8D vectors $\mathbf{w}_i$.
    Decompose each vector into coordinate signs, radius, and positive unit direction: $\mathbf{s}_i = \mathrm{sign}(\mathbf{w}_i)$, $(\mathbf{w}_i)_+ = |\mathbf{w}_i|$, $r_i = \|(\mathbf{w}_i)_+\|_2$, and $\mathbf{d}_i = (\mathbf{w}_i)_+ / \max(r_i, \epsilon)$.
    Initialize the positive-direction codebook by radius-weighted spherical k-means; assign z_i by cosine similarity and fit pre-quantization radii by least-squares projection onto assigned directions.
    Parameterize radii by continuous variables $\boldsymbol{\rho}$ with softplus mapping, and parameterize each codebook entry by angular variables $\boldsymbol{\phi}$ so that all directions remain on $\mathbb{S}_+^7$.
  end for
  Replace selected linear modules in layer l with SPHQuant pseudo-quantized modules. During each forward pass, apply group-wise V-bit radius fake quantization with a straight-through estimator and reconstruct weights as $\hat{\mathbf{w}}_i = \hat{r}_i \mathbf{s}_i \odot \mathbf{c}_{z_i}$.
  Optimize $\boldsymbol{\rho}$ and $\boldsymbol{\phi}$ for T epochs using layer-wise reconstruction loss between $F_l^{\mathrm{SPH}}(\mathbf{x}_q)$ and $F_l^{\mathrm{FP}}(\mathbf{x}_{\mathrm{fp}})$, and select the checkpoint with the lowest validation loss.
  Finalize discrete radius codes, radius group minima and scales, sign bits, direction indices, and optimized direction codebook entries. For the W2 setting with $V = 4$, pack each 8D vector into the 16-bit base code and pack the extra radius bits of every 16 vectors into one auxiliary 32-bit word.
  Update activation streams: set $\mathbf{x}_q \leftarrow F_l^{\mathrm{SPH}}(\mathbf{x}_q)$ and $\mathbf{x}_{\mathrm{fp}} \leftarrow F_l^{\mathrm{FP}}(\mathbf{x}_{\mathrm{fp}})$, then proceed to the next layer.
end for
Return the quantized model and packed CUDA-friendly metadata for inference.

B Additional experiment settings

Effective bit-width and throughput fairness. SPHQuant assigns two extra radial bits to each 8D vector in the W2 setting, resulting in an encoded weight payload of 2.25 bits per weight, excluding group-wise quantization parameters and codebook storage. This extra budget is used only for the radius, as motivated by Sec. 3.2, where outlier magnitude is concentrated. Although this gives SPHQuant 0.25 more bits than strict 2-bit methods, the comparison should be interpreted with this storage difference in mind. Additional bits increase memory traffic, while Tab. 3 shows that SPHQuant is still faster than QTIP and ParoQuant. This is enabled by the compact positive-direction codebook and packed radial-bit layout described in Sec. 3.4.

Additional decoding evaluation. The original throughput experiment uses RTX A6000. The additional evaluation uses RTX 4090 with 24 GB of memory, Qwen3-VL-8B-Instruct, batch size 1, 256 prefill tokens, and 512 generated tokens. Tab. 10 reports the additional throughput results, and Tab. 11 reports measured GPU-memory consumption. These quantities are presented separately from encoded bits per weight because they capture different aspects of deployment cost.

Matched encoded weight budgets. The additional comparison in Tab. 12 evaluates RTN, ParoQuant, and SPHQuant at 3.00 and 3.125 encoded bits per weight on Qwen3-VL-8B-Instruct. For

Table 8: Additional training settings on Qwen3-VL-8B-Instruct.

Method	Training Samples	Epochs	Time	Device
QTIP	512	10	8h	1× RTX PRO 6000
ParoQuant	1024	10	6h	1× RTX PRO 6000
SPHQuant	2048	20	6h	1× RTX PRO 6000

SPHQuant with 8D vectors, the encoded payload is $(7B + V)/8$ bits per weight; it excludes group-wise quantization parameters and direction codebooks. The SPHQuant W3 row for Qwen3-VL-8B-Instruct in Tab. 1 corresponds to the 3.125-bit result. Keeping these payloads explicit avoids interpreting nominal W2/W3 labels as complete model-storage measurements.

Additional training settings. For fair comparison, we keep the optimization budget of QTIP, ParoQuant, and SPHQuant in a similar single-GPU range. Since the three methods use different optimization procedures, e.g., QTIP optimizes linear layers individually, ParoQuant [14] uses a two-stage optimization pipeline, and SPHQuant performs layer-wise spherical reconstruction, we match their practical quantization cost rather than forcing identical hyperparameters. Tab. 8 reports the training samples, epochs, wall-clock time, and device used for Qwen3-VL-8B-Instruct.

C DeepStack-aware calibration for Qwen3-VL

Qwen3-VL-8B-Instruct [1] uses DeepStack features to inject visual information into selected language decoder layers. During calibration, SPHQuant explicitly preserves this multimodal path. The calibration inputs are first preprocessed by `process_model`, and the decoder layers that should receive DeepStack features are recorded through `_mbq_calib_deepstack_layers`. This carries visual DeepStack features with each calibration batch to the corresponding decoder layers.

When SPHQuant captures the first-layer hidden states and the layer-wise keyword arguments, it also stores these DeepStack-related intermediate features as part of the layer input context. Therefore, during layer-wise reconstruction, each quantized layer receives the same type of multimodal context as in the original forward pass. This avoids text-only hidden-state approximation and aligns calibration with the actual vision-language inference path.

D Additional CUDA implementation details

The following details describe the fused GEMV implementation for the W2 configuration with 8D vectors, a 64-entry positive-direction codebook, and four radius bits. They supplement the packed-weight layout in Sec. 3.4.

Thread mapping and activation access. Each thread block contains 256 threads and computes eight output channels. Each thread processes 32 FP16 activations, corresponding to four 8D weight vectors. This mapping exposes parallelism across output channels and input-vector groups while allowing the packed codes and activations to be consumed within the same kernel.

Shared-memory codebook. The 64×8 FP16 direction codebook occupies 1 KiB and is staged in shared memory. A transposed layout arranges codebook coordinates for lookup by the decoding threads and reduces access conflicts. Direction indices select the positive unit vectors, and the separately stored sign bits restore coordinate signs during reconstruction.

Packed decoding and fused computation. Bit operations extract the coordinate signs, six-bit direction index, and low radius bits from each 16-bit base word. The additional 32-bit word for a group supplies the extra radius bits of its 16 vectors. After assembling the radius code, a fused multiply-add applies the group's scale and offset to reconstruct the radius. Direction lookup, sign restoration, radius reconstruction, and multiplication with the activations are fused, avoiding a global-memory buffer of dequantized weights.

Vectorized products and reduction. The kernel uses vectorized `half2` instructions for pairwise operations on FP16 values and accumulates partial results in FP32. Warp-shuffle operations and shared-memory reductions combine thread-local contributions into the output channels. The CUDA kernel will be released with the implementation.

E Additional comparisons and deployment experiments

E.1 Comparison with direction–magnitude quantization

PVQ [26] and PCDVQ [33] provide closely related approaches that separately represent weight direction and magnitude. Tab. 9 compares their evaluated implementations with SPHQuant on Qwen3-VL-8B-Instruct. SPHQuant improves MMMU accuracy by 2.0 and 2.4 percentage points over PVQ and PCDVQ, respectively, and TextVQA by 3.1 and 2.6 percentage points. It also achieves higher measured decoding throughput in this comparison. Together, these results support the practical effectiveness of SPHQuant's combination of rotation-free quantization, adaptive angular codebooks, and radial bit allocation; they do not isolate the contribution of any one component.

Table 9: Additional comparison with direction–magnitude quantizers on Qwen3-VL-8B-Instruct. SPHQuant uses its W2 configuration with 2.25 encoded bits per weight.

Method	MMMU_val ↑	TextVQA ↑	Tokens/s ↑
PVQ	40.2	73.7	209.03
PCDVQ	39.8	74.2	210.11
SPHQuant	**42.2**	**76.8**	**288.63**

E.2 Decoding on RTX 4090

We further evaluate Qwen3-VL-8B-Instruct on an RTX 4090 with 24 GB of memory, using batch size 1, a prefill length of 256 tokens, and a decoding length of 512 tokens. As shown in Tab. 10, SPHQuant reaches 288.63 tokens/s, or 2.70× the FP16 throughput. It is 37.6% faster than QTIP and 4.9% faster than ParoQuant in this setting, while RTN remains slightly faster. This extends the throughput evaluation to a GPU with less memory than RTX A6000.

Table 10: Batch-one decoding throughput of Qwen3-VL-8B-Instruct on RTX 4090-24GB, with 256 prefill tokens and 512 generated tokens. Payload excludes auxiliary storage.

Method	Payload (bits/weight)	Tokens/s ↑	Speedup
FP16	16	106.85	1.00×
QTIP	2.00	209.76	1.96×
ParoQuant	2.00	275.07	2.57×
SPHQuant	2.25	288.63	2.70×
RTN	2.00	**300.059**	**2.81**×

E.3 Measured GPU-memory consumption

Tab. 11 reports measured GPU-memory consumption for the quantized Qwen3-VL-8B-Instruct model under the same inference configuration across the three methods. SPHQuant uses 5.35 GB, compared with 5.02 GB for QTIP and 5.24 GB for ParoQuant. Thus, its measured footprint is 0.33 GB and 0.11 GB larger, respectively. Encoded weight payload and GPU-memory consumption are distinct quantities: the latter also includes unquantized components and auxiliary storage. These measurements characterize the tested configuration; additional radius storage can still matter when available memory is close to a deployment limit.

E.4 Comparison at matched encoded bit-widths

To separate accuracy gains from additional weight-storage capacity, we compare SPHQuant, RTN, and ParoQuant on Qwen3-VL-8B-Instruct at 3.00 and 3.125 encoded bits per weight. Tab. 12 shows

Table 11: Measured GPU-memory consumption of quantized Qwen3-VL-8B-Instruct. Payload refers to encoded weights and does not include auxiliary storage.

Method	Payload (bits/weight)	GPU memory (GB) ↓
QTIP	2.00	5.02
ParoQuant	2.00	5.24
SPHQuant	2.25	5.35

that SPHQuant obtains the highest MMMU and TextVQA accuracy among these methods at both budgets. At 3.00 bits, the margins over ParoQuant are 0.1 and 0.3 percentage points; at 3.125 bits, they increase to 0.7 and 0.8 percentage points. These comparisons complement the nominal W2/W3 results by making the encoded weight budget explicit.

Table 12: Accuracy at matched encoded weight budgets on Qwen3-VL-8B-Instruct. The bit-widths refer to encoded weights, excluding auxiliary storage.

Bits/weight	Method	MMMU_val ↑	TextVQA ↑
3.00	RTN	46.9	75.4
3.00	ParoQuant	50.5	79.5
3.00	SPHQuant	**50.6**	**79.8**
3.125	RTN	47.8	76.3
3.125	ParoQuant	51.0	80.2
3.125	SPHQuant	**51.7**	**81.0**

F Limitations

SPHQuant is designed for extreme low-bit weight-only VLM quantization. Our throughput evaluation covers RTX A6000 and RTX 4090, but not embedded or mobile devices. The RTX 4090 evaluation uses batch size 1 and a fixed sequence-length configuration; broader batch-size and context-length studies remain necessary. Performance depends on memory hierarchy, shared-memory capacity, instruction scheduling, and kernel integration, so wider hardware validation is needed. Extra radius storage may also affect whether a model fits near a device's memory limit. Although we evaluate representative multimodal benchmarks, applications such as mobile OCR, document understanding, visual assistance, and embodied reasoning require further task-specific evaluation under realistic latency, input-resolution, and generation-length constraints.

G Broader impacts

By reducing the memory footprint and improving decoding throughput of VLMs, SPHQuant may help make multimodal models more accessible on resource-constrained devices. This can support privacy-preserving on-device inference, reduce dependence on cloud connectivity, and lower the energy and hardware cost of deploying useful visual-language applications. More accessible VLM deployment can also enable privacy-invasive monitoring or other misuse, and quantization may alter model reliability or amplify existing errors and biases. Deployment should retain the base model's safeguards, respect data and model licenses, and validate accuracy and safety in the intended application.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Please refer to our abstract and introduction.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss the limitations in the appendix.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

Justification: The paper does not include theoretical results.

Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: We have provided implementation details. We will release all the code and models.

   Guidelines:

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

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

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

Answer: [No]

Justification: We promise to release the code, dataset, and all models.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We have provided implementation details.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: See the experiment part.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: See experiment part.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Please refer to the appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: This work poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We have credited most previous works in the paper.

    Guidelines:

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

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

    Answer: [Yes]

    Justification: We will release code and models. We have provided implementation details to reproduce our results.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

Answer: [N/A]

Justification: LLMs were not used as part of the core methodology or experimental pipeline.

Guidelines:

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