DMQ: Dissecting Outliers of Diffusion Models for Post-Training Quantization

Supplementary Material

1. Implementation details

This section provides a more detailed description of the experimental implementation presented in the main manuscript. Tab. 1 summarizes the hyperparameters used across all experiments for W4A8. The number of sampling steps for generating calibration data is denoted by T, while n represents the amount of calibration data sampled per step. For training, we set the batch size to B and use iteration to denote the number of training steps required to learn the LES factors τ. The adaptive timestep weighting loss is controlled by α, where a higher value prioritizes optimization on early denoising steps. κ determines the agreement threshold for selecting PTS factors, ensuring stability in the voting mechanism. The momentum parameter ξ smooths the moving average update of accumulated loss values, preventing abrupt changes in timestep weights. D represents the range of candidate PTS factors, allowing for optimal scaling adjustments across channels. Specifically, the calibration dataset for the unconditional generation consists of 256 randomly generated samples, sampled using a 20-step DDIM sampler. By including all samples from the intermediate steps, the total number of calibration data points is 5120. For a class-conditional generation, we further utilize calibration data with classifier-free guidance [5] (cfg), where a guidance scale is set to 3.0. For text-guided generation, the data is sampled using a 25-step DDIM sampler, resulting in a total of 6400 calibration data points. These hyperparameters are carefully tuned for different datasets and model architectures to achieve robust and stable quantization performance.

2. Experiment setup details

We build upon official PyTorch implementation[1] of LDM and used provided pre-trained models. For a fair comparison, all methods quantize each layer except the input and output layers, which remain in full precision per common practice. Specifically, the original implementaion of PTQD [4] and TFMQ-DM [6] do not quantize *skip-connection*, *downsample*, and *upsample* layers. Thus, we modify their code to quantize those layers for fair comparison. Then we directly run original code provided by the baselines.

To control for metric fluctuations caused by differences in generated samples, we unified all baselines into a single codebase and ensured identical samples were generated using the same random seed. Evaluation was conducted using the implementation of guided-diffusion[2].

[1]https://github.com/CompVis/stable-diffusion
[2]https://github.com/openai/guided-diffusion

3. Quantization granularity

In this section, we provide a detailed explanation of quantization granularity. This expands on the concepts introduced in the Preliminary section of the main paper, providing additional details. If you are already familiar with quantization, you may skip this section. The basic quantization and dequantization functions follow Eq. 3 in the main paper. Note that modern libraries implement convolutions using algorithms such as img2col [1] or implicit GEMM [2], making them functionally equivalent to linear layers. Therefore, we describe our approach in the context of linear layers. To recap, the linear layer defined in the main text is given by:

$$\mathbf{Y} = \mathbf{X}\mathbf{W} \approx (s^{(\mathbf{X})}\tilde{\mathbf{X}})(s^{(\mathbf{W})}\tilde{\mathbf{W}}) \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{B\times C_{in}}$ is the activation, $\mathbf{W} \in \mathbb{R}^{C_{in}\times C_{out}}$ is the weight, and $\mathbf{Y} \in \mathbb{R}^{B\times C_{out}}$ is the output. $s^{(\mathbf{X})}$ and $s^{(\mathbf{W})}$ denote the scales for $\mathbf{X}$ and $\mathbf{W}$, respectively.

To investigate which quantization granularity is appropriate, we first examine per-element quantization—the finest granularity—where each element of a weight or activation has its own scale. Matrix multiplication with per-element quantization can be expressed as:

$$\mathbf{Y}_{ij} \approx \left(\sum_{k=1}^{p}(s_{ik}^{(\mathbf{X})}\tilde{\mathbf{X}}_{ik})\cdot(s_{kj}^{(\mathbf{W})}\tilde{\mathbf{W}}_{kj})\right). \tag{2}$$

However, since the scales are floating-point values, including them inside the summation results in floating-point matrix multiplication, defeating the purpose of quantization. To enable efficient integer matrix multiplication, it is necessary to factor out the scale s from the summation. This requires the scales to be independent of k [14]. Specifically, activation scale $s^{(\mathbf{X})}$ must be constant across columns (i.e., one scale per row), and weight scale $s^{(\mathbf{W})}$ must be constant across rows (i.e., one scale per column). Under this condition, the matrix multiplication can be formulated as:

$$\mathbf{Y}_{ij} \approx s_i^{(\mathbf{X})}s_j^{(\mathbf{W})}\left(\sum_{k=1}^{C_{in}}\tilde{\mathbf{X}}_{ik}\cdot\tilde{\mathbf{W}}_{kj}\right). \tag{3}$$

This formulation shows that integer matrix multiplication is feasible when activations are quantized with per-tensor or per-sample (row) granularity and weights are quantized in per-tensor or per-channel (column) granularity. Since per-sample quantization requires online computation, per-tensor quantization is generally used for activations. In this paper, we use per-tensor activation quantization, meaning that $s^{(\mathbf{X})}$ is a scalar value. However, for consistency with the general formulation, we retain the index i rather than removing it.

Experiment	T	n	cfg	N	α	κ	ξ	D	B	$iteration$
LDM-8 LSUN-Church (uncond.)	20	256	–	5120	25	0.85	0.95	3	32	4000
LDM-4 LSUN-Bedroom (uncond.)	20	256	–	5120	20	0.85	0.95	3	32	6000
LDM-4 FFHQ (uncond.)	20	256	–	5120	20	0.85	0.95	3	32	6000
LDM-4 ImageNet (class cond.)	20	256	✓	10240	20	0.85	0.95	3	32	6000
Stable Diffusion (text cond.)	25	256	✗	6400	20	0.85	0.95	3	8	6000

Table 1. Hyperparameters for all experiments. T/c is the number of sampling steps for generating the calibration data. n is the amount of calibration data per sampling step, and N is the size of calibration dataset. *cfg* indicates whether classifier-free guidance was used or not. B is the batch size. *iteration* is the number of training steps used to learn τ.

4. Details of quantization approach

In this section, we provide a detailed explanation of the quantization process, combined with proposed Learned Equivalent Scaling (LES) and Power-of-Two Scaling (PTS). Note that LES are applied to all layers and LES combined with PTS are only applied to skip connection layers. Before introducing each method, we first recap the basic quantization process.

Basic quantization. The standard quantization and dequantization functions for activations and weights are defined as follows:

$$\tilde{\mathbf{X}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{X}}{s^{(\mathbf{X})}} \right\rceil, l, u\right), \mathbf{X} \approx s^{(\mathbf{X})} \cdot \tilde{\mathbf{X}}, \tag{4}$$

$$\tilde{\mathbf{W}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{W}}{s^{(\mathbf{W})}} \right\rceil, l, u\right), \mathbf{W} \approx s^{(\mathbf{W})} \cdot \tilde{\mathbf{W}}, \tag{5}$$

where $\lfloor\cdot\rceil$ denotes round-to-nearest-integer operator. clamp$(\cdot)$ truncates values to $[l, u]$, yielding a low-bit representation $\tilde{\mathbf{X}}$ of $\mathbf{X}$. Here, the range $[l, u]$ is determined by the bit-width of the quantization. When s is a vector, $\mathbf{X}/s$ implies channel-wise division.

Quantization with LES. To recap Eq. 5 from the main paper, matrix multiplication with Equivalent Scaling can be expressed as:

$$\mathbf{Y} = (\mathbf{X}/\tau)\left(\tau^\top \odot \mathbf{W}\right) = \hat{\mathbf{X}}\hat{\mathbf{W}}, \tag{6}$$

where $/$ and $\odot$ denote channel-wise division and multiplication, respectively. We apply quantization and dequantization to the scaled activation $\hat{\mathbf{X}}$ and weight $\hat{\mathbf{W}}$, formulated as:

$$\tilde{\mathbf{X}} = \text{clamp}\left(\left\lfloor \frac{\hat{\mathbf{X}}}{s^{(\mathbf{X})}} \right\rceil, l, u\right), \hat{\mathbf{X}} \approx s^{(\mathbf{X})} \cdot \tilde{\mathbf{X}}, \tag{7}$$

$$\tilde{\mathbf{W}} = \text{clamp}\left(\left\lfloor \frac{\hat{\mathbf{W}}}{s^{(\mathbf{W})}} \right\rceil, l, u\right), \hat{\mathbf{W}} \approx s^{(\mathbf{W})} \cdot \tilde{\mathbf{W}}. \tag{8}$$

However, this formulation presents a key challenge. Since the weight $\mathbf{W}$ remains fixed, $\tau^\top \odot \mathbf{W}$ can be precomputed and stored for direct use. In contrast, activation $\mathbf{X}$ change dynamically, making it impractical to precompute and store their scaled values. As a result, directly applying this method would require dividing by τ at every inference step just before quantization, introducing additional computational overhead. To resolve this, we leverage the following transformation, which allows τ to be seamlessly fused into the scale factor:

$$\frac{\hat{\mathbf{X}}}{s^{(\mathbf{X})}} = \frac{\mathbf{X}/\tau}{s^{(\mathbf{X})}} = \frac{\mathbf{X}}{\tau \odot s^{(\mathbf{X})}}. \tag{9}$$

This eliminates the need to divide activations by τ at every step, effectively reducing the computational overhead. This optimization is feasible because we use static quantization, where the activation scale remains constant. In contrast, dynamic quantization, which updates the activation scale at each step, does not allow fusing τ into the scale. The quantization functions incorporating this method are formulated as follows:

$$\tilde{\mathbf{X}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{X}}{\{\tau \odot s^{(\mathbf{X})}\}} \right\rceil, l, u\right) \tag{10}$$

$$\tilde{\mathbf{W}} = \text{clamp}\left(\left\lfloor \frac{\{\tau^\top \odot \mathbf{W}\}}{s^{(\mathbf{W})}} \right\rceil, l, u\right), \tag{11}$$

where $\{\cdot\}$ indicates pre-computed values that are stored in memory. The matrix multiplication with dequantization remains the same as in Eq. (3). This approach allows us to eliminate additional overhead while efficiently integrating LES into the quantization process.

Quantization with LES and PTS. For skip connection layers in residual block, where extreme outliers are observed, PTS is applied alongside LES. The quantization and dequantization functions incorporating both LES and PTS are formulated as follows:

$$\tilde{\mathbf{X}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{X}}{\{2^{\delta} \odot \tau \odot s^{(\mathbf{X})}\}} \right\rceil, l, u\right), \hat{\mathbf{X}} \approx (2^{\delta} \odot s^{(\mathbf{X})}) \cdot \tilde{\mathbf{X}} \tag{12}$$

$$\tilde{\mathbf{W}} = \text{clamp}\left(\left\lfloor \frac{\{\tau^\top \odot \mathbf{W}\}}{s^{(\mathbf{W})}} \right\rceil, l, u\right), \hat{\mathbf{W}} \approx s^{(\mathbf{W})} \cdot \tilde{\mathbf{W}}. \tag{13}$$

Additionally, the matrix multiplication with dequantization can be expressed as:

$$\mathbf{Y}_{ij} \approx \left(\sum_{k=1}^{C_{in}} (2^{\delta_k} s_i^{(\mathbf{X})} \tilde{\mathbf{X}}_{ik}) \cdot (s_j^{(\mathbf{W})} \tilde{\mathbf{W}}_{kj})\right). \tag{14}$$

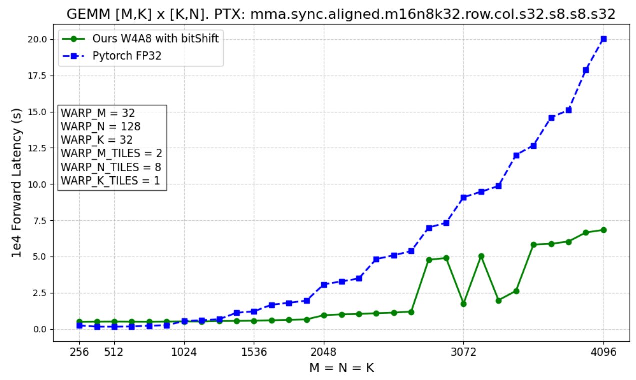

Figure 1. Comparison on latency between pytorch fp32 GEMM and our custom W4A8 GEMM kernel including quantization, bit-shifting on weight, GEMM, and dequantization.

Since the PTS factor 2^{δ_k} is indexed by k, it cannot be factored out of the summation. Moreover, modern GPU architectures do not natively support multiply-bitshift-add operations, making direct computation of the above formulation inefficient. To address this, we apply bit-shifting to weights immediately after loading them during kernel execution, ensuring efficient computation:

$$\tilde{\mathbf{W}}_{kj}^{shifted} = 2^{\delta_k}\tilde{\mathbf{W}}_{kj} = \tilde{\mathbf{W}}_{kj} \ll \delta_k, \tag{15}$$

where $\ll$ denotes the left-bit-shift operation. Dequantization functions incorporating Eq. (15) are formulated as:

$$\mathbf{Y}_{ij} \approx s_i^{(\mathbf{X})} s_j^{(\mathbf{W})} \left(\sum_{k=1}^{C_{in}} \tilde{\mathbf{X}}_{ik} \cdot \tilde{\mathbf{W}}_{kj}^{shifted} \right). \tag{16}$$

As a result, PTS effectively handles extreme outliers with minimal computational overhead, requiring only a bit-shifting operation on weight.

5. Speedup of Power-of-Two Scaling

To evaluate the speedup effect of Power-of-Two Scaling in quantization, we implemented a custom CUDA kernel that integrates quantization, bit-shifting on weights, GEMM (General Matrix Multiplication), and dequantization. The comparison of latency between our custom W4A8 GEMM kernel and PyTorch FP32 GEMM is shown in Fig. 1. Our method achieves up to $5.17\times$ speedup over FP32 at M=3072, demonstrating the efficiency of our approach. Although bit-shifting introduces minor additional overhead, these results confirm that it can be handled highly efficiently in practice. Additionally, since Power-of-Two Scaling is applied only to a small subset of the network (specifically, skip connection layers with extreme outliers), its overall impact on network latency remains minimal.

6. Limitation and future Work

In this paper, we consider robust quantization to alleviate the inter-channel variance of the neural network, which is often overlooked by the current quantization methods for diffusion models. Consequently, we propose Learned Equivalent Scaling and channel-wise Power-of-Two Scaling, the optimization processes that consider the inter-channel variance and the iterative nature of the generative process in diffusion models.

As our approach follows the common configuration used in previous research and does not contain specific assumptions for the calibration dataset or intermediate distribution of diffusion trajectory, some settings for quantization can be further optimized to improve the upper bound of the proposed method. For instance, we have adopted uniform timestep sampling for calibration data, following Q-Diffusion [7]. However, one can freely adopt other data composition strategies such as sampling from normally distributed timesteps [11] or aligning more closely with original training samples [8]. Furthermore, various quantization techniques, including noise correction [4, 15] and introducing timestep-specific quantization parameters [6, 12], can be orthogonally adapted to our method.

Nevertheless, various quantization methods for diffusion models, including ours, only consider reconstructing the output of diffusion models exactly with low-bit precision. Moving towards robustness and efficiency in more extreme low-bit quantization, such as 2- or 3-bit representation, analyzing the feature map of diffusion models and prioritizing the salient feature could prove more effective, such as utilizing mixed-precision weight [13]. In addition, extending the our approach to include various downstream tasks that require efficient fine-tuning, such as personalized text-to-image diffusion models [10, 16], holds significant potential for the practical application of our method.

7. Societal impact

While our research primarily focuses on the quantization of diffusion models to achieve efficient generation, it is important to acknowledge the broader societal impact of generative models. By enabling more efficient generation, our approach could make generative models more accessible to a wider range of users, increasing the potential for both positive and negative applications such as generating deepfakes, NSFW content, or copyright-infringing materials. Integrating safeguards like erasing NSFW content from diffusion models [3] into the quantization process should be considered to mitigate potential risks.

8. More Visual Results

Figs. 2 to 5 shows samples generated with W4A6 quantized models using state-of-the art methods on various datasets.

The results demonstrate that the proposed quantization method effectively generates high-quality images while maintaining the structure of samples from full-precision models.

References

[1] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006. 1

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 1

[3] Rohit Gandikota, Joanna Materzynska, Jaden Fiotto-Kaufman, and David Bau. Erasing concepts from diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2426–2436, 2023. 3

[4] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 1, 3

[5] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. 1

[6] Yushi Huang, Ruihao Gong, Jing Liu, Tianlong Chen, and Xianglong Liu. Tfmq-dm: Temporal feature maintenance quantization for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7362–7371, 2024. 1, 3, 5, 6

[7] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023. 3, 5, 6

[8] Xuewen Liu, Zhikai Li, Junrui Xiao, and Qingyi Gu. Enhanced distribution alignment for post-training quantization of diffusion models. *arXiv preprint arXiv:2401.04585*, 2024. 3

[9] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 5, 6

[10] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 22500–22510, 2023. 3

[11] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1972–1981, 2023. 3

[12] Junhyuk So, Jungwon Lee, Daehyun Ahn, Hyungjun Kim, and Eunhyeok Park. Temporal dynamic quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 3

[13] Yang Sui, Yanyu Li, Anil Kag, Yerlan Idelbayev, Junli Cao, Ju Hu, Dhritiman Sagar, Bo Yuan, Sergey Tulyakov, and Jian Ren. Bitsfusion: 1.99 bits weight quantization of diffusion model. *arXiv preprint arXiv:2406.04333*, 2024. 3

[14] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020. 1

[15] Yuzhe Yao, Feng Tian, Jun Chen, Haonan Lin, Guang Dai, Yong Liu, and Jingdong Wang. Timestep-aware correction for quantized diffusion models. *arXiv preprint arXiv:2407.03917*, 2024. 3

[16] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023. 3

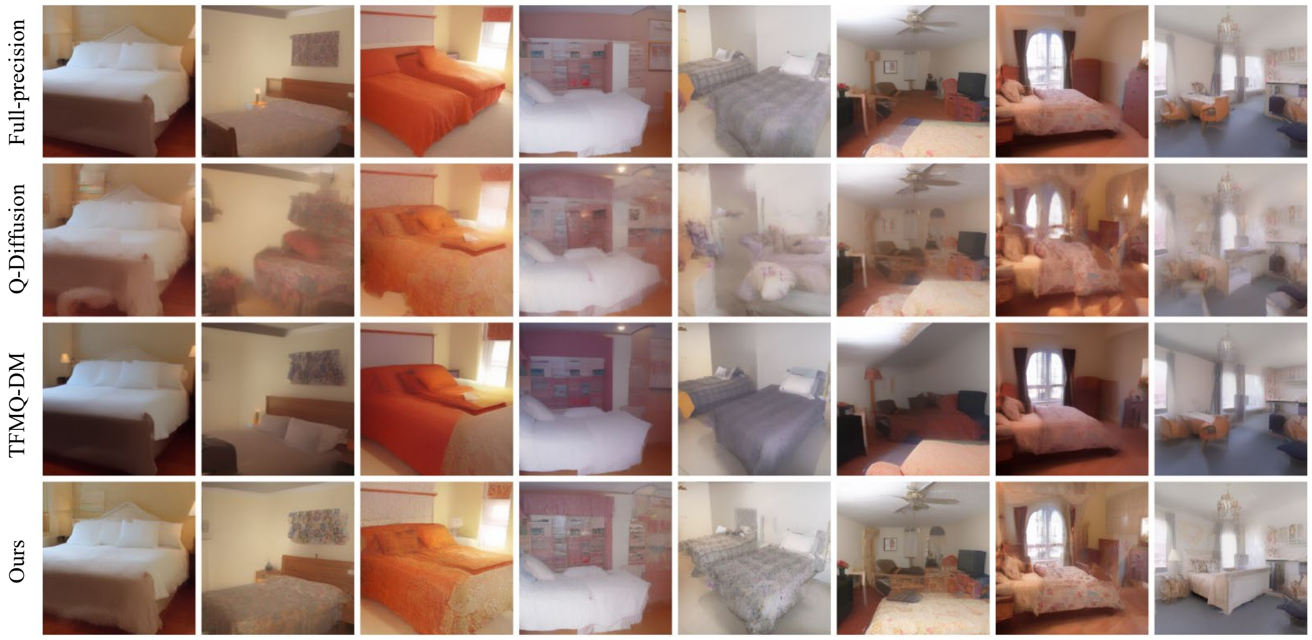

Figure 2. Visualization of samples on LSUN-Bedrooms 256×256 generated by full precision LDM [9] and W4A6 quantized models using Q-Diffusion [7], TFMQ-DM [6], and ours.

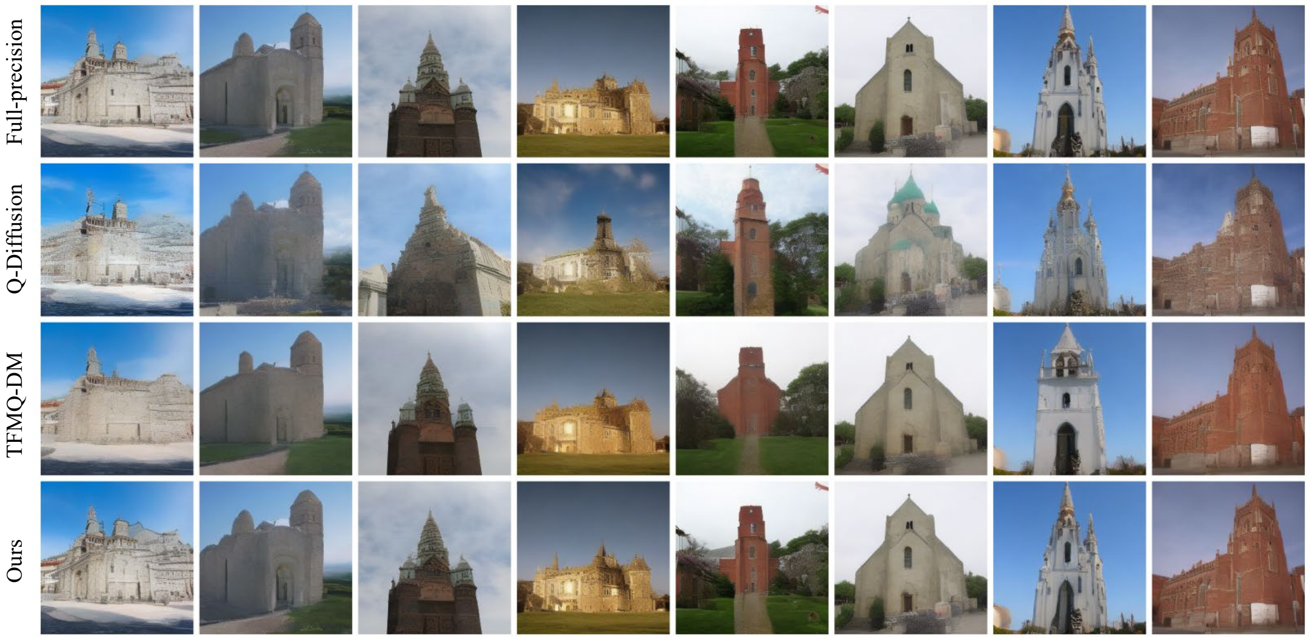

Figure 3. Visualization of samples on LSUN-Chruches 256×256 generated by full precision LDM [9] and W4A6 quantized models using Q-Diffusion [7], TFMQ-DM [6], and ours.

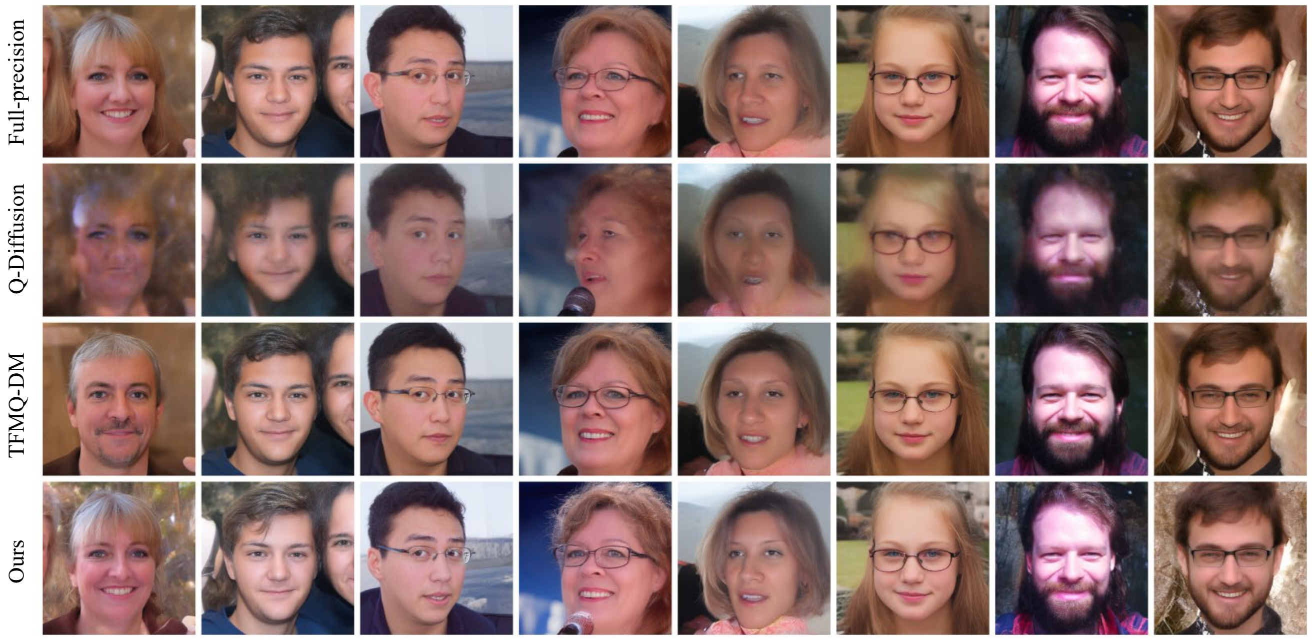

Figure 4. Visualization of samples on FFHQ 256×256 generated by full precision LDM [9] and W4A6 quantized models using Q-Diffusion [7], TFMQ-DM [6], and ours.

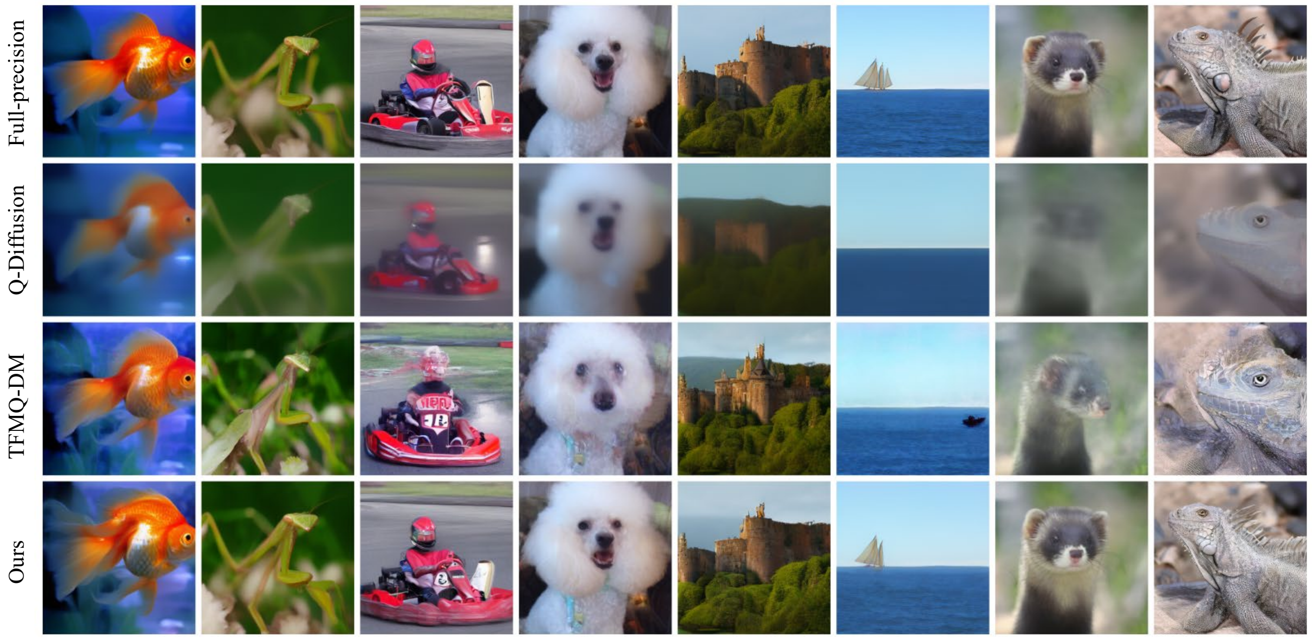

Figure 5. Visualization of samples on ImageNet 256×256 generated by full precision LDM [9] and W4A6 quantized models using Q-Diffusion [7], TFMQ-DM [6], and ours.