# Real-Time On-Device Diffusion: Practical Acceleration via Fused Low-Bit Kernels

Xia Ruize
Nanjing Foreign Language School
xiaruize0911@gmail.com

Weizhi Gao
North Carolina State University
wgao23@ncsu.edu

Jiapeng Hu
North Carolina State University
jhu48@ncsu.edu

Xiaorui Liu
North Carolina State University
xliu96@ncsu.edu

## Abstract

*Diffusion models have demonstrated remarkable effectiveness as generative models, but their high computational cost, caused by the iterative denoising process and computationally heavy backbone networks, remains a major barrier to on-device and real-time deployment. Modulated Diffusion (MoDiff) provides a post-training quantization framework that supports highly efficient low-bit activation quantization. However, prior work evaluates the acceleration benefits of MoDiff only through operation-count analysis, without validating them on real hardware. In this paper, we present a kernel implementation of MoDiff for practical hardware acceleration. In particular, we propose a cache-update fusion paradigm that eliminates additional I/O operations during inference. Experiments and ablation studies demonstrate that our implementation achieves up to a 1.8× runtime speedup over FP32 models and up to a 42.2% reduction in memory I/O usage compared to FP32.*

## 1. Introduction

Diffusion models achieve state-of-the-art generative performance in both image and language domains [1–3]. They generate complex outputs by iteratively refining noise samples via a learned reverse diffusion process [1, 4]. However, this iterative denoising and the use of heavy backbones incur high inference overhead, limiting deployment on edge devices [3].

Quantization improves efficiency by converting high-precision floating-point values to low-precision integers, reducing memory and computation [5, 6]. While weight quantization is relatively straightforward, activation quantization is harder due to wide dynamic ranges and outliers [7]. As a result, mismatched bitwidths between weights and activations (e.g., W4A8 or W1A4) often reduce the effectiveness of hardware acceleration.

Modulated Diffusion (MoDiff) [8] is a recent method that alleviates the challenges of low-bit activation quantization. Exploiting redundancy across consecutive generation steps, MoDiff can be combined with existing schemes to reduce activation precision from 8 bits to 4 bits without degrading performance. However, its benefits have so far been assessed only theoretically, in terms of model size and binary operations (BOPs), and its practical acceleration has not been studied.

We address this gap by designing efficient kernel functions for MoDiff. We refine its notation to identify shared cache memory across the computation pipeline, and develop fused kernels for normalization and convolution layers to remove extra cache I/O. Experiments with the CUTLASS library show up to $1.8\times$ GPU speedup, moving diffusion models closer to practical edge deployment. Our main contributions are as follows:

- We identify where cache memory is shared in the forward pipeline, providing guidance for kernel design.
- We build the kernels using the CUTLASS library and fuse the cache-update operations into the layer kernels to avoid excessive I/O overhead.
- We conducted benchmarking experiments and demonstrated performance optimizations in both computational throughput and memory-access efficiency.

## 2. Related Work

### 2.1. Diffusion Models

Diffusion models constitute a major advancement in generative modeling, facilitating notable progress in image synthesis, video generation, and molecular design [2, 9, 10]. These models function by gradually adding noise to data and then learning to reverse this process to generate new samples [11]. However, the iterative denoising process of diffusion models leads to high computational costs, limiting their applicability to edge devices and real-time scenarios [9, 12]. To address this issue, numerous studies have sought to reduce the number of sampling steps by developing more advanced ODE solvers. For instance, Denoising Diffusion Implicit Models (DDIMs) propose a non-Markovian formulation of the diffusion process, thereby substantially reducing the required sampling steps [13].

### 2.2. Modulated Diffusion

In contrast to approaches that reduce the number of denoising steps, quantization aims to enhance the inference efficiency of diffusion models by encoding floating-point values as low-bit integers [14, 15]. Post-training quantization (PTQ) is particularly appealing because it eliminates the need for retraining [16]. However, PTQ faces significant challenges in activation quantization due to the presence of activation outliers and dynamic input ranges [17–24]. Additionally, discrepancies in bitwidth between weights and activations further diminish the computational efficiency of quantization on hardware [25].

Modulated Diffusion Models (MoDiff) [8] introduce a general framework to address activation-quantization challenges in PTQ methods. By exploiting redundancy throughout the generation process and incorporating error compensation mechanisms, MoDiff reduces quantization error and prevents error accumulation. As a result, MoDiff reduces activation quantization from 8 bits to 4 bits without compromising performance relative to existing baselines. Nevertheless, the original evaluation of MoDiff is limited to theoretical bit-operation (BOP) analysis and does not include empirical hardware-acceleration results.

## 3. Method

This section details the kernel implementation of MoDiff. We begin by refining the notation of MoDiff to clearly distinguish integer operations from floating-point operations. We then describe how the kernels are fused with cache updates in the U-Net architecture.

### 3.1. Integer Arithmetic Formulation

**Diffusion Models.** sing Denoising Diffusion Probabilistic Models (DDPMs) as an example [9], the forward process

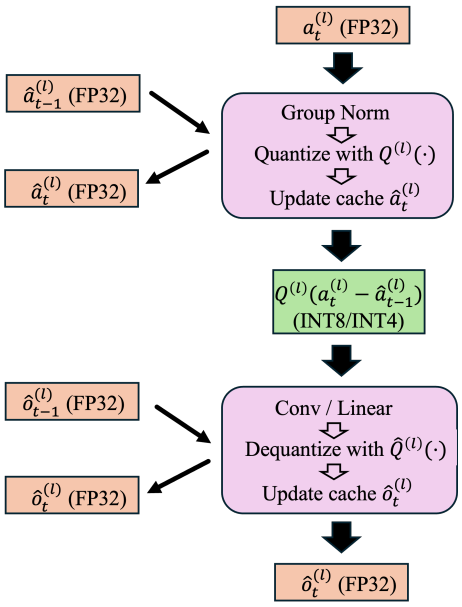


Figure 1. Kernel Design for MoDiff. The quantization operation and the update of $\hat{\mathbf{a}}_t^{(l)}$ are fused into the normalization layers, while the dequantization operation and the update of $\hat{\mathbf{o}}_t^{(l)}$ are fused into the computational layers.

incrementally adds noise to the image at each step, gradually transforming the data distribution into a standard Gaussian distribution:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = N(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I). \quad (1)$$

Meanwhile, the reverse process progressively denoises the Gaussian distribution, reconstructing the original image distribution step by step with a denoising network:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = N(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 I), \quad (2)$$

where $\mu_\theta(\mathbf{x}_t, t)$ is predicted by a neural network, while $\sigma_t$ is typically set to $\beta_t$. With this parametrization, the sampling process can be expressed as:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}, \quad (3)$$

where $\bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_t)$, and $\epsilon_\theta(\mathbf{x}_t, t)$ represents a U-Net that predicts the noise.

**Quantization.** Given FP input $\mathbf{x}$, the quantizer $Q$ converts $\mathbf{x}$ into an integer $\mathbf{x}_{\text{int}}$ and the dequantizer $\hat{Q}$ converts it back to the floating-point numbers $\mathbf{x}_{\text{deq}}$:

$$\mathbf{x}_{\text{int}} = Q(\mathbf{x}) = \text{clamp}(\left\lfloor \frac{\mathbf{x}}{s} \right\rfloor + \mathbf{z}; 0, 2^b - 1), \quad (4)$$

$$\mathbf{x}_{\text{deq}} = \hat{Q}(\mathbf{x}_{\text{int}}) = s(\mathbf{x}_{\text{int}} - \mathbf{z}), \quad (5)$$

where $b$ represents the quantization bitwidth, and clamp$(\cdot)$ enforces value cut-offs between two integer bounds. The parameters $s$ and $\mathbf{z}$ correspond to the scale factor and zero point, respectively. For notational simplicity, we denote integer numbers with INT8/INT4 and floating-point numbers with FP32, where the numbers indicate the bitwidth.

**Modulated Diffusion.** Modulated diffusion operates on linear operators in the denoising network, such as linear and convolutional layers. We take the $l$-th convolutional layer as an example. Its weight tensor is quantized, and for simplicity we denote the quantized weight as $\mathbf{w}_{\text{int}}^{(l)}$. We use $*$ to denote FP convolution and $\circledast$ to denote INT convolution. At time step $t$, the input and output of the $l$-th convolutional layer are denoted by $\mathbf{a}_t^{(l)}$ and $\mathbf{o}_t^{(l)} = \mathbf{w}_{\text{int}}^{(l)} * \mathbf{a}_t^{(l)}$, respectively. Since the temporal difference $\mathbf{a}_t^{(l)} - \mathbf{a}_{t+1}^{(l)}$ is more amenable to quantization, modulated quantization applies the activation quantizer $Q^{(l)}$ to this difference and uses it to approximate $\mathbf{o}_t^{(l)}$ with $\hat{\mathbf{o}}_t^{(l)}$:

$$\hat{\mathbf{a}}_t^{(l)} = \hat{Q}^{(l)}\left(Q^{(l)}(\mathbf{a}_t^{(l)} - \hat{\mathbf{a}}_{t+1}^{(l)})\right) + \hat{\mathbf{a}}_{t+1}^{(l)}, \tag{6}$$

$$\hat{\mathbf{o}}_t^{(l)} = \hat{Q}^{(l)}\left(\mathbf{w}_{\text{int}}^{(l)} \circledast Q^{(l)}(\mathbf{a}_t^{(l)} - \hat{\mathbf{a}}_{t+1}^{(l)})\right) + \hat{\mathbf{o}}_{t+1}^{(l)} \tag{7}$$

$$\approx \mathbf{w}_{\text{int}}^{(l)} * \mathbf{a}_t = \mathbf{o}_t^{(l)}, \tag{8}$$

where $t = T-1, \ldots, 1$, and $\hat{\mathbf{a}}_t^{(l)}$ serves as an intermediate variable to prevent error accumulation during generation by ensuring $\hat{\mathbf{o}}_t^{(l)} = \mathcal{A}(\hat{\mathbf{a}}_t^{(l)})$. The process is initialized with $\hat{\mathbf{a}}_T^{(l)} = \mathbf{a}_T^{(l)}$ and $\hat{\mathbf{o}}_T^{(l)} = \mathbf{w}_{\text{int}}^{(l)} * \mathbf{a}_T^{(l)}$. The rewrite shows that all convolutional operations are performed using integer arithmetic. Although this approach is effective, introducing the caches $\hat{\mathbf{a}}_t^{(l)}$ and $\hat{\mathbf{o}}_t^{(l)}$ necessitates accumulating historical computations, which poses challenges for kernel design in hardware acceleration.

### 3.2. Kernel Fusion

We design specialized kernel fusion methods for MoDiff to integrate cache accumulation into layer computation, thereby avoiding additional I/O overhead [26–28]. Specifically, in the backbone, each convolutional layer is preceded by a GroupNorm layer. Therefore, we fuse the quantizer $Q$ and the update of the cache $\hat{\mathbf{a}}_t$ into the GroupNorm layer. We further fuse the dequantization operation $\hat{Q}$ and the update of the cache $\hat{\mathbf{o}}_t$ into the post-accumulation stage of the convolutional layer. As shown in Figure 1, the GroupNorm layer takes the FP32 input $\mathbf{a}_t^{(l)}$ and the cache $\hat{\mathbf{a}}_{t+1}^{(l)}$ as inputs, and produces two outputs: the integer activation $Q^{(l)}\left(\mathbf{a}_t^{(l)} - \hat{\mathbf{a}}_{t+1}^{(l)}\right)$ for the subsequent forward computation, and the FP32 cache $\hat{\mathbf{a}}_t^{(l)}$ for the next diffusion step. The convolutional layer then takes the integer input $Q^{(l)}\left(\mathbf{a}_t^{(l)} - \hat{\mathbf{a}}_{t+1}^{(l)}\right)$ together with the cache $\hat{\mathbf{o}}_{t+1}^{(l)}$, and outputs the FP32 activation $\hat{\mathbf{o}}_t^{(l)}$ for forward propagation, which is also stored as the cache for the next step. Our design avoids additional I/O operations for the cache, improving the efficiency of the low-bit inference pipeline [27].

## 4. Experiments

In this section, we evaluate the performance of our implementation on benchmark tasks. We begin by introducing the experimental settings. We then present the main results with runtime measurements. Finally, we conduct ablation studies for a fine-grained analysis of the acceleration.

### 4.1. Experimental Settings

**Dataset, Model, and Measurement.** We conduct experiments on the LSUN-Church-Outdoor dataset at a resolution of $256 \times 256$ [29]. For the model, we use a Latent Diffusion Model with a downsampling factor of 8, denoted LDM-8 [30]. We use the DDIM sampler with 200 sampling steps and $\eta = 0$ [13]. To assess the model's efficiency, we measure the runtime per generated sample and the DRAM transfer volume. We also report the Fréchet Inception Distance (FID) [31] evaluated on 50k generated images to assess the generation quality.

**Quantization Method.** We apply channel-wise static MSE quantization for the weights and dynamic tensor-wise quantization for the activations. To better handle activation outliers, we integrate SmoothQuant using training images from LSUN-Churches [23].

**Hardware and Computation Platform.** We conduct experiments on an NVIDIA A40 GPU running Ubuntu 22.04 Server. Our software environment uses CUDA 12.4.1, Python 3.11, and PyTorch 2.4.0.

### 4.2. Main Results

We generate 128 images with a batch size of 32 to evaluate runtime efficiency. Our experiments are conducted under FP32, INT8, and INT4 settings. For notational simplicity, we denote each configuration by its quantization method and precision. For instance, the method without modulated computation under INT8 precision is denoted as "Baseline INT8", while the method with modulated computation is denoted as "MoDiff INT8". We draw the following conclusions from the results.

**Running Time Speedup.** As shown in Table 1, MoDiff achieves significant speedups over FP32 generation. Specifically, INT8-quantized generation with MoDiff achieved a speedup of $1.668\times$, while INT4-quantized generation with MoDiff achieved a speedup of $1.800\times$. It demonstrates the effectiveness of the optimizations and highlights MoDiff's potential to accelerate deep learning models.

**Efficient Memory Usage.** As shown in Table 1, the DRAM transfer is substantially lower than that of FP32. For

Table 1. Running time and DRAM transfer comparison on LSUN-Churches. DRAM transfers include weight, activation, output, cache read, and cache write. Speedup and DRAM transfer reduction are calculated against FP32.

| | Runtime | | DRAM Transfer (GB) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Mode | ms/Sample | Speedup | Weight | Act | Output | Cache Rd | Cache Wr | Total | Ratio |
| FP32 | 613.4 | 1.000× | 219.2 | 26.3 | 19.2 | 0.0 | 0.0 | 264.6 | 100% |
| Baseline INT8 | 333.0 | 1.842× | 66.2 | 26.3 | 19.2 | 0.0 | 0.0 | 111.6 | 42.2% |
| **MoDiff INT8 (Ours)** | **367.7** | **1.668**× | 66.2 | 26.3 | 19.2 | 51.6 | 36.9 | **200.2** | **75.6%** |
| Baseline INT4 | 306.2 | 2.003× | 42.6 | 26.3 | 19.2 | 0.0 | 0.0 | 88.0 | 33.2% |
| **MoDiff INT4 (Ours)** | **340.9** | **1.800**× | 42.6 | 26.3 | 19.2 | 51.6 | 36.9 | **176.6** | **66.7%** |

Table 2. Component-wise time cost and speedup. The speedup is compared to FP32.

| Component | FP32 | INT8 | Speedup | INT4 | Speedup |
|---|---|---|---|---|---|
| Conv2d | 1410.0 | 1355.3 | 1.04× | 1126.3 | 1.26× |
| Attention | 1378.5 | 750.4 | 1.77× | 268.8 | 1.84× |
| Linear | 69.4 | 552.4 | 0.13× | 547.1 | 0.13× |
| GroupNorm | 7.5 | 7.7 | 0.98× | 60.1 | 1.00× |
| SiLU | 21.1 | 44.4 | 0.48× | 166.2 | 0.35× |

example, MoDiff INT4 uses only 66.7% of the memory required by FP32. Although the DRAM transfer increases slightly relative to the baseline due to additional cache I/O, the overall memory savings remain promising.

**Generation Quality.** We evaluate the generation quality under different precision settings. With our kernels, the FID for FP32 is 5.73, for INT8 is 5.85, and for INT4 is 14.80. It shows that low-precision activations can still preserve generation quality, aligning with the results reported in the MoDiff paper.

### 4.3. Ablation Study

We conduct ablation studies to provide a fine-grained analysis of the performance gains achieved by our kernels. We first partition the model into its individual components and evaluate the runtimes. We then investigate how different shapes affect the acceleration achieved in convolution.

**Computational Cost Breakdown.** As shown in Table 2, most of the runtime accounts for the convolutional and linear layers. Our kernels reduce the runtime of both convolutional and linear layers. However, the remaining non-quantized layers still contribute significantly to the total computation, which explains why the overall optimization does not achieve the theoretical optimum.

**Layer-Wise Analysis.** To study the fine-grained acceleration behavior, we report the runtime for convolutional layers with different shapes. As shown in Table 3, most convolutional layers in LDM have a large number of channels. With these layers accounting for the majority of the computation, the achieved acceleration is substantial. The results demonstrate that our kernels provide significant improvements in computational efficiency.

Table 3. Layer-wise speedup analysis across convolutional layers with different shapes. Here, $C_{in}$ denotes the number of input channels, $C_{out}$ denotes the number of output channels, and $K$ denotes the kernel size. Count indicates how many times each layer shape appears in the model. The speedup is compared to FP32.

| ($C_{in}$, $C_{out}$, K) | Count | FP32 | INT8 | Speedup | INT4 | Speedup |
|---|---|---|---|---|---|---|
| (1536, 768, 3) | 5 | 0.3677 | 0.1452 | 2.53× | 0.0802 | 4.59× |
| (768, 768, 3) | 23 | 0.2220 | 0.0791 | 2.81× | 0.0439 | 5.06× |
| (384, 384, 3) | 21 | 0.1508 | 0.0518 | 2.91× | 0.0332 | 4.54× |
| (192, 192, 3) | 9 | 0.1434 | 0.0802 | 1.79× | 0.0479 | 2.99× |
| (576, 192, 3) | 1 | 0.0712 | 0.0654 | 1.09× | 0.0373 | 1.91× |
| (768, 384, 1) | 4 | 0.0370 | 0.0251 | 1.47× | 0.0218 | 1.70× |
| (384, 192, 1) | 2 | 0.0353 | 0.0228 | 1.55× | 0.0213 | 1.65× |

(a) FP32 (b) INT8 (c) INT4

Figure 2. Generation Results Visualization.

### 4.4. Visualization

We provide the visualization of the generated images in Fig. 2. The results show that the low-bit quantization can still preserve the generation quality.

## 5. Conclusion

In this paper, we present a practical kernel implementation of MoDiff to bridge the gap between theoretical quantization efficiency and real hardware acceleration for diffusion models. By introducing a cache-update fusion paradigm, our method eliminates additional I/O overhead during inference and enables efficient low-bit activation quantization in practice. Extensive experiments and ablation studies show that our implementation achieves up to 1.8x runtime speedup over FP32 models and reduces memory I/O usage by up to 42.2%, demonstrating its effectiveness for accelerating diffusion model inference on hardware. These results highlight the promise of hardware-aware post-training quantization for improving the deployability of diffusion models in on-device and real-time scenarios. In future work, we will extend our evaluation to more datasets and model architectures, and compare against a broader set of quantization baselines to further validate the generality and effectiveness of our approach.

## References


[1] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 1

[2] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *arXiv preprint arXiv:2105.05233*, 2021. 2

[3] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 1

[4] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015. 1

[5] Benoit Jacob, Skirmantas Kligys, Bo Chen, and et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018. 1

[6] Amir Gholami, Sehoon Kim, Zhen Dong, and et al. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021. 1

[7] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *NeurIPS*, 2019. 1

[8] Weizhi Gao, Zhichao Hou, Junqi Yin, Feiyi Wang, Linyu Peng, and Xiaorui Liu. Modulated diffusion: Accelerating generative modeling with modulated quantization, 2025. 1, 2

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020. 2

[10] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. https://doi.org/10.48550/arXiv.2209.04747, 2022. arXiv:2209.04747. 2

[11] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. 2

[12] Dongqi Zheng. Diffusion models on the edge: Challenges, optimizations, and applications. *arXiv preprint arXiv:2504.15298*, 2025. 2

[13] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2, 3

[14] Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021. 2

[15] Eunhyeok Yang, Dongjun Choi, Sungjoo Lee, and Sungwoo Oh. Quantization networks. *arXiv preprint arXiv:1912.12655*, 2019. 2

[16] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021. 2

[17] Xiuyu Li, Meng Zhang, Shuai Zhang, Lei Li, Wei Chen, Xing Zhang, Kaisheng Lin, Xiaoguang Zhang, Jie Luo, and Xiaolin Li. Q-diffusion: Quantizing diffusion models. *arXiv preprint arXiv:2302.04304*, 2023. 2

[18] Yuzhang Wang, Jiahao Li, Zhijian Wang, Cheng Wang, Minghao Wang, Mingyang Wang, Xiaowei Wang, Dong Li, Xin Wang, and Wenxin Li. Towards accurate post-training quantization for diffusion models. *arXiv preprint arXiv:2402.04324*, 2024.

[19] Yushi Huang, Ruihao Li, Yifan Wang, Wei Wang, Shi Gu, Xiaoguang Li, Jie Luo, and Xiaolin Li. Tfmq-dm: Temporal feature-modulated quantization for diffusion models. *arXiv preprint arXiv:2405.19577*, 2024.

[20] Yuzhe Shang, Zhihang Wang, Yue Wang, Yang Wang, Linghao Wang, Pan Wang, Yajuan Wang, Tong Wang, Mingyang Wang, Xiaowei Wang, et al. Post-training quantization on diffusion models. *arXiv preprint arXiv:2311.06209*, 2023.

[21] Yixin He, Shoufa Li, Lingyu Wang, Wei Wang, Tian Wang, Mingyang Wang, Xiaowei Wang, Dong Li, Xin Wang, and Wenxin Li. Ptqd: Accurate post-training quantization for diffusion models. *arXiv preprint arXiv:2305.10687*, 2023.

[22] Xiuyu Zhao, Yuzhang Li, Minghao Wang, Mingyang Wang, Xiaowei Wang, Dong Li, Xin Wang, and Wenxin Li. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. *arXiv preprint arXiv:2406.02540*, 2024.

[23] Guangxuan Xiao, Ji Yao, Wenguang Zhang, Sheng Luo, Wei Wang, and Xiaoying Yang. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438*, 2023. 3

[24] Yuji Feng, Xin Li, Jing Wang, Wei Wang, Shi Gu, Xiaoguang Li, Jie Luo, and Xiaolin Li. Mpq: Memory-efficient patch-wise quantization for stable diffusion models. *arXiv preprint arXiv:2402.03575*, 2024. 2

[25] Dibakar Gope, Jesse Beu, and Matthew Mattina. High throughput matrix-matrix multiplication between asymmetric bit-width operands. *arXiv preprint arXiv:2008.00638*, 2020. 2

[26] NVIDIA Corporation. Cutlass: Fast linear algebra in cuda c++. https://github.com/NVIDIA/cutlass, 2023. 3

[27] Tianqi Chen, Thierry Moreau, Ziheng Jiang, and et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, 2018. 3

[28] Nicolas Vasilache, Oleksandr Zinenko, Georgios Theodoridis, and et al. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. In *arXiv preprint arXiv:1802.04730*, 2018. 3

[29] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. In *arXiv preprint arXiv:1506.03365*, 2015. 3

[30] 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 (CVPR)*, 2022. 3

[31] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems*, 30, 2017. 3