Supplementary Material for PTQ4VM: Post-Training Quantization for Visual Mamba

A. Implementation Details

A.1. Quantization Configurations

In this section, we describe the detailed quantization configuration for reproducing PTQ4VM. In the Image Classification task, we trained for 10 epochs for W8A8, 50 epochs for W6A6, and 100 epochs for W4A4. The learning rates used in our experiments are reported in Table 1S. We denoted the learning rate for the smoothing scale s as lr_s and the learning rate for the quantization parameters (step size Δ_X and Δ_W) as lr_q. Additionally, all models used in the experiments used pretrained checkpoints provided by the official repository of Vision Mamba [6], VMamba [4], and LocalMamba [2].

	Vim-Ti	Vim-S	Vim-B	VMamba-T/S/B
lr_s	1e-2	1e-3	1e-2	1e-4
lr_q	5e-4	5e-4	5e-4	1e-4
	LocalVim-T†	LocalVim-T	LocalVim-S	LocalVMamba-S
lr_s	1e-2	1e-2	1e-4	1e-4
lr_q	5e-4	5e-4	5e-4	1e-4

Table 1S. Learning rates for Image Classification task.

For Object Detection and Instance Segmentation tasks, we used different learning rates and epochs depending on the quantization bit-width. For W8A8, we used a learning rate 1e-5 for both smoothing scale and step sizes, training for 10 epochs. For W6A6 and W4A4, we used a learning rate 1e-4 and trained for 50 epochs. As mentioned in our main manuscript, we used the VMamba-T [4] backbone trained with the MASK R-CNN 3X [1] MS setting, provided by the official VMamba repository. We applied quantization only to the backbone for our experiments. Additionally, we cropped the input images to 1280×800 to use PTS in our experiments.

B. Acceleration Kernel

In this section, we provide a more detailed explanation of the hardware implementation and experiment of PTQ4VM.

B.1. Implementation

The hardware kernel of PTQ4VM is based on CUDA programming and CUTLASS 3.5 [3], and consists of four main parts: Activation smoothing, Quantization, INT GEMM, and Dequantization.

In the Activation smoothing part, we upload activation smoothing vector to the shared memory of each block in the Tensor core. We then apply element-wise multiplication to each row of the activation tensor to perform smoothing. We can utilize more efficient operation because we apply multiplication to shared memory uploaded data.

During the Quantization part, we quantize FP16 activations to INT4/INT8. Weights are not quantized during this phase because they are already saved and loaded as INT4/INT8 values. It is noticeable that the smallest data type of integer in PyTorch [5] is INT8. Therefore, when performing INT4 quantization, we concatenate two adjacent quantized values and pack them into INT8 data. The pseudo-code for this process is as follows: `packedData = (data[1] << 4) | (data[0] & 15)`

For the INT GEMM part, we utilize CUTLASS Tensor core INT4/INT8 GEMM, as CUTLASS is known to be the most efficient open-source linear algebra library currently available. In the Dequantize part, we generate the output step size tensor by conducting the inner product of the weight step-size vector and the activation step size vector. We employ CUBLAS GEMM for this operation because it is more efficient than CUTLASS for FP16 GEMM and is also used in PyTorch's default tensor multiplication, making implementation straightforward. The output step size tensor undergoes element-wise multiplication with the output tensor to produce the final result tensor. Similar to the activation smoothing process, this operation is performed with uploading to the shared memory of each tensor core block, thereby maximizing acceleration performance.

B.2. Experimental Settings

The latency measurement for the kernel-implemented model were conducted on a single RTX 3090 GPU with a batch size of 32. To ensure accurate measurements, we employed a warming-up phase consisting of 100 iterations

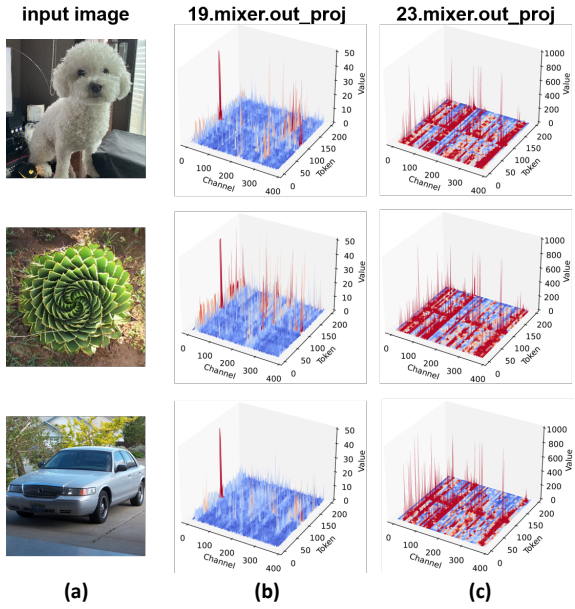

Figure 1S. The activation distribution of the out_proj layer in Vim-Ti. (a) the input image, (b) the 19th out_proj, and (c) the 23rd out_proj. Here, the layer index uses 0-based numbering.

prior to the actual timing. Subsequently, we measured the 100 times of inference latency and reported the median value. By this methodology, we can conduct more reliable assessment of the performance by mitigating the effects of initial overhead and potential outliers. The use of the median value provides a robust tendency that is less sensitive to extreme values compared to the arithmetic mean.

C. Additional Visualization

In the main manuscript, we only reported the distribution of the 22nd out_proj layer of Vim-Ti. These observations are also present at other layer indices, and additional visualizations can be found in Figure 1S. Furthermore, we have reported Observations 1 and 2 for VMamba, LocalVim†, LocalVim, and LocalVMamba in Figure 2S, and Observation 3 in Figure 3S. We visualized the input activation of the out_proj layer of backbones in Figure 2S, except for LocalVMamba-S case, which is visualization of dt_proj layer.

References

[1] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 1

[2] Tao Huang, Xiaohuan Pei, Shan You, Fei Wang, Chen Qian, and Chang Xu. Localmamba: Visual state space model with windowed selective scan. *arXiv preprint arXiv:2403.09338*, 2024. 1

[3] Andrew Kerr, Duane Merrill, Julien Demouth, and John Tran. Cutlass 3.5.1. https://github.com/NVIDIA/cutlass, 2017 (accessed September 10, 2024). 1

[4] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024. 1

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1

[6] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024. 1

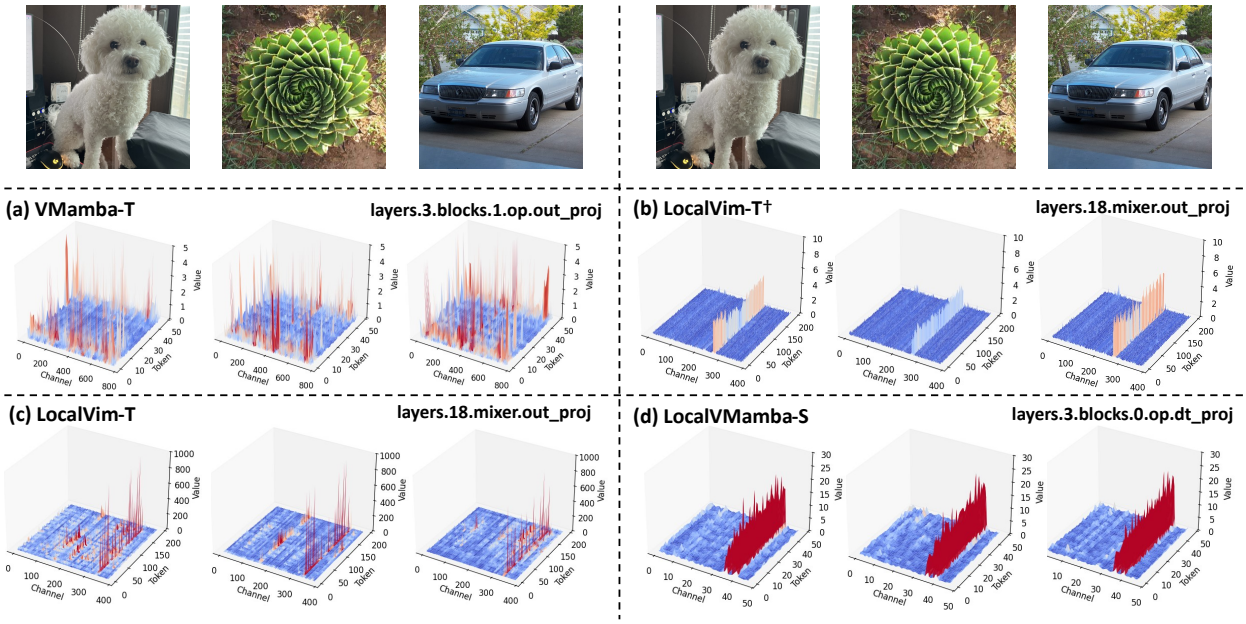

Figure 2S. Activation distributions for different input images. The x-axis is the channel dimension, the y-axis is the token dimension, and the z-axis represents the absolute value. (a) VMamba-T, (b) LocalVim-T†, (c) LocalVim-T, and (d) LocalVMamba-S.

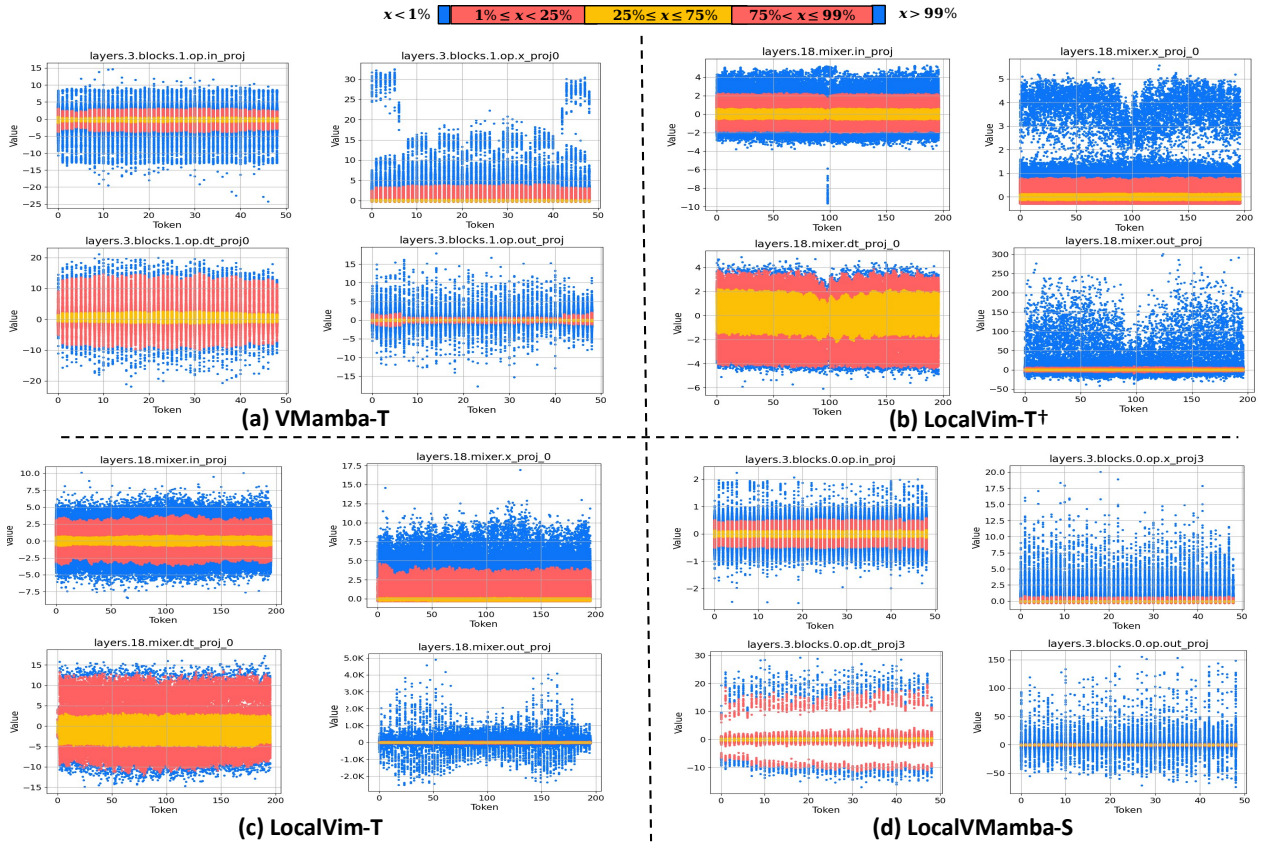

Figure 3S. Activation distributions across token direction. We visualized the above distribution using 32 randomly sampled images. (a) VMamba-T, (b) LocalVim-T†, (c) LocalVim-T and (d) LocalVMamba-S.