ViM-VQ: Efficient Post-Training Vector Quantization for Visual Mamba

Supplementary Material

Algorithm 1 ViM-VQ

1: **Input:** Weight matrix $\mathbf{W}$, calibration data $(\mathbf{x}, \mathbf{y})$
2: **Output:** Codebook $\mathbf{C}$, assignments $\mathbf{A}$
3: $\mathbf{C}, \mathbf{A} = \text{K-Means}(\mathbf{W}, k)$
4: ▷ Phase 1: Fast Convex Combination Optimization
5: **for** each sub-vector $w_{o,i/d}$ in $\mathbf{W}$ **do**
6: $\mathcal{C}_{o,i/d} = \text{Top}_n(-\|w_{o,i/d} - c(k)\|_2^2)$
7: $\mathcal{R}_{o,i/d} = \text{softmax}(z_{o,i/d})$
8: Convex combination: $(\mathcal{C}_{o,i/d}, \mathcal{R}_{o,i/d})$
9: **end for**
10: $\widehat{\mathbf{W}} = \sum \mathcal{C} \odot \mathcal{R}$
11: ▷ Initialize codewords and ratios
12: **while** not converged **do**
13: $\mathcal{L}_{\text{init}} = \|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2$
14: $\mathcal{C} \leftarrow \mathcal{C} - O(\nabla_c \mathcal{L}_{\text{init}}, \theta_c), \quad \mathcal{R} \leftarrow \mathcal{R} - O(\nabla_r \mathcal{L}_{\text{init}}, \theta_r)$
15: **end while**
16: ▷ Phase 2: Incremental Vector Quantization
17: **for** calibration step $t = 1$ to T **do**
18: $\widehat{\mathbf{y}} = \epsilon_q(\mathbf{x})$
19: $\mathcal{L} = \mathcal{L}_t + \mathcal{L}_{\text{bkd}} + \mathcal{L}_r$
20: $\mathcal{C} \leftarrow \mathcal{C} - O(\nabla_c \mathcal{L}, \theta_c), \quad \mathcal{R} \leftarrow \mathcal{R} - O(\nabla_r \mathcal{L}, \theta_r)$
21: ▷ Confirm high-ratio assignments
22: **for** each sub-vector with $r^m_{o,i/d} > \tau$ **do**
23: $\widehat{w}_{o,i/d} \leftarrow c^m_{o,i/d}$
24: $a_{o,i/d} \leftarrow \text{index}(c^m_{o,i/d})$
25: **end for**
26: ▷ Adaptive candidate replacement
27: **for** each candidate with $r^m_{o,i/d} < \lambda$ **do**
28: $c^m_{o,i/d} \leftarrow \underset{c(k) \in \mathbf{C} \setminus \mathcal{C}_{o,i/d}}{\text{argmin}} \|\widehat{w}_{o,i/d} - c(k)\|_2^2$
29: **end for**
30: **end for**
31: **return** $\mathbf{C}$, $\mathbf{A}$

8. Algorithm Details

Algorithm 1 presents the complete pseudocode of ViM-VQ, which consists of two main phases: Fast Convex Combination Optimization (lines 4-15, 26-29) and Incremental Vector Quantization (lines 17-25).

In the first phase, the algorithm initializes the codebook $\mathbf{C}$ and assignments $\mathbf{A}$ using K-Means clustering (line 3). For each weight sub-vector, it identifies the top-n nearest candidate codewords based on Euclidean distance (line 6) and computes soft assignment ratios using the softmax function (line 7). The convex combinations of candidate codewords and their ratios are then optimized to minimize reconstruction error (lines 11-15).

The second phase performs incremental quantization over T calibration steps. In each iteration, the algorithm updates both codewords and ratios using gradient descent (line 20) based on a combined objective function that includes task loss, block-wise knowledge distillation, and regularization (line 19). High-confidence soft assignments (with ratio $r > \tau$) are confirmed as hard assignments (lines 21-25), while candidates with low ratios ($r < \lambda$) are adaptively replaced with better alternatives (lines 26-29).

Type	Method	Top-1 Accuracy (%) 3-bit	W3A8	2-bit	W2A8	1-bit
UQ	GPTQ	76.81	-	2.47	-	-
	AWQ	77.10	-	1.90	-	-
	OmniQuant	78.44	-	5.14	-	-
	MambaQuant	77.01	76.67	3.49	3.19	-
	PTQ4VM	77.17	77.02	3.62	3.54	-
VQ	DKM	80.02	79.60	79.18	78.62	74.57
	VQ4DiT	79.73	79.51	78.86	78.56	74.37
	ViM-VQ	**80.34**	**80.16**	**79.46**	**79.19**	**75.58**

Table 7. Quantization results of Vim-B on ImageNet-1K validation dataset. Codebooks are quantized to 8-bit to align with "A8".

Method	Setting	Weight Storage Size	Ratio	Time
FP	-	360MB	1.0×	-
GPTQ	per-group (g128)	Qweight + S + Z = 22.5MB + 6.0MB	12.6×	35m
AWQ				1m
OmniQ				48m
MambaQ	per-channel	Qweight + S + Z = 22.5MB + 1.3MB	15.1×	16m
PTQ4VM				92m
DKM	256 × 4	Qweight + C = 22.5MB + **0.5MB**	**15.7**×	1 day
VQ4DiT				104m
ViM-VQ				88m

Table 8. Calibration efficiency of 2-bit quantization of Vim-B. 'S', 'Z', and 'C' denote the memory usage of scales, zero-points, and codebooks, respectively. 'Ratio' and 'Time' represent compression ratio and calibration time, respectively.

9. Additional Experimental Results

To provide a thorough evaluation, we benchmark ViM-VQ against a wide range of state-of-the-art quantization methods on the Vim-B model. The comparisons, detailed in Table 7, cover both weight-only quantization, such as the 3-bit setting, and mixed-precision quantization, denoted as

WbA8. Our baselines include specialized techniques for Mamba models like MambaQuant and PTQ4VM, alongside high-performance methods from the language model domain like AWQ [23] and OmniQuant [32], to ensure a comprehensive and challenging comparison.

The results in Table 7 demonstrate the superiority of vector quantization (VQ) at extremely low bit-widths. While uniform quantization (UQ) methods perform reasonably at 3-bit, their performance degrades substantially at 2-bit, with accuracy dropping to as low as 1.90% for AWQ and 2.47% for GPTQ. In contrast, VQ methods remain effective. ViM-VQ, in particular, establishes a new state-of-the-art by achieving a Top-1 accuracy of 80.34% at 3-bit and 79.46% at 2-bit. At an extreme 1-bit precision, ViM-VQ maintains an accuracy of 75.58%, outperforming its closest VQ competitor. The leading performance of ViM-VQ extends to mixed-precision settings, for instance, achieving 79.19% in the W2A8 setting.

Table 8 analyzes the practical efficiency of various methods, focusing on storage compression and calibration time. Vector quantization methods demonstrate superior storage efficiency due to their compact codebook representation. While UQ methods like GPTQ and MambaQuant add 6.0 MB and 1.3 MB of overhead for scales and zero-points, respectively, all VQ methods add only 0.5 MB for their codebooks. As a result, ViM-VQ achieves the highest compression ratio of 15.7$\times$, reducing the model size from 360 MB to just 23 MB, making it suitable for deployment on resource-constrained devices.

Regarding time efficiency, ViM-VQ achieves an effective balance between performance and cost. It completes its calibration in 88 minutes, a duration considerably shorter than that of other high-performing VQ methods. For comparison, DKM requires an entire day for calibration, while VQ4DiT takes 104 minutes. The efficient calibration of the model is attributed to our fast convex combination optimization algorithm, which efficiently manages a small set of candidate codewords.