Q-MambaIR: Accurate Quantized Mamba for Efficient Image Restoration

Supplementary Material

Yujie Chen[1], Haotong Qin[1]†, Zhang Zhang[2], Michele Magno[1], Luca Benini[1], Yawei Li[1,3]†

[1]ETH Zürich [2]Shenzhen Automotive Research Institute [3]Nanyang Technological University

{li.yawei.ai@gmail.com, haotong.qin@pbl.ee.ethz.ch}

1. Technical Appendices and Supplementary Material

We provide additional information in this supplementary material. In Sec. 1.1, we provide detailed experimental details to aid in the reproducibility of our method. In Sec. 1.2, we provide more experimental results to aid in comparison and further validate the validity of our approach.

1.1. Experimental Setup and Training Details

We adopt a task-specific training strategy across various Image Restoration tasks. Below, we detail the training configurations for each scenario.

1.1.1. Classic Image Super-Resolution (SR)

For classic SR, we adopt the following training configuration: batch size 4, learning rate 1×10^{-4}, L_1 loss, GT size 192×192, and 6 Residual State Space Blocks (ss2D + CAB). We employ DLS to dynamically adjust the quantization range and mitigate peak truncation loss caused by outliers, while L_1 loss helps retain fine image details during optimization.

1.1.2. Lightweight Image Super-Resolution

For lightweight SR, we use a smaller configuration suited for edge deployment: batch size 2, learning rate 2×10^{-4}, L_1 loss, GT size 192×192, and 4 Residual State Space Blocks. RFA enables flexible rounding via adaptive thresholds, preserving high-frequency details under reduced model depth.

1.1.3. Image Denoising (DN)

For image denoising, we use batch size 4, learning rate 1×10^{-4}, Charbonnier loss, GT size 128×128, and 6 Residual State Space Blocks. We combine DLS and RFA to mitigate information loss under aggressive quantization while preserving texture and structure.

1.1.4. JPEG Compression Artifact Reduction (JPEG CAR)

For JPEG CAR, we use batch size 4, learning rate 1×10^{-4}, Charbonnier loss, GT size 128×128, and 6 Residual State Space Blocks. RFA-driven quantization helps preserve edge structures and fine details that are typically degraded by rigid quantization schemes.

1.1.5. Training Milestones

The learning rate is decayed at iterations 10,000, 15,000, 17,500, and 18,750 to refine optimization throughout training.

1.2. Experimental Results

In this section, we provide additional experiments to further demonstrate the effectiveness and general applicability of Q-MambaIR in a wide range of IR tasks.

1.2.1. Comparison on extended tasks

Comparison on Lightweight Image SR. We apply our method to quantize the MambaIR-light models. In the 4-bit case, our Q-MambaIR outperforms LSQ [2] by up to 0.21 dB PSNR on the $2\times$ scale Urban100 dataset. In the 2-bit case, our Q-MambaIR outperforms LSQ [2] by up to 2.49 dB PSNR on the $2\times$ scale Urban100 dataset. Our method provides a possible solution for ultra-low bit quantization. As shown in Tab. 1, this addition further validates the scalability and adaptability of our method across different levels of upsampling.

Comparison on Gaussian Denoising. We extend our gaussian denoising experiment with two other noise level settings by 15 and 25 in Tab. 3. It can be seen that our Q-MambaIR significantly outperforms the state-of-the-art methods at different noise levels, demonstrating the effectiveness and robustness of our method.

1.2.2. Comparison on other IR architectures.

We further quantize both CNN-based and Transformer-based IR architectures with our method. As shown in Tab. 2, our model also outperforms other quantization methods across different architectures in both 2-bit and 4-bit settings. The best performances are highlighted in red.

2. Quantization Details

2.1. Quantized Operators

2.1.1. Weight Quantization

We employ a learnable uniform quantization RFA quantizer for all weight parameters. The complete pseudocode of the RFA quantizer, which follows the forward-

Table 1. Quantitative comparison on **lightweight image SR** with SOTA methods.

Method	Scale	Bit (w/a)	Set5		Set14		B100		Urban100		Manga109	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
MambaIR-light [4]	2×	32/32	38.16	0.9610	34.00	0.9212	32.34	0.9017	32.92	0.9356	39.31	0.9779
MambaQuant* [7]	2×	8/8	38.05	0.9600	33.73	0.9174	32.28	0.9006	32.37	0.9301	38.72	0.9762
Quamba* [1]	2×	8/8	38.07	0.9600	33.81	0.92	32.28	0.9007	32.58	0.9322	38.92	0.9773
MambaQuant* [7]	2×	4/4	25.62	0.5518	24.46	0.5276	24.72	0.5393	23.06	0.5292	24.62	0.5231
Quamba* [1]	2×	4/4	26.22	0.5778	25.02	0.5610	25.47	0.5833	23.53	0.5578	24.55	0.5285
LSQ [2]	2×	4/4	37.94	0.9602	33.71	0.9189	32.18	0.8995	32.28	0.9304	38.81	0.9774
QuantSR [6]	2×	4/4	37.88	0.9598	33.44	0.9166	32.04	0.8977	31.37	0.9216	38.15	0.9756
Q-MambaIR	2×	4/4	37.99	0.9604	33.77	0.9193	32.22	0.9000	32.43	0.9314	38.91	0.9776
LSQ [2]	2×	2/2	36.95	0.9563	32.63	0.9088	31.50	0.8904	29.87	0.9033	36.42	0.9705
QuantSR [6]	2×	2/2	37.31	0.9576	32.86	0.9112	31.67	0.8928	30.39	0.9103	37.09	0.9728
Q-MambaIR	2×	2/2	37.39	0.9580	33.01	0.9129	31.80	0.8950	30.81	0.9161	37.43	0.9738
MambaIR-light [4]	3×	32/32	34.72	0.9296	30.63	0.8475	29.29	0.8099	29.00	0.8689	34.39	0.9495
Quamba* [1]	3×	8/8	35.54	0.9278	30.37	0.8427	28.15	0.8060	28.45	0.8572	33.67	0.9442
MambaQuant* [7]	3×	8/8	34.52	0.9274	30.36	0.8422	29.15	0.8060	28.37	0.8556	33.58	0.9432
Quamba* [1]	3×	4/4	23.09	0.4316	22.32	0.3944	22.13	0.3758	21.01	0.3987	22.41	0.4221
MambaQuant* [7]	3×	4/4	24.42	0.4878	23.26	0.4423	23.12	0.4216	21.45	0.4268	21.77	0.3952
LSQ [2]	3×	4/4	34.41	0.9270	30.34	0.8424	14.00	0.3182	28.23	0.8551	33.66	0.9446
QuantSR [6]	3×	4/4	34.34	0.9265	30.23	0.8390	29.02	0.8023	27.82	0.8459	33.03	0.9407
Q-MambaIR	3×	4/4	34.46	0.9273	30.40	0.8434	29.15	0.8058	28.37	0.8576	33.78	0.9455
LSQ [2]	3×	2/2	32.95	0.9127	29.39	0.8240	14.04	0.3217	26.28	0.8058	30.62	0.9170
QuantSR [6]	3×	2/2	33.34	0.9170	29.58	0.8278	28.60	0.7922	26.64	0.8176	31.36	0.9259
Q-MambaIR	3×	2/2	33.58	0.9197	29.78	0.8318	28.73	0.7955	27.03	0.8285	31.98	0.9316
MambaIR-light [4]	4×	32/32	32.51	0.8993	28.85	0.7876	27.75	0.7423	26.75	0.8051	31.26	0.9175
Quamba* [1]	4×	8/8	32.30	0.8972	28.70	0.7834	27.67	0.7390	26.41	0.7944	30.74	0.9112
MambaQuant* [7]	4×	8/8	32.40	0.8971	28.70	0.7835	27.67	0.7391	26.41	0.7942	30.73	0.9111
Quamba* [1]	4×	4/4	22.47	0.4193	20.68	0.3298	20.33	0.2984	18.84	0.3200	21.23	0.3842
MambaQuant* [7]	4×	4/4	19.72	0.3081	18.61	0.2445	19.00	0.2367	17.63	0.2508	17.52	0.2320
LSQ [2]	4×	4/4	32.16	0.8935	28.57	0.7805	27.57	0.7358	26.11	0.7873	30.37	0.9063
QuantSR [6]	4×	4/4	32.06	0.8925	28.37	0.7757	27.49	0.7327	25.55	0.7674	29.86	0.8987
Q-MambaIR	4×	4/4	32.16	0.8941	28.62	0.7816	27.59	0.7367	26.17	0.7891	30.53	0.9080
LSQ [2]	4×	2/2	30.84	0.8724	27.71	0.7597	27.03	0.7168	24.68	0.7345	27.86	0.8662
QuantSR [6]	4×	2/2	31.08	0.8769	27.84	0.7633	27.11	0.7198	24.82	0.7419	28.26	0.8747
Q-MambaIR	4×	2/2	31.15	0.8779	28.00	0.7665	27.18	0.7227	25.05	0.7499	28.67	0.8800

discretization and backward-approximation strategies described in Sec. 2.3 of the main paper, is provided in Algorithm 1.

The **soft forward threshold** is implemented as the variable T, representing the set of adaptive boundaries $\{T_0, T_1, \ldots, T_N\}$ that define the partitioning of the input space. These thresholds are trained jointly with the model and allow the quantization scheme to dynamically adjust to the underlying data distribution.

The forward hard rounding responsible for discretizing the input values, is realized via the operation $x_{forward} = torch.where(x > threshold, q_{new}, q_{old})$, which assigns to each input the nearest quantization level q_i in a uniform distribution. For an n-bit quantizer, we define the set of quantization centers as a symmetric integer range:

$$\mathcal{Q} = \left\{ q_i \,\middle|\, q_i \in \mathbb{Z},\ -2^{n-1} \leq q_i < 2^{n-1} \right\}.$$

This emulates the behavior of non-uniform to uniform rounding.

To enable efficient gradient flow during quantization-aware training, we adopt a two-phase gradient approximation strategy. The **soft backward approximation** is implemented using a piecewise linear interpolation. Within the active transition interval $[T_0, T_1]$, we apply a linear interpo-

Table 2. Quantitative comparison on **SwinIR** and **EDSR** (4x).

Method	Bit (w/a)	Set5	Set14	B100	Urban100
EDSR [2]	32/32	32.46	28.72	27.72	26.67
QuantSR [6]	4/4	31.93	28.37	27.49	25.55
Ours	4/4	31.96	28.62	27.59	26.17
QuantSR [6]	2/2	31.51	27.84	27.11	24.82
Ours	2/2	31.55	28.00	27.18	25.05
SwinIR [5]	32/32	32.4	28.72	27.72	26.67
QuantSR [6]	4/4	32.11	28.37	27.49	25.55
Q-MambaIR	4/4	32.16	28.62	27.59	26.17
LSQ [2]	2/2	30.84	27.71	27.03	24.68
QuantSR [6]	2/2	31.08	27.84	27.11	24.82
Q-MambaIR	2/2	30.87	28.00	27.18	25.05

lation between T_0 and T_1:

$$y = \frac{\Delta q}{T_1 - T_0} x + q_0 - \frac{\Delta q}{T_1 - T_0} T_0$$

where $\Delta q = q_1 - q_0$ is the quantization level spacing.This design ensures that inputs near the quantization boundaries receive meaningful gradients during training, thereby mitigating the discontinuities introduced by hard rounding.

To address the issue of vanishing gradients in flat quantization regions, we introduce a flat region fallback mechanism, which assigns a small constant slope (0.1) to regions where the quantization function would otherwise be flat and nondifferentiable. Outside this interval, to avoid vanishing gradients, a flat-slope fallback is used:

$$x_{\text{backward}} = \begin{cases} y & \text{if } x > \tau_b \\ 0.1x + q_0 - 0.1T_0 & \text{otherwise} \end{cases}$$

Finally, we decouple the forward and backward behaviors of the quantization function. This is expressed as $x_{forward}.detach() + x_{backward} - x_{backward}.detach()$, which allows the forward pass to use discrete quantized values while the backward pass flows through the soft approximation. This technique is critical for enabling effective end-to-end optimization. We also implemented a CUDA acceleration for this part to release the training burden.

Overall, this code-level design enables RFA to perform discretization-aware learning while preserving gradient flow, making it particularly suitable for state space models and image restoration tasks that are sensitive to quantization artifacts.

2.1.2. Activation Quantization

We employ DDA quantizer for all activations in both convolutional and linear layers. Especially for SSM components, we integrate this adaptive mechanism into all inputs

Algorithm 1 Range-Floating Flexible Allocator (RFA)

Input: Input tensor x, learnable thresholds $T = \{T_0, T_1, ..., T_N\}$, quant levels $Q = \{q_0, q_1, ..., q_{N-1}\}$
Output: Quantized output $\hat{x}$

Initialize $x_{\text{forward}} \leftarrow x$, $x_{\text{backward}} \leftarrow x$
for $i = 0$ to $N - 1$ **do**
  if $i = 0$ **then**
    Set forward threshold $\tau_f \leftarrow T_0$, backward threshold $\tau_b \leftarrow T_0$
    Set quantization step $\Delta q \leftarrow q_1 - q_0$
    Compute interpolated gradient path: $y \leftarrow \frac{\Delta q}{T_1 - T_0} x + q_0 - \frac{\Delta q}{T_1 - T_0} T_0$
    Update $x_{\text{forward}} \leftarrow \text{where}(x > \tau_f, q_0, q_0)$
    Update $x_{\text{backward}} \leftarrow \text{where}(x > \tau_b, y, 0.1x + q_0 - 0.1T_0)$
  else
    Set forward threshold $\tau_f \leftarrow \frac{T_{i-1} + T_i}{2}$, backward threshold $\tau_b \leftarrow T_i$
    if $i < N - 1$ **then**
      $\Delta q \leftarrow q_{i+1} - q_i$
      $y \leftarrow \frac{\Delta q}{T_{i+1} - T_i} x + q_i - \frac{\Delta q}{T_{i+1} - T_i} T_i$
    else
      $y \leftarrow x$ {last interval, identity fallback}
    end if
    Update $x_{\text{forward}} \leftarrow \text{where}(x > \tau_f, q_i, x_{\text{forward}})$
    Update $x_{\text{backward}} \leftarrow \text{where}(x > \tau_b, y, x_{\text{backward}})$
  end if
end for
Final slope fallback: $x_{\text{backward}} \leftarrow \text{where}(x > T_{N-1}, 0.1x + q_{N-1} - 0.1T_{N-1}, x_{\text{backward}})$
Combine forward and backward paths:

$$\hat{x} \leftarrow \text{detach}(x_{\text{forward}}) + x_{\text{backward}} - \text{detach}(x_{\text{backward}})$$

return $\hat{x}$

of selective scan and all main operator in VSSM module. It can dynamically adjust quantized parameters based on input distribution features and handle dynamic range variations in SSM computations:

$$\alpha = |w_1 \cdot \mu + w_2 \cdot \sigma + w_3 \cdot x_{\min} + w_4 \cdot x_{\max}| \quad (1)$$

$$\beta = w_{21} \cdot \mu + w_{22} \cdot \sigma + w_{23} \cdot x_{\min} + w_{24} \cdot x_{\max} \quad (2)$$

where μ and σ are the mean and standard deviation, and w_i are learnable parameters.

2.2. Quantized Modules

Our framework quantizes all major components: (1) all convolutional layers in CAB and residual connections, (2) linear projections in SS2D modules, (3) all SSM internal states and parameters (input states, time steps, state matrices, skip

Table 3. Gaussian color image denoising

Method	Bit	BSD68		Kodak24		McMaster		Urban100	
	(w/a)	σ=15	σ=25	σ=15	σ=25	σ=15	σ=25	σ=15	σ=25
MambaIR [4]	32/32	34.48	31.80	35.42	32.91	35.70	33.35	35.37	32.99
Quamba* [1]	8/8	30.10	30.11	31.29	31.30	31.59	31.60	30.34	30.34
MambaQuant* [7]	8/8	30.49	26.73	30.79	26.31	29.48	27.09	28.62	25.50
Quamba* [1]	4/4	28.26	24.97	28.60	25.29	28.53	24.71	27.58	24.11
MambaQuant* [7]	4/4	20.49	21.00	20.17	20.16	20.43	20.27	22.98	21.10
LSQ [2]	4/4	34.27	29.93	35.14	30.48	35.32	30.06	34.78	28.90
QuantSR [6]	4/4	34.29	31.59	34.97	32.59	34.38	32.91	34.07	32.33
Q-MambaIR	4/4	34.36	31.73	35.26	32.81	35.49	33.21	35.03	32.81
LSQ [2]	2/2	32.68	24.85	33.07	25.09	32.37	25.34	31.66	24.21
QuantSR [6]	2/2	33.76	31.04	34.43	31.87	34.40	31.96	33.63	30.99
Q-MambaIR	2/2	33.81	31.09	34.51	31.95	34.49	32.06	33.69	31.11

connections), and (4) projection matrices for SSM parameters. We demonstrate effective quantization down to 2-bit precision, achieving up to 16× model compression while maintaining competitive performance.

3. Jetson Orin Deployment Workflow

This section provides a complete overview of our deployment workflow on the NVIDIA Jetson Orin platform. The pipeline is designed to ensure stable inference under constrained compute resources, and consists of three major stages: ONNX export, TensorRT engine construction, and on-device inference. We additionally describe practical considerations that arise when deploying low-bit models under real-world embedded hardware constraints.

3.1. Prerequisites

The deployment environment relies on the software and hardware components listed below:

- **NVIDIA Jetson Orin device** with JetPack SDK (including CUDA, cuDNN, TensorRT, and system libraries).
- **Docker environment** providing a reproducible runtime with CUDA, PyTorch, and TensorRT support. We adopt a custom image (`realdn:with_tensorrt`) to ensure consistent operator availability.
- **PyTorch model checkpoint** (`.pth`) containing trained FP32 or quantized model weights.
- **TensorRT toolkit** and the command-line utility `trtexec` for building optimized inference engines.

This setup ensures that the model can be exported, converted, and executed consistently across different Jetson Orin units.

3.2. Step 1: ONNX Model Export

The deployment pipeline begins with exporting the PyTorch model into the ONNX format. This step is executed either on the host machine or within the Docker container. During export, we:

- load the PyTorch checkpoint and perform module rewrapping if required by TensorRT;
- trace or script the model to handle dynamic control flow;
- export the model using `torch.onnx.export` with ONNX opset 18 to match TensorRT's supported operator schema;
- validate the exported ONNX graph using `onnxsim` or `onnxruntime` to ensure graph correctness.

A valid ONNX file is required for the following optimization step.

3.3. Step 2: TensorRT Engine Building

Given the exported ONNX model, we build a TensorRT engine optimized for Jetson Orin's GPU architecture. TensorRT parses the ONNX graph, selects the best available kernels, performs layer fusion, and generates a platform-specific inference engine.

Engine construction is performed using the `trtexec` utility with the following key configuration options:

- `--optShapes=input:1x3x256x256` to specify the optimal static input resolution;
- `--memPoolSize=workspace:4096M` (FP16) or `8192M` (INT8) to allocate sufficient workspace for kernel fusion;
- `--builderOptimizationLevel=1` to ensure broad operator compatibility during graph fusion;
- `--int8` to enable low-precision execution.

TensorRT produces a serialized engine (`.plan`) file, which is directly executable on the device.

Overall, this deployment pipeline ensures that both FP16 and INT8 variants of our model can be executed reliably on Jetson Orin while achieving the performance reported in the main paper.

3.4. Explanation and Summary

We report the end-to-end latency of full-precision MambaIR and its int4 quantized variants on the Jetson Orin platform. We tested them with our **INT8 deployment pipeline**. Below, we provide a concise explanation of why all 4-bit models exhibit similar latency and why Q-MambaIR introduces no overhead.

Although our models are nominally quantized to 4-bit weights and activations, the current *TensorRT* pipeline on Jetson Orin only supports INT8 deployment. Consequently, both the FP32 MambaIR model and the 4-bit variants are internally executed using **INT8/FP16 mixed-precision kernels**.

Despite this, the 4-bit models still exhibit lower latency (110–121 ms) compared to the original FP32 model (198.21 ms). This speedup arises not from true 4-bit arithmetic, but from operator-level optimizations introduced in the Q-MambaIR module, including:

- **Pre-quantized weight layout**: For RFA-based weights, the quantization is applied offline, allowing the engine to directly consume quantized weights without additional preprocessing. This reduces memory accesses and kernel overhead.
- **Activation computation optimizations**: For DDA, the input activations are pre-processed to capture their statistical characteristics, and the selective-scan operations are reorganized to improve memory coalescing and parallel execution on the GPU.

As a result, although the actual computation still uses INT8 kernels, the optimized operator structure allows the nominal 4-bit models to achieve lower latency than the unoptimized FP32 model. Differences among Baseline-4bit, DLS-4bit, RFA-4bit, and Q-MambaIR are negligible in runtime, explaining the similar measured latencies across these variants.

Operator Rearrangement for Efficient Deployment

To enable efficient deployment of our 4-bit models on Jetson Orin, we introduce two implementation optimizations within Q-MambaIR. First, for the weights involved in the RFA-based rounding scheme, we apply the 4-bit transformation *offline* and directly deploy the quantized weights to the edge device. This eliminates any on-device quantization overhead and ensures that the selective-scan operator consumes already–quantized parameters.

Second, for the activation-side transformation in DDA, we optimize the computation of input activations following techniques similar to those used in [3]. Specifically, we restructure the activation computation to reduce redundant operations while accurately capturing the statistical characteristics of the input tensors. This improves numerical fidelity under low-bit quantization without introducing additional kernels or memory-transfer cost.

Together, these optimizations allow Q-MambaIR to retain the same latency as the baseline 4-bit model while benefiting from more stable quantization behavior and improved restoration accuracy.

References

[1] Hung-Yueh Chiang*, Chi-Chih Chang*, Natalia Frumkin, Kai-Chiang Wu, and Diana Marculescu. Quamba: A post-training quantization recipe for selective state space models. In *ICLR*, 2025. 2, 4

[2] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019. 1, 2, 3, 4

[3] Shuhang Gu, Wen Li, Luc Van Gool, and Radu Timofte. Fast image restoration with multi-bin trainable linear units. In *ICCV*, pages 4190–4199, 2019. 5

[4] Hang Guo, Jinmin Li, Tao Dai, Zhihao Ouyang, Xudong Ren, and Shu-Tao Xia. MambaIR: A simple baseline for image restoration with state-space model. In *ECCV*, pages 222–241. Springer, 2024. 2, 4

[5] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. SwinIR: Image restoration using swin transformer. In *ICCV*, pages 1833–1844, 2021. 3

[6] Haotong Qin, Yulun Zhang, Yifu Ding, Yifan liu, Xianglong Liu, Martin Danelljan, and Fisher Yu. QuantSR: Accurate low-bit quantization for efficient image super-resolution. In *NeurIPS*, 2023. 2, 3, 4

[7] Zukang Xu, Yuxuan Yue, Xing Hu, Zhihang Yuan, Zixu Jiang, Zhixuan Chen, Jiangyong Yu, Chen Xu, Sifan Zhou, and Dawei Yang. MambaQuant: Quantizing the mamba family with variance aligned rotation methods. *arXiv preprint arXiv:2501.13484*, 2025. 2, 4