Low-bitwidth Floating-Point Quantization for Diffusion Models

by

Cheng Chen

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto

Low-bitwidth Floating-Point Quantization for Diffusion Models

Cheng Chen
Master of Applied Science
The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
University of Toronto
2024

Abstract

Diffusion models are emerging models that generate images by iteratively denoising random Gaussian noise using deep neural networks. These models typically exhibit high computational and memory demands, necessitating effective post-training quantization for high-performance inference. Recent works propose low-bitwidth (e.g., 8-bit or 4-bit) quantization for diffusion models, however 4-bit integer quantization typically results in low-quality images. We observe that on several widely used hardware platforms, there is little or no difference in compute capability between floating-point and integer arithmetic operations of the same bitwidth (e.g., 8-bit or 4-bit). Therefore, we propose an effective floating-point quantization method for diffusion models that provides better image quality compared to integer quantization methods. We employ a floating-point quantization method that was effective for other processing tasks, specifically computer vision and natural language tasks, and tailor it for diffusion models by integrating weight rounding learning during the mapping of the full-precision values to the quantized values in the quantization process. We comprehensively study integer and floating-point quantization methods in state-of-the-art diffusion models. Our floating-point quantization method not only generates higher-quality images than that of integer quantization methods, but also shows no noticeable degradation compared to full-precision models (32-bit floating-point), when both weights and activations are quantized to 8-bit floating-point values, while has minimal degradation with 4-bit weights and 8-bit activations. Additionally, we introduce a better methodology to evaluate quantization effects, highlighting shortcomings with existing output quality metrics and experimental methodologies. Finally, as an additional potential benefit, our floating-point quantization method increases model sparsity by an order of magnitude, enabling further optimization opportunities.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Andreas Moshovos, for providing me with the invaluable opportunity to study and conduct research at the University of Toronto. His unwavering guidance and support have been instrumental in making this work possible. I am also grateful to Christina Giannoula for her invaluable advice and assistance with my first paper.

I would like to extend my heartfelt thanks to my fellow research group members: Kareem Ibrahim, Qinyang Bao, Enrique Torres Sanchez, and Milos Nikolic, for their insightful suggestions and feedback during our group meetings. Their contributions have greatly enriched my research experience.

Outside of academics, basketball has been my greatest passion. I am deeply thankful to all my basketball teammates for the joy they have provided. Basketball is a much-needed balance to my research work. The time I spent at the UofT Athletic Centre will always hold a special place in my memories.

Lastly, but most importantly, I want to express my profound gratitude to my parents, Yongzheng Chen and Yuqiong Sun. Their unwavering support, both financial and emotional, has been the foundation of my academic journey in Canada. I am who I am today because of their endless love and encouragement.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 2
- 1.2 Key Results . . . 3
- 1.3 Thesis Organization . . . 4

2 Background **5**
- 2.1 Introduction to Deep Learning . . . 5
- 2.2 Layer Types . . . 5
  - 2.2.1 Linear Layer . . . 6
  - 2.2.2 Convolution Layer . . . 6
  - 2.2.3 Normalization Layer . . . 7
  - 2.2.4 ResNet Block . . . 7
  - 2.2.5 Attention Block . . . 7
- 2.3 Diffusion Models . . . 8
- 2.4 Quantization . . . 10
  - 2.4.1 Uniform Integer Quantization . . . 11
  - 2.4.2 Floating Point Quantization . . . 11
- 2.5 Related Works . . . 13
- 2.6 Summary . . . 13

3 Workload Characterization and Methods **15**
- 3.1 Compute and Memory Demands Characterization . . . 15
- 3.2 Our Floating-Point Quantization Method . . . 17
  - 3.2.1 Encoding and Bias Value Selection . . . 17
  - 3.2.2 Gradient-Based Rounding Learning for Low-Bitwidth Weights . . . 19
- 3.3 Summary . . . 21

4 Experiments and Results **22**
- 4.1 Methodology . . . 22
- 4.2 Image Generation Quality Metrics . . . 23
- 4.3 Facilitating Fair Comparisons Across Runs . . . 24
- 4.4 Unconditional Image Generation . . . 25
- 4.5 Text-to-Image Generation . . . 26
- 4.6 CLIP Score . . . 31

4.7 Sparsity 31
4.8 Summary 32

5 Conclusion **33**

Bibliography **34**

List of Tables

2.1 Encoding candidates for each floating-point format 12

3.1 Ouput Image Quality Degradation with FP4 Weight/FP8 Activation Quantization . 19

4.1 CIFAR10 Quantitative Evaluation 26
4.2 LDM(LSUNBedroom) Quantitative Evaluation 26
4.3 Stable Diffusion Quantitative Evaluation (Reference: MS-COCO) 26
4.4 Stable Diffusion Quantitative Evaluation (Reference: Full-Precision Model Generated Images) 27
4.5 Stable Diffusion Quantitative Evaluation (Reference: Full-Precision Model Generated Images), encoding candidates: E4M3, E5M2 29
4.6 SDXL Quantitative Evaluation 29

List of Figures

2.1 Deep learning and neuroscience. Adopted from [10] . . . 6
2.2 Architecture of ResNet block . . . 7
2.3 Architecture of attention block . . . 8
2.4 Stable Diffusion Architecture. . . . 9
2.5 Diffusion model forward process. Converts an image into Gaussian random noise . . 10
2.6 Diffusion model backward process. Generate an image by iteratively denoising from a Gaussian random noise. . . . 10

3.1 Breakdown of inference latency for different types of layers, when running Stable Diffusion with batch size 1 and 8 on CPU and GPU. . . . 16
3.2 Inference memory requirements . . . 17
3.3 Regularization term . . . 20
3.4 Round to nearest . . . 20
3.5 Rounding learning . . . 20

4.1 LDM(LSUNbedroom) Qualitative Evaluation: example images generated by different quantized models . . . 23
4.2 Stable Diffusion Qualitative Evaluation . . . 27
4.3 Percentage of each encoding selected by our quantization method in the weights . . . 30
4.4 SDXL Qualitative Evaluation . . . 30
4.5 Stable Diffusion CLIP Score. The dotted red line indicates the CLIP score of images generated by the full-precision model . . . 31
4.6 Percentage of weights that are zero in Stable Diffusion and LDM(LSUNbedroom) . . 31

Chapter 1

Introduction

Diffusion models [54, 53, 16] have demonstrated remarkable success in image synthesis and image generation, surpassing the previous state-of-the-art GAN-based generative models[5]. These models have proven to be highly effective in a variety of applications, including image superresolution [25], inpainting[34], restoration [26], translation, editing [36], and even in autonomous vehicles [32, 42].

Diffusion models generate new images by starting with random noise and using a noise estimation network, often the U-Net [44], to iteratively denoise the input. The process begins with Gaussian-distributed random noise, which the network gradually denoises step-by-step until a final image is produced. Typically, tens to hundreds of denoising steps are used per image.

The denoising process of diffusion models is computationally and memory-intensive, whether generating a single image or a batch. For example, the Stable Diffusion text-to-image model [43] uses a U-Net with 860 million parameters and typically requires 50 denoising steps per image. Generating one image with Stable Diffusion takes about 6 seconds on an Nvidia V100 GPU and 304 seconds on an Intel Xeon Gold 5115 CPU. This lengthy inference process makes it impractical to use Stable Diffusion in real-time applications deployed on resource-constrained edge devices.

Quantization [39] reduces the memory footprint of neural networks by changing their value representation from 32-bit floating-point (FP32) to narrower bitwidths. Quantization entails a trade-off between increased efficiency and output quality. Approaches to mitigate the impact of quantization on output quality are Quantization Aware Training (QAT) and Post Training Quantization (PTQ). QAT trains the network for a target datatype [39], requiring a training dataset and significant time; for instance, training the Latent Diffusion Model-4 (LDM4) for one epoch takes about 7 hours on an NVIDIA A100 [43]. QAT setup is also complex and may need refinement if the target datatype is too restrictive. PTQ, in contrast, starts with an already-trained network, does not need a training dataset, and is much faster, optionally using fine-tuning to reduce the impact.

Quantization poses unique challenges for diffusion models due to: 1) the noise introduced by quantization and 2) the model's use of multiple iterations. The precision loss from quantization appears as noise in computed values. For models with a single pass, like image classification, this is less concerning. However, in diffusion models, the iterative denoising process during inference can cause quantization noise to accumulate and amplify over multiple time steps.

The use of integer PTQ without fine-tuning to reduce inference costs in diffusion models has received significant attention [49, 27, 13]. These methods improve the balance between quantization

aggressiveness and output quality, however, further improvements are still needed. Using state-of-the-art integer PTQ methods, the generated image quality degrades when quantizing to 8-bit integer (INT8), and the degradation is even greater with 4-bit integer (INT4) quantization.

In contrast to prior works, we argue that quantizing diffusion models to 8-bit floating point (FP8) instead of INT8, and to 4-bit floating point (FP4) instead of INT4, can be equally advantageous in performance efficiency for two reasons. First, the memory footprint and bandwidth remain identical, as they depend on bitwidth, not bit interpretation. Second, on some platforms, there is little or no difference in compute capability between integer and floating-point operations. For example, this is true for desktop/server GPUs (2000 TFLOPS peak FP8 tensor core VS. 2000 TOPS peak INT8 tensor core on NVIDIA H100) [1] and embedded Orin NVidia GPUs [19]. Additionally, NVIDIA H100 GPU Tensor Core units have added support for FP8 format, offering twice the computational throughput of 16-bit operations [47].

However, it is not clear if floating-point quantization will provide better output quality over integer quantization. Careful consideration needs to be given, since although the bitwidth is the same, quantization to floating-point vs. to integer entails a non-trivial trade-off between precision and range: for the same bitwidth, the integer representation offers more precision for some values, while the floating-point offers a wider range.

Therefore, our *goal* in this work is to study floating-point quantization in diffusion models, targeting to improve output quality vs. integer quantization to the same bitwidth. We quantize both weights and activations using a PTQ method. To our knowledge, our work is the *first* to apply floating point quantization in diffusion models, and to successfully quantize the weights of diffusion models to FP4.

We propose a floating point quantization method for diffusion models inspired by the method developed by Kuzmin et al. [22] which quantizes computer vision and natural language processing models. This method has not been applied to diffusion models, thus important improvements are needed to be effective for low-bitwidth quantized diffusion models. Specifically, we introduce a rounding learning technique (inspired by Nagel et al. [38]) in floating-point diffusion model quantization: we leverage the gradient descent algorithm to learn how to map full-precision values (32-bit) to quantized values (8-bit or 4-bit) more effectively, rather than simply mapping full-precision values to their corresponding nearest values in the quantized range. This is the *first* work that applies this key technique to floating-point quantization of deep neural networks.

We evaluate our floating-point quantization method versus prior integer quantization methods on two tasks: unconditional generation and text-to-image generation. In unconditional generation, we use the Latent Diffusion Model(LDM) pre-trained on the LSUN-Bedrooms256x256 dataset [62] and the DDIM [53] pre-trained on the 32×32 CIFAR-10 dataset [20]. This model is trained to generate a specific class of images. In text-to-image generation, that generates different classes of images based on a textual description of what the image should depict, we use Stable Diffusion [43] and Stable Diffusion XL [41] pre-trained on the 512x512 LAION-5B datasets [48]. For our target quantization bitwidth, we quantize weights to both FP8 and FP4 and activations to FP8.

1.1 Contributions

Overall, this thesis makes the following contributions:

- We propose a floating-point quantization method that uses (i) a greedy-search approach to assign per tensor floating-point formats, and (ii) a gradient-based rounding learning approach to improve output quality. Our work is the first to enable high output quality, when quantizing weights and activations of diffusion models to FP4 and FP8.
- We conduct the first experimental analysis comparing floating-point and integer quantization on diffusion models, and demonstrate that our method significantly improves output quality over prior approaches.
- We introduce a better methodology for measuring output quality by choosing more appropriate reference images for fair comparisons.

1.2 Key Results

We highlight the following experimental findings:

- Our floating-point quantization method for diffusion models outperforms integer quantization at the same bitwidth, providing better quality. One metric used to evaluate the quality of generated images is the Fréchet Inception Distance (FID) [15], which will be discussed in Chapter 4. For the Latent Diffusion Model (LDM) pre-trained on the LSUN-Bedrooms256x256 dataset, the FP8 quantized model outperforms the INT8 quantized model by a factor of 1.12 in terms of FID, while the FP4 quantized model performs 1.14× better than the INT4 quantized model. In the case of Stable Diffusion, the FP8 quantized model is 2× better than the INT8 quantized model, and the FP4 quantized model is 1.06× better than the INT4 quantized model. Since the memory costs and compute capabilities of integer and floating-point representations of the same bitwidth are identical on many platforms, our method offers an attractive trade-off between efficiency and output quality.
- In text-to-image generation (Stable Diffusion), models with weights quantized to FP4 and activations to FP8 generate higher-quality images compared to models with both weights and activations quantized to INT8. Quantitatively, the model with weights quantized to FP4 and activations quantized to FP8 achieves an FID score of 5.53, which is identical to the score obtained by the model with both weights and activations quantized to INT8.
- We find that popular metrics used to measure the output quality of diffusion models do not always accurately reflect human perception of image quality and should be used with caution. Visual inspection of integer-quantized model outputs reveals discrepancies between metric-reported quality and actual perceived quality.
- Our quantization method increases the sparsity in weights of diffusion models by an order of magnitude. For Stable Diffusion, our methods achieve a 31.6× increase in sparsity when quantizing weights to FP8, and a 617× increase when quantizing to FP4, compared to the full-precision model. For the Latent Diffusion Model (LDM) trained on LSUN-Bedrooms, the increase in sparsity is 20.1× with FP8 quantization and 428.5× with FP4 quantization, relative to the full-precision model. Sparsity can be exploited to further improve inference latency, since it provides additional optimization opportunities.

1.3 Thesis Organization

- In Chapter 2, we begin with an introduction to deep learning and explore the various types of layers used in neural networks. Additionally, we delve into the workings of diffusion models and present the formulation of both integer and floating-point quantization.
- In Chapter 3, we analyze representative diffusion model workloads and introduce our low-bitwidth floating-point quantization method to reduce the computational and memory demands during inference.
- In Chapter 4, we evaluate our method across different tasks and models pre-trained on various datasets. Additionally, we propose an improved methodology for evaluating quantized diffusion models.
- In Chapter 5, we offer concluding remarks and suggest directions for future research.

Chapter 2

Background

2.1 Introduction to Deep Learning

Deep learning, also known as neural networks, is a subset of machine learning algorithms loosely inspired by an understanding of the structure and function of the human brain. These models are designed to simulate the decision-making process of the brain by using layers of interconnected neurons. As shown in Figure 2.1, a typical deep learning model consists of multiple layers: an input layer, several hidden layers, and an output layer. Each layer contains neurons, which are connected to those in the next layer through weighted connections, similar to the synapses in the brain.

It is believed that neurons in the human brain transmit signals to each other through these synapses to enable complex thought processes and decision-making. Similarly, in a deep learning model, each neuron receives input from the neurons in the previous layer, processes this input using the parameters of the layer and activation function, and then passes the result to the neurons in the next layer. The outputs of each layer are referred to as *activations* and the parameters of each layer are referred to as *weights*. This process allows the model to learn and recognize patterns in data. The model can utilize the patterns to perform tasks such as image classification and text generation.

The architecture of a deep learning model can vary, with different types of layers (e.g., Convolutional layers, Linear layers) designed to handle specific types of data and tasks. Before making predictions with the model, the values of the weights need to be tuned based on the tasks to maximize the prediction accuracy. This process is referred to as *training*. After training, the weights are fixed and we can perform *inference* with the model.

The power of deep learning lies in its ability to automatically extract features from raw data and build hierarchical representations, which has led to significant advancements in various fields. In this thesis, we focus on diffusion models, which are image generative models.

2.2 Layer Types

In this section, we explore three fundamental components that appear in many advanced machine learning models: the Linear Layer, the Convolution Layer, and the Normalization Layer. Each of these layers has a distinct and crucial role in processing and transforming data as it moves through a neural network. By looking into their mathematical foundations and functional purposes in detail,

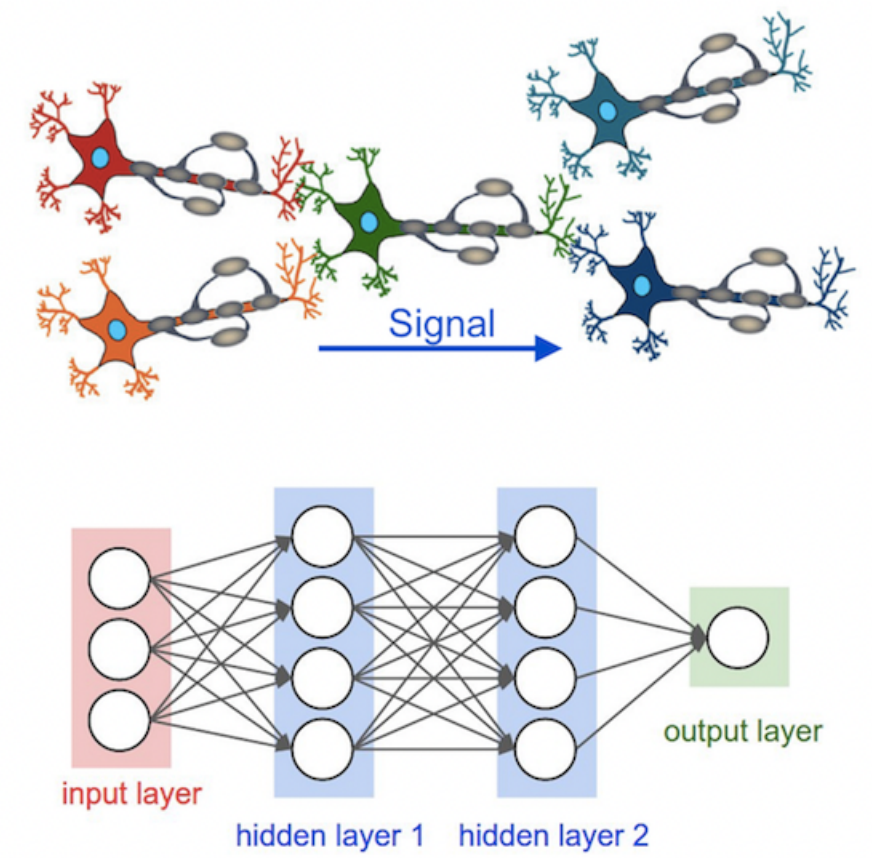


Figure 2.1: Deep learning and neuroscience. Adopted from [10]

we gain valuable insights into how deep learning models extract features and make predictions across diverse applications, from computer vision to natural language processing. In addition, we discuss ResNet [12] and Attention [56] blocks, which are composed of the aforementioned layers and serve as the fundamental building blocks of diffusion models. While there are various types of layers in deep learning, this thesis focuses specifically on diffusion models and therefore includes only the layers relevant to these models.

2.2.1 Linear Layer

The linear layer performs a linear projection of an input matrix using a set of learnable parameters (weights). Let us denote X as input, W as weights, b as the bias. The output of the layer Y can be obtained by:

$$Y = W \times X + b \tag{2.1}$$

2.2.2 Convolution Layer

The convolution layer is widely used in deep learning models for computer vision because it is particularly effective at extracting locality features. Convolution layers use a kernel (also known as filter) to sweep through the input matrix. Mathematically, the convolution layer can be expressed

as:

$$S(i,j) = (X * K)(i,j) = \Sigma_m \Sigma_n X(i+m, j+n) \cdot K(m,n) \tag{2.2}$$

where X is the input, K is the kernel and S is the output. $*$ stands for the convolution operation. i, j and m, n are the coordinates in the output and kernel respectively.

2.2.3 Normalization Layer

Normalization layers such as GroupNorm [59] and LayerNorm [2] are designed to normalize the activations to help with training stability and convergence. The normalization can be expressed as follows:

$$Y = \frac{X - E[X]}{\sqrt{var[X] + \epsilon}} \cdot \gamma + \beta \tag{2.3}$$

where X is the input and Y is the output. $E[X]$ denotes the mean of X and $var[X]$ denotes the variance of X. γ and β are trainable parameters and ϵ is to ensure that there is no division by zero.

2.2.4 ResNet Block

The ResNet architecture, first proposed by He et al.[12], plays a crucial role in diffusion models. A typical ResNet block, as shown in Figure 2.2, consists of two GroupNorm layers and two Convolution layers. One of the most innovative aspects of ResNet is the use of skip connections, where a copy of the original input is added or concatenated with the activations to produce the final output. This technique addresses the problem of vanishing gradients, which can occur in deep networks trained using backpropagation [45], by allowing gradients to propagate directly from the last layer to the first. As a result, it leads to faster convergence during training.

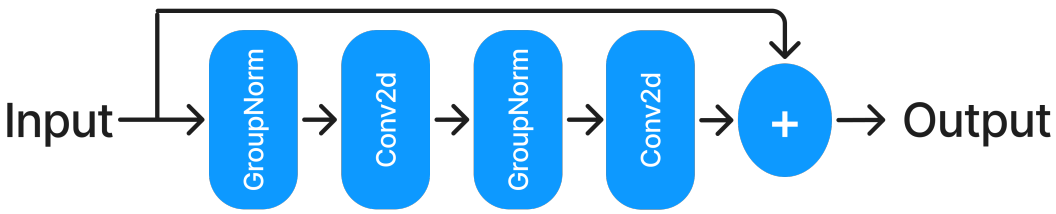


Figure 2.2: Architecture of ResNet block

2.2.5 Attention Block

The attention mechanism, introduced by Vaswani et al. [56], has become a fundamental component in many computer vision and natural language processing models. It consists of three Linear layers, a GroupNorm layer, and a Softmax layer, as illustrated in Figure 2.3. The three Linear layers generate intermediate outputs known as Query (q), Key (k), and Value (v), respectively. The

Query and Key are multiplied, and the resulting product is passed through a Softmax function to produce the attention scores. These scores then multiply with the Value, yielding the final output. The attention mechanism excels at capturing relationships between different elements in the input. In the context of diffusion models, attention is employed to learn the relationship between input prompts and the generated images, as well as the relationships between different parts of the image, ensuring that the resulting images are high-quality and align with the descriptions provided in the prompts.

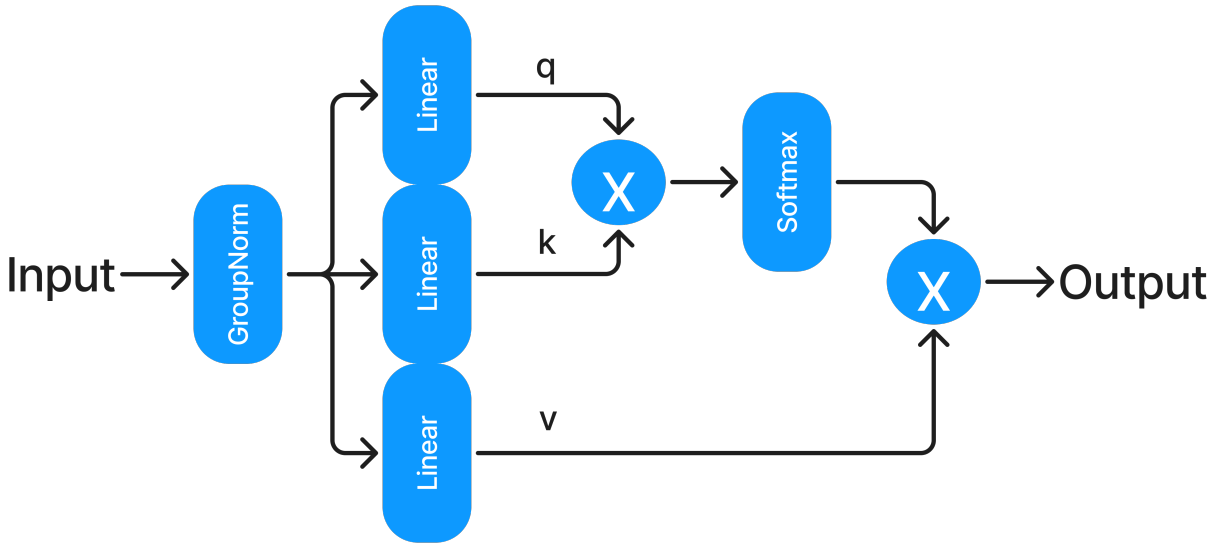


Figure 2.3: Architecture of attention block

2.3 Diffusion Models

Diffusion models can perform a multitude of tasks. In this work, we focus on two widely used applications: unconditional generation and text-to-image generation. Figure 2.4 illustrates a high-level view of the Stable Diffusion text-to-image generation model [43]. Stable Diffusion accepts as input a Gaussian-distributed random noise and a prompt that describes the image to be generated. The prompt is then encoded by a "text encoder" before getting passed to the U-Net denoising subnetwork. The U-Net then iteratively removes the noise from the noise input over multiple steps (typically 50). The final generated image is produced by the "Autoencoder/Decoder" subnetwork which is invoked once at the end. The "Autoencoder/Decoder" converts the latent space output produced by the U-Net to the pixel space as the output image.

The U-Net itself comprises several ResNet [12] and Attention [56] blocks. Figure 2.4 shows how the blocks are connected along with a simplified yet representative view of the internals of such a block. A typical block uses a combination of attention, convolutional, linear, group and/or layer normalization layers. A distinctive characteristic of U-Net compared to other deep neural

network architectures is the presence of block-to-block skip connections. A skip connection saves the activations from an earlier layer so that it can later concatenate them with the activations passed as input to another later layer. This technique has been proven to be helpful to the model's predictive or generative performance [12]. Specifically, the output of the first block of the U-Net is also passed as input to the last block as shown by the arrows in Figure 2.4. The output of the second U-Net block is passed as input to the last block, and so on. This is atypical of other conventional neural networks such as transformers where activations are typically consumed immediately after each block.

Supervised training of diffusion models, as with any other neural network, requires a dataset with ground truth. To learn how to generate images from random noise, this dataset is generated as follows: the *forward process* starts with real data x_0 and incrementally introduces Gaussian noise to x_0. The Gaussian noise is added T times, and if T is sufficiently large, the forward process ends up with random Gaussian noise. The forward process is illustrated in Figure 2.5. Since the exact noise added at each step is known, the noise will be used as the ground truth during the training of the diffusion model. The forward process is defined as follows [16]:

$$q(x_{1:T}|x_0) = \Pi_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.4}$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, \beta_t I) \tag{2.5}$$

where β_t and α_t are hyperparameters regulating the intensity of Gaussian noise added at each step, β_t = 1-α_t. x_t refers to the image at the current step, and $q(x_t|x_{t-1})$ is the sampled Gaussian noise to be added at step t.

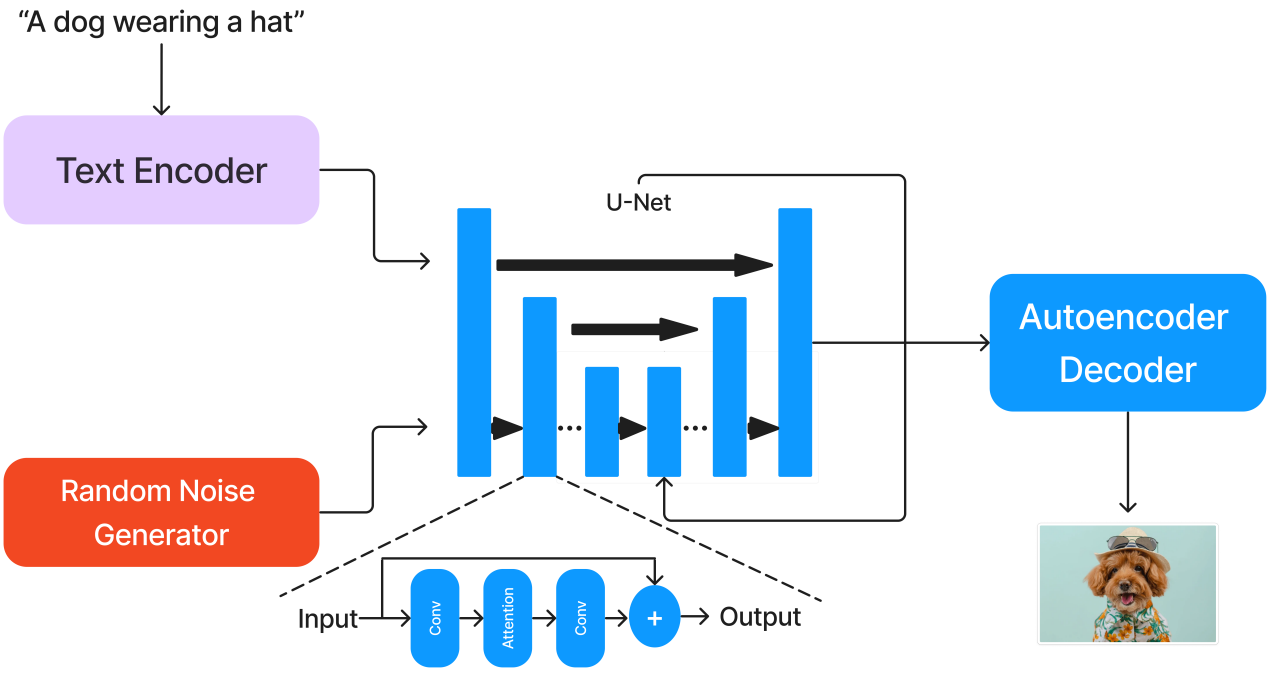


Figure 2.4: Stable Diffusion Architecture.

The *backward process* aims to eliminate the noise from the noisy data to generate high-quality images. As shown in Figure 2.6, the backward process starts with random Gaussian noise x_T. The diffusion model eliminates the predicted noise from the noisy data at each step and repeats the process for T steps. As the distribution $q(x_{t-1}|x_t)$ is intractable, diffusion models sample from a learned Gaussian distribution $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$, where the mean is reparameterized by a noise prediction network $\epsilon_\theta(x_t, t)$:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) \tag{2.6}$$

where $\bar{\alpha}_t = \Pi_{i=1}^{t}\alpha_i$.

A detailed explanation of diffusion models and of their mathematical foundation is provided by Luo [35].

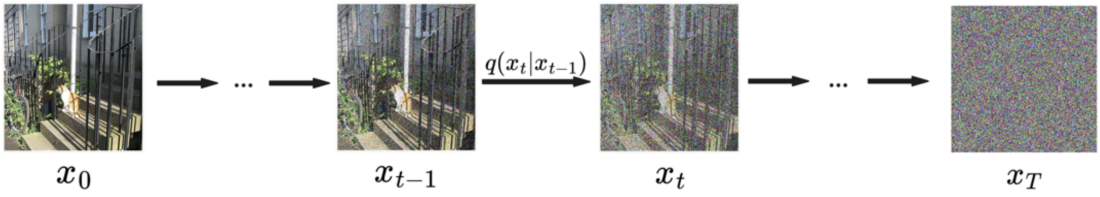


Figure 2.5: Diffusion model forward process. Converts an image into Gaussian random noise

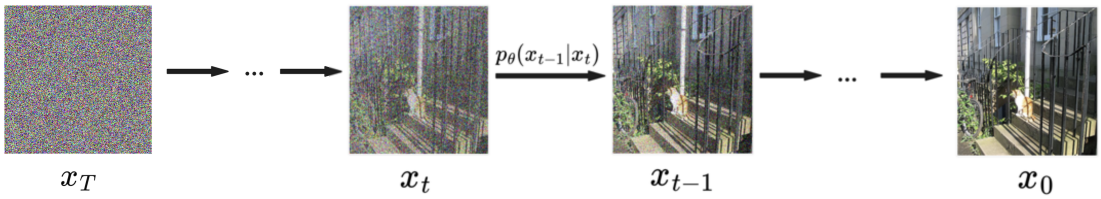


Figure 2.6: Diffusion model backward process. Generate an image by iteratively denoising from a Gaussian random noise.

2.4 Quantization

The goal of quantization is to reduce memory and/or compute costs by representing values with more efficient datatypes while, ideally, maintaining the output quality of the full-precision model. More efficient datatypes typically are those of lower precision, e.g., quantizing a model that originally used floating-point 32 bit to integer 8 bit.

Typically, models are trained to work with 32 bit floating-point. We will refer to these models as the *baseline* or *full-precision*. The goal of any quantization method is to use a more efficient datatype with as little as possible effect on the model's output quality. We will use the term "*task performance*" to refer to any metric that quantifies the quality of the output.

In post-training quantization (PTQ), the full-precision model is converted directly to a lower bitwidth model without retraining. Some PTQ methods use fine-tuning [8], involving additional training steps. The dataset for fine-tuning may differ from the training dataset. In this work, we focus on PTQ methods that do not use fine-tuning which would be a significant overhead in time

and effort and would require access to an appropriate dataset.

PTQ methods typically require only a lightweight calibration dataset to determine the quantization range, minimizing the error between the quantized and full-precision data. The next section will discuss methods for determining the quantization range. Generally, lower-bitwidth quantization results in greater quality degradation compared to the full-precision model.

Quantization requires "rounding" some values of the original full-precision model to another value. The simplest method is rounding to the nearest representable value in the quantized model, but this is not always optimal. Recent studies propose alternative rounding strategies [38] and block-wise reconstruction [29] to mitigate quantization effects. However, these methods focus only on integer quantization, leaving low-bitwidth floating-point quantization relatively underexplored. Our goal is to investigate whether and to what extent narrow floating-point quantization can reduce the negative impact on task performance for diffusion models.

Before we present our quantization method we first overview baseline integer and floating-point quantization approaches.

2.4.1 Uniform Integer Quantization

In uniform integer quantization, given a floating-point vector X and the target integer bitwidth b, the quantization process is defined as:

$$X^{int} = s \cdot (clamp(\left\lfloor \frac{X}{s} \right\rceil + z; 0, 2^b - 1) - z) \tag{2.7}$$

where $\lfloor \cdot \rceil$ is the round-to-nearest function and $clamp(\cdot; min, max)$ is the element-wise clipping operation to ensure that the quantized values are within the range that can be represented by the target bitwidth. The scaling factor s is defined as $\frac{max(X)-min(X)}{2^b-1}$, and, finally, the zero point z is $-\left\lfloor \frac{min(X)}{s} \right\rceil$.

2.4.2 Floating Point Quantization

A standard floating-point number is represented as follows:

$$f = (-1)^s 2^{p-b}(1 + \frac{d_1}{2} + \frac{d_2}{2^2} + \cdots + \frac{d_m}{2^m}) \tag{2.8}$$

where $s \in \{0, 1\}$ is the sign bit, $d_i \in \{0, 1\}$ is the m-bit mantissa, p is an integer and $0 \leq p \leq 2^e$, where e denotes the number of exponent bits, and b is an integer exponent bias, which is typically defined to be 2^{e-1}.

Floating-point numbers can be seen as a uniform m-bit grid between two consecutive integer powers: 2^a, 2^{a+1}. The distance between two adjacent grid points is 2^{a-m}. Therefore, floating-point quantization is performed in similar fashion as described in (2.7).

The original values X are first clipped to the range of the target floating-point format:

$$X' = clamp(X; -c, c) \tag{2.9}$$

$$c = (2 - 2^{-m})2^{2^e - b - 1} \tag{2.10}$$

Table 2.1: Encoding candidates for each floating-point format

FP8	FP4
E2M5, E3M4, E4M3, E5M2	E1M2, E2M1

where c is the absolute maximum value that the target floating-point format can represent.

Changing the value of the exponent bias b can control the range of the target floating-point number format. Then, quantization is performed on the clipped values:

$$X_i^{fp} = clamp(s_i \left\lfloor \frac{X_i'}{s_i} \right\rceil; -c, c) \tag{2.11}$$

where X_i' denotes the i-th element of X', and X_i^{fp} denotes quantized X_i'. s_i is the scale for the i-th element of X' and is equal to:

$$s_i = \begin{cases} 2^{\lfloor \log_2 |X_i'| + b \rfloor - b - m} & \lfloor \log_2 |X_i'| + b \rfloor > 1 \\ 2^{1-b-m} & otherwise. \end{cases} \tag{2.12}$$

Kuzmin et al. present a more comprehensive overview [22].

Unlike integer quantization, floating-point quantization allows for different encodings within the same bitwidth. In FP8, one bit is allocated for the sign, and the remaining 7 bits are divided between the exponent and mantissa bits. More bits can be allocated to the exponent vs. the mantissa if a wider range over precision is needed. In our study, as done by Kuzmin et al. [22], we consider four encoding candidates for FP8: E2M5 (2-bit exponent and 5-bit mantissa), E3M4 (3-bit exponent and 4-bit mantissa), E4M3 (4-bit exponent and 3-bit mantissa), and E5M2 (5-bit exponent and 2-bit mantissa). For FP4, we consider two encodings: E1M2 (1-bit exponent and 2-bit mantissa) and E2M1 (2-bit exponent and 1-bit mantissa).

Regarding energy efficiency between FP and INT, Qualcomm estimates that an 8-bit FP unit consumes 50% more energy than an INT8 unit [3]. However, this estimate only applies to isolated INT and FP units, excluding data access and movement costs. In GPUs, the energy overhead of FP operations is minimal due to: 1) the integration of INT and FP within the same unit, 2) the dominance of data movement, as register files are typically in the order of 100K, and 3) the crossbar between the datapath and shared memory, which further exacerbates data movement costs. The energy difference is expected to be even smaller for 4-bit operations due to the reduced exponent range and mantissa length.

Currently, the Nvidia H100 GPU supports only E4M3 and E5M2 FP8 formats [1]. Xia et al. [60] implemented a CUDA kernel that accelerates any FP8 model by sign-extending FP8 to FP16 during computation, while transferring data between global and shared memory in FP8 when the GPU does not natively support the FP8 format. Table 2.1 summarizes the encoding candidates considered for different floating-point formats in this thesis. Moreover, although the bias value is commonly defined as 2^{e-1}, it can be modified. In this work, we use per tensor biases — the bias value needs to be stored as metadata and using per tensor biases makes this metadata overhead negligible.

2.5 Related Works

Quantization is a widely used optimization technique for deep neural networks that reduces the size and computational complexity of the model by using lower-bitwidth weights/activations. Deep Compression [11] used pruning to reduce the number of connections in deep neural networks by 9x to 13x and then used quantization to reduce the number of bits that represent each connection from 32 to 5. Combining pruning, quantization, and Huffman encoding, Deep Compression was able to reduce the storage required by AlexNet [21] and VGG-16 [51] 35× and 49× respectively without loss of accuracy. Jacob et al. [18] proposed a quantization technique that allows inference to be carried out using integer-only arithmetic and explored the latency-vs-accuracy tradeoff for quantized models on CPUs.

Recently, with the growing size of deep neural networks, particularly Large Language Models (LLMs), there has been a surge of research focusing on integer quantization for LLMs [50, 61, 9, 30, 57]. In contrast, floating-point quantization remains relatively underexplored. Fan et al. [7] quantized convolutional neural networks(CNN) to 8-bit Block Floating-Point(BFP) on FPGA, which supports efficient arithmetic with the fixed-point computational complexity and floating-point range. Kuzmin et al. [22] explored the effects of FP8 quantization on various computer vision and natural language processing models and provided an efficient implementation for FP8 simulation. Liu et al. [33] was the first work that quantized both the weights and activations of LLMs to FP4.

In this thesis, we focus on applying floating-point quantization to diffusion models. Diffusion models generally have fewer parameters compared to LLMs. However, a single inference in diffusion models requires multiple timesteps, leading to high latency and memory footprint. A few recent works [27, 49, 13, 28, 52] have studied quantization methods in diffusion models, and their effects on the output quality of diffusion models. PTQ4DM [49] was the first work to apply PTQ to quantize diffusion models trained on low-resolution datasets with INT8. Q-diffusion [27] expanded on PTQ4DM and successfully quantized diffusion models trained on high-resolution datasets into INT8 and INT4 data values. PTQD [13] introduced quantization noise correction methods to further improve the tradeoff between efficiency improvement and output quality degradation. Q-DM [28] quantized diffusion models to lower bitwidth such as INT2 or INT3. However, Q-DM was only evaluated on low-resolution datasets. TDQ [52] quantized both the weights and activations of diffusion models to INT4 by dynamically adjusting the quantization interval based on time step information. We quantitatively compare our quantization method over Q-diffusion, which is a stronger state-of-the-art baseline in this context, since our goal in this work is to compare the impact of the same-bitwidth floating-point vs integer quantization on diffusion models. More advanced techniques such as [28, 52, 13] complement our method to further improve the image generation quality of quantized models.

2.6 Summary

Deep learning, inspired by the structure of the human brain, uses neural networks with layers of interconnected neurons to learn and recognize patterns in data. These models consist of input, hidden, and output layers, with neurons connected through weighted connections. Layers like linear, convolutional, and normalization layers play specific roles in processing input data. Diffusion models,

a type of deep learning model, are used for tasks such as image generation by iteratively denoising random noise. One of the most commonly used techniques to reduce the costs of deep neural networks is quantization. Quantization reduces memory and computational costs by converting model weights and/or activations to lower precision. Quantization entails a tradeoff between cost reduction and output quality. Ideally, a quantization method would maintain output quality while using more efficient datatypes.

Prior work on quantization of diffusion models targeted quantization to integer values, with significant reduction in output quality for shorter bitwidths (e.g., 8-bit or 4-bit). Our goal is to explore quantization to floating-point formats instead with the hope that it would capture the memory benefits of quantization to integer values while not sacrificing output quality as much.

Chapter 3

Workload Characterization and Methods

In this chapter, we first characterize Stable Diffusion during inference. We choose to characterize Stable Diffusion as text-to-image generation is a more challenging task compared to unconditional generation. Our analysis considers inference latency and the peak VRAM usage for GPU inference and motivates our work on reducing these costs via quantization. After that, we present our method for floating-point quantization of diffusion models.

3.1 Compute and Memory Demands Characterization

As mentioned, the U-Net is expected to handle most of the work in Stable Diffusion, as it runs multiple times during a single inference. We ran Stable Diffusion with a batch size of 1 and 50 denoising steps on an NVIDIA V100 (32GB VRAM, PyTorch version 1.12.1). We measured the runtime by running 12 times in total and averaging the last 10 runs. The U-Net accounts for 6.1 out of the total 6.6 seconds on average, while the text encoder and the autoencoder decoder together account for only 0.5 seconds.

Figure 3.1 shows a breakdown of inference latency across the various layers in the U-Net. Latency is measured for one denoising step using the U-Net of Stable Diffusion and for batch sizes of 1 and 8 images in order to show the differences between smaller batch sizes and larger batch sizes. The figure shows breakdowns on two types of machines: a CPU-based (Intel(R) Xeon(R) Gold 5115 @ 2.40GHz on CentOS-7 with 98GB RAM) and GPU-based (NVIDIA Tesla V100 32GB VRAM on Red Hat Enterprise Linux 8.6). In either case we use PyTorch version 1.12.1. The breakdowns are normalized to 1.0 to highlight the relative contribution of each layer type, with total inference latency in seconds shown above each bar.

Most of the time is spent on the Conv2d and linear layers (including layers inside the attention units). The normalization and SiLU layers account for only about 25% of the overall latency on the GPU, and a negligible portion on the CPU. The relative contribution across layer types changes little with the batch size on the CPU, whereas on the GPU, increasing the batch size to 8 increases the relative contributions of the linear layers. This is because increasing the batch size increases the memory traffic. Linear layers, which process larger arrays, are affected more by the increased batch

size than conv layers, which process smaller arrays because conv layers use a kernel to sweep through the data. Since CPUs have lower compute capability than GPUs, therefore they are compute-bound in most deep learning workloads, they show no significant change in relative latency between linear and conv layers with increased batch size. On the other hand, GPUs have higher compute capability than CPUs by parallelizing the workload, making GPUs more sensitive to increases in memory traffic compared to CPUs. Additionally, inference on the GPU is 31× and 72× faster than on the CPU for batch sizes of 1 and 8, respectively.

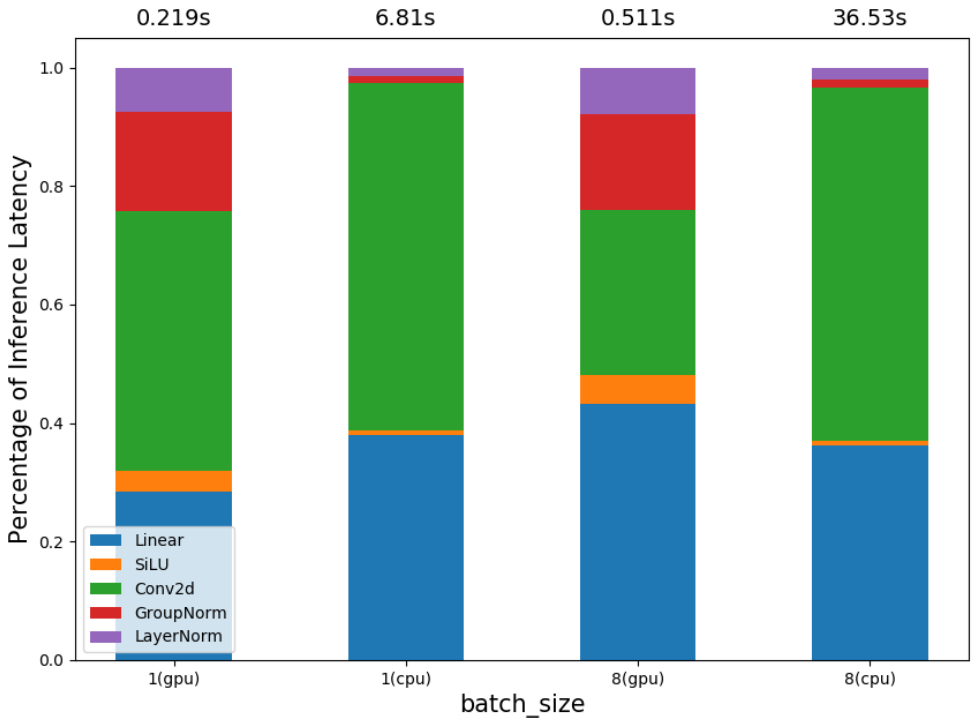


Figure 3.1: Breakdown of inference latency for different types of layers, when running Stable Diffusion with batch size 1 and 8 on CPU and GPU.

Figure 3.2 reports the peak inference VRAM usage. The measurements were collected using Nvidia's Nsight Systems on an Nvidia A100 80GB VRAM. With a batch size of 16, the VRAM usage reaches as high as 54.9GB during inference out of the 80GB available on the specific GPU model. However, this is far beyond what is available on consumer level and embedded offerings. Even if a single image is generated (batch size of 1), the memory consumption stands at 8.37GB.

Closer inspection reveals that most of the memory consumed is largely due to the size of the data generated in the attention layers. In an attention layer, there are three linear layers that output the key, query, and values. The attention tensor is created by multiplying the key and query tensors. For example, the shape of the key and query tensor is (256,4096,40) when the batch size is 16, which results in a (256,4096,4096) tensor after multiplication. This tensor would require at least 17GB of memory if the precision is FP32. This VRAM requirement could be reduced by 4× and 8× by quantizing to FP8 and FP4, respectively.

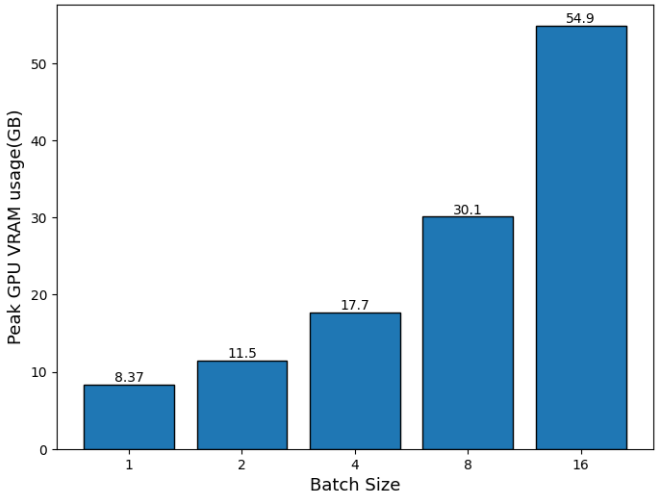


Figure 3.2: Inference memory requirements

3.2 Our Floating-Point Quantization Method

As mentioned previously, the encodings and bias values can be different for each tensor in the same floating-point format. We start by discussing the method we use to choose the best per-tensor floating-point format (i.e., mantissa and exponent bit counts) and bias values. Besides selecting the floating-point format and since the quality of generated images experienced significant degradation when quantized to FP4, we also apply rounding learning as Section 3.2.2 explains.

3.2.1 Encoding and Bias Value Selection

Algorithm 1 Finding the bias value and encoding

input : *encoding_candidates*, *bias_candidates*
output: *encoding*,*bias*
$prev\_mse = 0$
for $i = 0$ **to** $len(encoding\_candidates)$ **do**
  for $j = 0$ **to** $len(bias\_candidates)$ **do**
    $curr\_encoding = encoding\_candidates[i]$
    $curr\_bias = bias\_candidates[j]$
    $curr\_mse = get\_mse(curr\_encoding, curr\_bias)$
    if $prev\_mse > curr\_mse$ **then**
      $encoding = curr\_encoding$
      $bias = curr\_bias$
      $prev\_mse = curr\_mse$
    else
      do nothing
    end
  end
end

Before performing quantization, we need to select the encoding and the bias value for the target floating-point format. In floating-point representation, there are trade-offs between the number of bits allocated for the mantissa and the exponent. Increasing the mantissa bits enhances the precision of the values, while increasing the exponent bits expands the range of representable values. To minimize the error introduced by quantization, it is crucial to carefully choose the encoding and bias value.

Kuzmin et al. [22] found that using different encodings and bias values for various quantized weight or activation tensors helps maintain task performance in image classification and natural language processing tasks. However, this method has not been tested on diffusion models. In this work, we adopt this method to investigate its effectiveness in quantizing diffusion models.

Algorithm 1 shows the search algorithm used to find the number of bits for the mantissa and the exponent, and also the bias per tensor. We consider both weight and activations. As we explain in detail below, since each tensor can have a different encoding including a tailored bias, we use a greedy approach to trim the search space. Specifically, the searching algorithm starts from the weight and activation tensors of the first layer in the model, selects the best possible choices for them, and fixes them before proceeding to the next layer. Quantization must begin with the first layer, as the output of each layer is dependent on the previous layer's quantization. This sequential process ensures that the quantized output from one layer serves as the input for the next, in order to maintain consistency throughout the model.

Since the bias is a continuous real value, testing all possible values is impractical. Instead, we generate a set of evenly spaced values between the minimum and maximum of the data being quantized and calculate the bias for each based on (2.10). We then use grid search to find the bias and encoding that minimize the mean squared error (MSE) between the quantized and full-precision data. For activation quantization, we use an *initialization dataset*, a small sample from the full-precision model's output gathered uniformly across all timesteps (iterations of the U-Net), to obtain the full-precision data needed for calculating MSE. Empirically, 111 bias values in the grid search offer the best trade-off between search time and task performance [22]. Therefore, we use 111 bias value candidates for all our experiments.

As explained in Section 2.4.2, there are 4 encoding candidates for FP8 and 2 for FP4. Together with 111 bias values, there are 444(4×111) and 222(2×111) combinations for each weight tensor or activation tensor for FP8 and FP4 respectively, making it possible to search through each combination. For each combination of encoding and bias, we calculate the combination that achieves the lowest MSE between the quantized tensor and the full-precision tensor. Currently, hardware that supports the FP8 format, such as the Nvidia H100 [47], only supports the E4M3 and E5M2 encodings. In Section 4.5, we will evaluate our quantization method with this restricted set of supported FP8 encodings as well.

As Section 4.1 reports, this quantization method maintains task performance similar to the full-precision model when using FP8. However, as Table 3.1 shows, it fails to generate meaningful images with FP4 weights. In this experiment, weights are quantized to FP4 and activations to FP8. The table reports the *Fréchet Inception Distance* (FID), a metric for generated image quality (discussed in Section 4.1), where lower FID indicates better quality. The FID for FP4/FP8 models is 11.6× and 97.7× worse than the full-precision model for text-to-image and unconditional generation, respectively, indicating significant quality degradation. This result motivates the next step in

Table 3.1: Ouput Image Quality Degradation with FP4 Weight/FP8 Activation Quantization

	Stable Diffusion	LDM(LSUNbedroom)
Bitwidth (W/A)[1]	FID↓	FID↓
Full Precision	22.71	2.95
FP4/FP8	262.8	288.2

[1]refers to the bitwidth quantization setting for weights and activations. For example, FP4/FP8 refers to when weights are quantized to FP4 and activations quantized to FP8

our approach: applying a gradient-based rounding learning method to minimize quality loss when quantizing weights to FP4.

3.2.2 Gradient-Based Rounding Learning for Low-Bitwidth Weights

In the context of low-bitwidth *integer* quantization, Nagel et al. [38] have shown that replacing the rounding-to-nearest operation with adaptive rounding can significantly improve task performance (see (2.7)) Since floating-point quantization also uses a rounding-to-nearest operation (see (2.11)). we adopt the learned rounding method previously proposed in the context of integer quantization to the floating-point domain and study whether it proves beneficial also in this context. The rest of this section describes how this approach *learns* what rounding to use for each value in a tensor.

Quantization can be seen as a lossy compression method. To mitigate the impact of quantization on task performance, one approach is to reduce the quantization error. Let's denote the quantized weights as W^q and the original weights as W and the output activations from the previous layer A.

The quantization error can be defined as follows:

$$\Delta Y = L(Y^q, Y) = L(W^q A, WA) \tag{3.1}$$

where Y^q represents the output of a layer in the quantized model, Y represents the output of the same layer in the full-precision model, and L represents the loss function. In the above formula, we are considering a linear layer without bias as an example for ease of understanding. This method can be applied to other layers such as Convolutional and with having a bias. Following the approach of prior work [4, 22], we choose our objective as minimizing the mean square error between the quantized layer output versus original layer output. The loss function L is defined as follows:

$$L(Y^q, Y) = L(W^q A, WA) = mean((W^q A - WA)^2) \tag{3.2}$$

To enable learning the rounding policy to be used for quantizing the weights, we modify (2.11) to:

$$W_i^q(\alpha_i) = clamp(s_i \cdot (\lfloor \frac{W_i^q}{s_i} \rfloor + \sigma(\alpha_i)); -c, c) \tag{3.3}$$

replacing the rounding-to-nearest operation with the floor operation $\lfloor \cdot \rfloor$ and adding the sigmoid function $\sigma(\alpha) \in [0, 1]$, where α is the parameter we are optimizing using gradient descent. The sigmoid function is chosen because its output is in the range [0,1] which simulates between rounding down and rounding up.

The optimization objective then becomes:

$$\underset{\alpha}{\operatorname{argmin}}\, mean((W^q(\alpha)A - WA)^2) + \lambda(\alpha) \tag{3.4}$$

$$\lambda(\alpha) = 1 - (|\sigma(\alpha) - 0.5| * 2)^{20} \tag{3.5}$$

where $\lambda(\alpha)$ is the regularization term visualized in Figure 3.3. This regularization term is minimized when $\sigma(\alpha)$ approaches 0 or 1, and therefore encourages $\sigma(\alpha)$ to converge towards 0 or 1 during rounding learning. During inference time, if $\sigma(\alpha)$ is less than 0.5, we set $\sigma(\alpha)$ to 0 indicating rounding down, and to 1 if $\sigma(\alpha)$ is greater than or equal to 0.5 indicating rounding up.

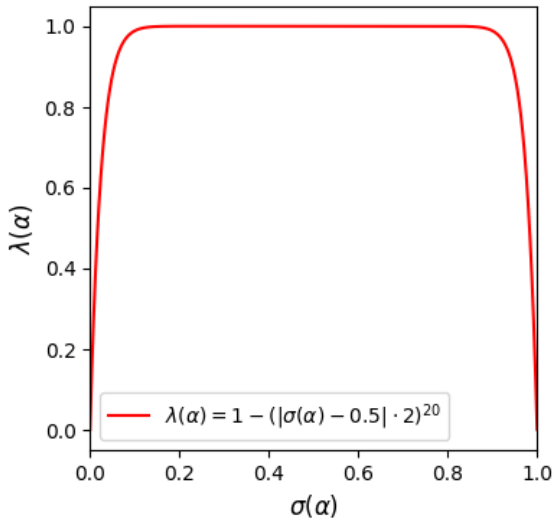


Figure 3.3: Regularization term

To learn the rounding we use a *calibration dataset* which we generate as follows: we obtain full-precision data used to calculate the loss during rounding learning, we save several samples from the output of each full-precision model timestep and then randomly select a few samples from the collected data as the calibration dataset for each iteration of rounding learning.

The round-to-nearest and rounding learning operations are visualized in Figure 3.4 and Figure 3.5 respectively. In round-to-nearest, only the closest value in the quantized range can be selected as the corresponding value after quantization. On the other hand, the rounding learning method can learn to round up or down and select the value that minimizes the loss function described above.

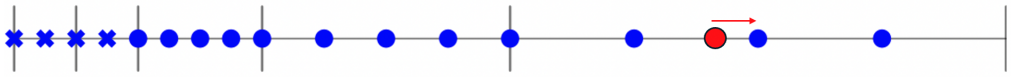


Figure 3.4: Round to nearest

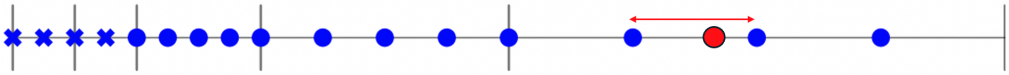


Figure 3.5: Rounding learning

Overall, our quantization starts with the initialization dataset described in Section 3.2.1 to select the optimal encoding for the target FP format and bias value, followed by rounding learning using the calibration dataset to further recover the lost task performance from quantization.

3.3 Summary

In this chapter, we first characterize the computational and memory demands of Stable Diffusion during inference, particularly its U-Net component, which dominates inference latency as the U-Net is run for multiple denoising timesteps during the generation of a single image or a batch of images. Detailed analysis shows that most of the inference time on a GPU is spent on Conv2d and linear layers, with memory traffic playing a significant role in performance, especially with larger batch sizes. Peak VRAM usage during inference can be substantial, driven mainly by attention layers, highlighting the need for efficient memory management. By quantizing both the weights and activations to FP8 or FP4, the VRAM requirement could be reduced by 4× and 8× respectively. We introduce a method for floating-point quantization of diffusion models. This involves selecting optimal encoding and bias values to minimize quantization error. While FP8 quantization maintains performance close to full precision, FP4 results in noticeable quality degradation, which motivates the investigation of gradient-based rounding techniques to enhance the accuracy of low-bitwidth floating-point quantization.

Chapter 4

Experiments and Results

4.1 Methodology

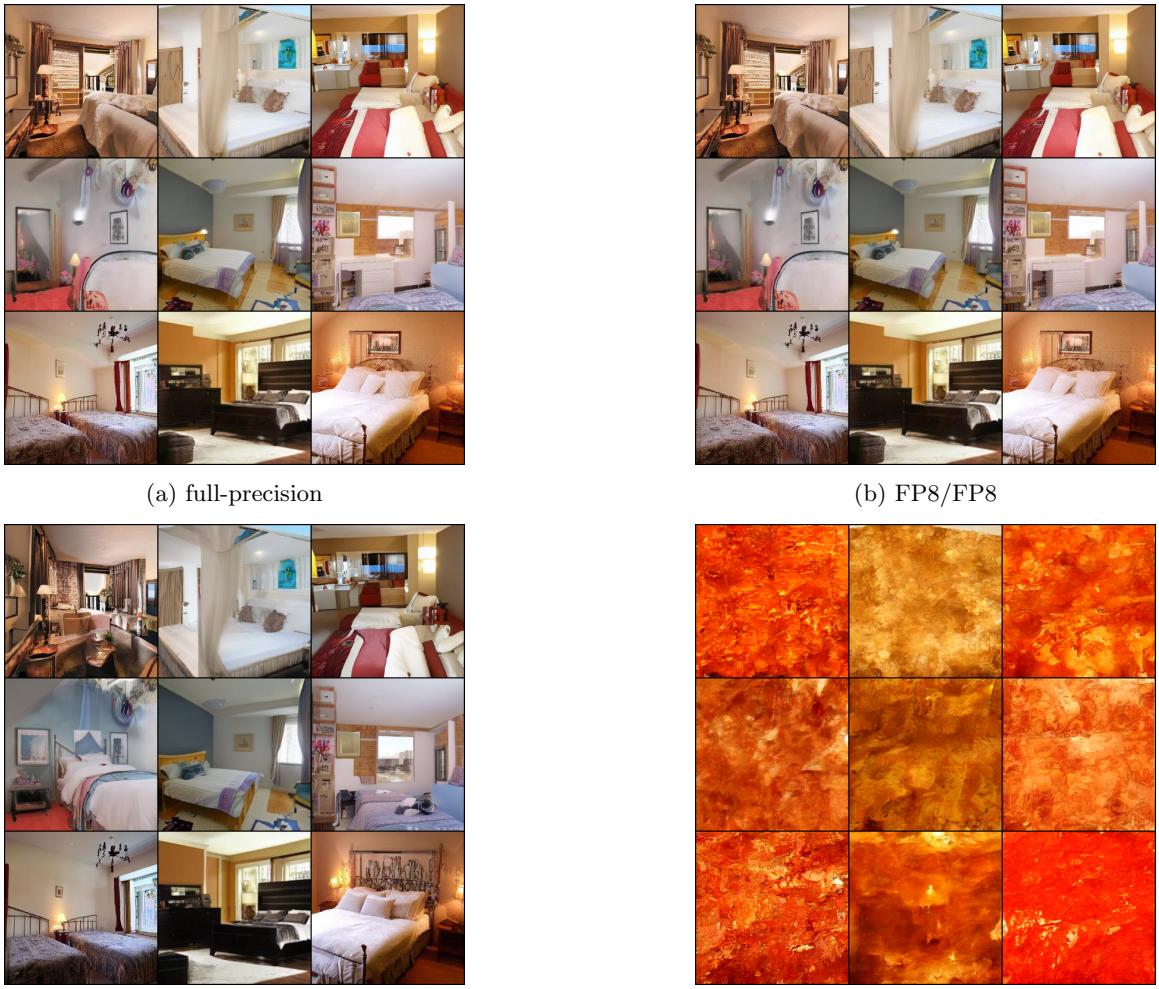
To evaluate our quantization methods, we conduct image synthesis experiments using state-of-the-art diffusion models: DDIM [53], Latent Diffusion Model (LDM), Stable Diffusion [43] and Stable Diffusion XL (SDXL) [41]. We evaluate the models on two tasks: unconditional generation and text-to-image generation. For unconditional generation, we use LDM trained on the LSUN-Bedrooms dataset (256×256) [62] and DDIM trained on CIFAR-10 [20], and generate 50,000 samples, following Q-diffusion [27]. For text-to-image generation, we use Stable Diffusion V1.5, sampling 2,000 prompts from the MS-COCO dataset [31] and generating 10,000 samples. For SDXL, we generate 2,000 samples for evaluation. We set 200 denoising steps for unconditional generation and 50 steps for text-to-image generation, consistent with the original LDM settings.

To construct the initialization dataset for searching mantissa bits and bias, we uniformly sample 128 samples from all denoising timesteps for unconditional generation and 16 samples for text-to-image generation. The number of samples in the calibration dataset is chosen empirically by comparing the task performance of the quantized model using different calibration data sizes. To construct the calibration dataset for rounding learning, we sample 5 samples from each step for unconditional generation and 20 samples from each step for text-to-image generation.

During each iteration of gradient-based rounding learning, we randomly select 16 samples for unconditional generation and 8 samples for text-to-image generation from the calibration dataset. The number of samples in the calibration dataset is chosen empirically by comparing the task performance of the quantized model using different calibration data sizes. To compare our method with integer quantization for diffusion models, we use Q-diffusion [27], the state-of-the-art integer quantization method, instead of PTQD [13], which focuses on quantization noise correction rather than the quantization method itself. Q-diffusion recommends separately quantizing the skip connection and the previous layer's output before concatenation due to their different value distributions. We apply this technique to floating-point quantization as well.

In all our experiments, we first quantize the weights and then the activations, with quantization performed on a per-tensor basis. For both tasks, as most of the inference time is spent on convolution and linear layers, and following standard practice, we quantize the weights and activations of the convolutional and linear layers, while leaving the normalization layers and SiLU activation function

in full precision. Since SiLU has no weights and the normalization layers have very few, quantizing them offers minimal benefit. The majority of the computational workload, as shown in Section 3.1, is concentrated in the linear and convolutional layers. This is exactly the same approach used by prior integer quantization methods for diffusion models.



(a) full-precision

(b) FP8/FP8

(c) FP4/FP8

(d) FP4/FP8 without rounding learning

Figure 4.1: LDM(LSUNbedroom) Qualitative Evaluation: example images generated by different quantized models

4.2 Image Generation Quality Metrics

We report four metrics: Fréchet inception distance (FID) [15], Spatial Fréchet Inception Distance (sFID) [40], Precision, and Recall [23, 46]. For FID and sFID, we first extract features from the generated images and the reference images using Inception V3 [55], and then compare the distance between the *distributions* of the two sets of features. The difference between FID and sFID is that sFID uses spatial features rather than the standard pooled features, providing a sense of spatial distributional similarity between the images. Since this is a distance metric, lower values indicate

better quality. Mathematically, FID is defined as follows:

$$FID = ||\mu_g - \mu_r||^2 - Tr(\Sigma_g + \Sigma_r - 2\sqrt{\Sigma_g \Sigma_r}) \tag{4.1}$$

where μ_g and Σ_g represent the mean and the covariance of the extracted features from the generated images, while μ_r and Σ_r represent the mean and covariance of the extracted features from the reference images. sFID is calculated using the same formula as FID except that sFID uses spatial features.

Precision is the probability that a random image from the generated images falls within the support of the reference image distribution. *Recall* is the probability that a random image from the reference images falls within the support of the generated image distribution. Precision and recall are defined as follows:

$$precision(\Phi_r, \Phi_g) = \frac{1}{|\Phi_g|} \sum_{\phi_g \in \Phi_g} f(\phi_g, \Phi_r) \tag{4.2}$$

$$recall(\Phi_r, \Phi_g) = \frac{1}{|\Phi_r|} \sum_{\phi_r \in \Phi_r} f(\phi_r, \Phi_g) \tag{4.3}$$

$f(\phi, \Phi)$ is a binary function defined as:

$$f(\phi, \Phi) = \begin{cases} 1, & \text{if } ||\phi - \phi'||_2 \leq ||\phi' - NN_k(\phi', \Phi)||_2 \text{ for at least one } \phi' \in \Phi \\ 0, & \text{otherwise} \end{cases} \tag{4.4}$$

where $NN_k(\phi', \Phi)$ returns kth nearest feature vector of ϕ' from set Φ.

For text-to-image generation, we also report the *CLIP score*[14], which measures the similarity between the prompts and generated images. It is important for text-to-image generation that the generated images are both high-quality and relevant to the input prompts. CLIP score is defined as follows:

$$CLIPScore(I, C) = max(100 * cos(E_I, E_C), 0) \tag{4.5}$$

where $cos(E_I, E_C)$ corresponds to the cosine similarity between visual CLIP embedding E_I for an image I and textual CLIP embedding E_C for an caption C.

We make the key observation that commonly used metrics do not always reflect changes in output image quality. To demonstrate this, we collect numerical metrics and visually inspect several output samples. There are cases where the metrics fail to fully describe the effects on quality because they use distributions as proxies and compare the distance between the distributions of reference and generated images. Often, quantization effects are not captured, when reference images differ significantly from the generated ones. We propose a better methodology for choosing reference images to more accurately evaluate the effects of quantization on output quality.

4.3 Facilitating Fair Comparisons Across Runs

Since the diffusion models use randomly generated noise as input, they naturally produce different images each time they are invoked. However, to facilitate proper comparisons across different ex-

periments we fix the seed across runs that are to be compared. This way the input noisy data to the model remains the same for each batch at different runs. This ensures the generation of "identical" images for each batch, facilitating a meaningful comparison.

4.4 Unconditional Image Generation

In this section, we evaluate our quantization methods on the LDM pre-trained on the 256×256 LSUN-Bedroom dataset and DDIM pre-trained on the 32×32 CIFAR-10 dataset, and compare against state-of-the-art integer quantization [27]. We use the same reference dataset used in Q-diffusion to measure all the metrics.

Quantitatively evaluating generative models is a challenging task, and it is possible that different metrics show different trends. Therefore, we include four metrics for a more comprehensive evaluation and to compare to Q-diffusion. The Q-diffusion study uses only FID for the evaluation of unconditional generation.

Table 4.1 and Table 4.2 report the generated image quality metrics for different quantization methods and bitwidth targets. Recall that FP32/FP32 (weights/activations) is the baseline full-precision model. All other rows that use floating point were generated via our methods.

Table 4.1 shows the generated image quality metrics for DDIM pre-trained on CIFAR-10. Both INT8/INT8 and FP8/FP8 models maintain the full-precision model performance, while there is minimal degradation when the weights are quantized to 4 bits. Although INT4/INT8 slightly outperforms FP4/FP8 in FID and Recall, FP4/FP8 is significantly better in sFID.

Table 4.2 shows the generated image quality metrics for LDM pre-trained on LSUNBedroom dataset. Comparing FP8/FP8 with INT8/INT8 and FP4/FP8 with INT4/INT8 shows that the models quantized with our methods outperform the corresponding integer quantized models of the same bitwidths in three out of the four metrics. INT8/INT8 scores higher only in Recall, and INT4/INT8 scores higher in Precision. Regardless, both score differences are minor. Moreover, as we discuss below, inspection of several, randomly selected output images shows *no* noticeable degradation of output quality.

The FP8/FP8 model outperforms the full-precision model in FID and Precision (we will discuss this result further during the visual inspection of output images). In contrast, the INT8/INT8 model struggles to maintain the full-precision model's task performance, as measured by FID. This indicates that the FP8/FP8 quantized model does not degrade image quality, while reducing memory and compute costs by 4x. Furthermore, the rounding learning method is essential for FP4/FP8, as without it, all FP4/FP8 metrics indicate major failure, with Precision equal to 0.00 and Recall equal to 0.0146, suggesting that the generated images and reference images are significantly different. Later in this section, we show that these images are, in fact, close to random noise.

Recall, the FID reports that FP8/FP8 is slightly better than the full-precision model. This is not impossible since it has been observed that in neural networks, sometimes a loss in precision can act as a regularizer defending against overfitting. Figure 4.1 shows a few examples of the generated images which are representative of the random samples of output images we visually inspected. Our visual inspection of the images from the full-precision model (part (a)) and from the FP8/FP8 model (part (b)), reveals *no* obvious change in image quality.

Considering images generated by the FP4/FP8 model (part (c)), we observe that the color is not

Table 4.1: CIFAR10 Quantitative Evaluation

Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision (FP32/FP32)	4.20	4.44	0.6657	0.5847
INT8/INT8	4.02	4.73	0.6406	0.5970
FP8/FP8 (Ours)	3.70	4.31	0.6619	0.5954
INT4/INT8	4.67	5.94	0.6496	0.5820
FP4/FP8 no RL (Ours)[1]	135.75	41.66	0.5724	0.1233
FP4/FP8 (Ours)	5.03	4.89	0.6513	0.5816

[1]refers to no rounding learning

Table 4.2: LDM(LSUNBedroom) Quantitative Evaluation

Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision (FP32/FP32)	2.95	7.05	0.6494	0.4754
INT8/INT8	3.29	7.51	0.6394	0.4806
FP8/FP8 (Ours)	2.93	7.44	0.6559	0.4706
INT4/INT8	4.36	7.99	0.6598	0.4404
FP4/FP8 no RL (Ours)[1]	288.21	151.96	0.00	0.0146
FP4/FP8 (Ours)	3.84	7.36	0.6247	0.4742

[1]refers to no rounding learning

as bright as the images from the full-precision model or FP8/FP8 model, but the degradation in terms of the image composition is negligible. On the other hand, without applying the gradient-based rounding learning method on FP4 quantization, the quantized model fails to generate meaningful images (part (d)).

Table 4.3: Stable Diffusion Quantitative Evaluation (Reference: MS-COCO)

Reference Dataset	**MS-COCO**			
Metric / Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision	22.71	64.81	0.6892	0.3966
INT8/INT8	21.29	63.55	0.6817	0.4230
FP8/FP8 (Ours)	22.23	64.17	0.6896	0.3924
INT4/INT8	20.95	63.54	0.6737	0.4313
FP4/FP8 no rounding learning (Ours)	262.63	266.53	0.007	0.00
FP4/FP8 (Ours)	21.75	64.93	0.6526	0.4232

4.5 Text-to-Image Generation

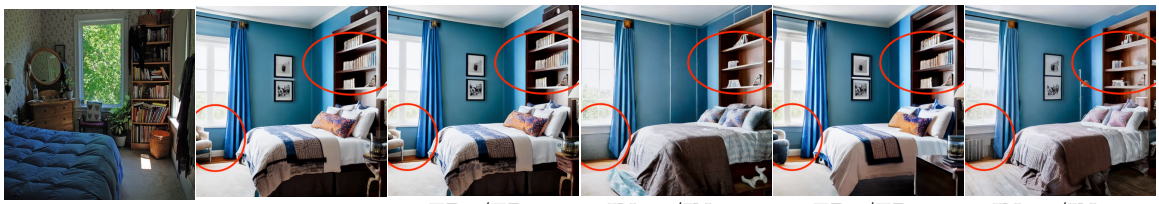
This section evaluates our quantization method on Stable Diffusion and SDXL pre-trained on the 512×512 LAION-5B dataset. We calculate the output quality metrics using the MS-COCO 2017 validation dataset as reference images following the methodology of Q-diffusion. As shown in Table 4.3, both integer and floating-point quantized models maintain performance of the full precision model as reported by the metrics, with integer quantization achieving lower FID and sFID scores

Table 4.4: Stable Diffusion Quantitative Evaluation (Reference: Full-Precision Model Generated Images)

Reference Dataset	**Full Precision Model Generated Images**			
Metric / Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision	0.00	0.00	1.00	1.00
INT8/INT8	5.53	33.38	0.8293	0.8659
FP8/FP8 (Ours)	2.76	18.50	0.9744	0.9746
INT4/INT8	5.85	33.97	0.8141	0.8474
FP4/FP8 no rounding learning (Ours)	253.25	220.53	0.0215	0.00
FP4/FP8 (Ours)	5.53	31.73	0.8426	0.8925

Prompt: A woman stands in the dining area at the table

Prompt: Bedroom scene with a bookcase, blue comforter and window



MS-COCO full-precision FP8/FP8 INT8/INT8 FP4/FP8 INT4/INT8

Figure 4.2: Stable Diffusion Qualitative Evaluation

compared to floating-point quantization at the same quantization setting. Further, as we quantize the models to lower bitwidths for both integer and floating-point quantization, the metrics show improvements compared to higher bitwidth settings.

Contrary to what the metrics suggest, qualitative inspection reveals that image quality degrades with lower bitwidth in integer quantization. Floating-point quantized models produce better-quality images, closer to those generated by the full-precision model. Figure 4.2 shows several examples illustrating the poor quality from integer quantization. In the first row, faces generated by integer quantized models are blurry, and table utensils have disappeared. In contrast, floating-point quantized models maintain facial features and table details. In the second row, integer models fail to produce physically meaningful books on the bookcase and the chair beside the bed has vanished. Moreover, the INT4/INT8-generated images show less diversity compared to INT8/INT8-generated ones. For example, in the first row, the chairs and tables have disappeared in the background.

The above observations suggest that the current methodology used to evaluate the effect of quantization on output quality ought to be revisited. Below we present what we believe is a better methodology for doing so.

A Better Methodology To Measure Output Quality: Without the loss of generality we focus our discussion on the FID metric.

The FID metric's goal is to measure how similar two *sets* of images are. In our application, FID is meant to compare the quality between a reference "ground-truth" *set* of images vs. a sample of the *set* of images the model produces. That is, in FID we do not compare a single reference image vs. another produced image that is supposed to be as identical as possible. FID converts each of the image sets into a distribution and then compares these distributions. The assumption is that the distributions are good enough proxies for the image content.

We first note that the images generated by the full-precision model differ significantly from the reference images from MS-COCO. This should have been expected since Stable Diffusion was trained on the LAION-5B dataset [48] and not on MS-COCO. Using a data distribution as a representative summary of the image set is less meaningful in this context given that the image sets differ by design.

The MS-COCO contains real-world images and was used as reference images in the original work that proposed Stable Diffusion to evaluate whether the generated images are close to real-world images. Since we are not making any modifications to the full-precision model architecture and since the goal of quantization is to reduce memory and compute costs while maintaining full-precision model task performance, we propose to use images generated by the full-precision model as the reference set, when evaluating the effects caused by quantization.

In this revised approach, Table 4.4 shows that, when using the same bitwidth, floating-point quantization outperforms integer quantization in all metrics. The FP8/FP8 model's FID is 2.77 lower (better) vs. the INT8/INT8, whereas the FP4/FP8 model's FID is 0.32 lower vs. the INT4/INT8. Notably, FP4/FP8 achieves the same FID, and better sFID, Precision and Recall compared to INT8/INT8. Similarly to unconditional generation, the rounding learning method for FP4 weight quantization reduces the degradation significantly as all 4 metrics for FP4/FP8 without rounding learning are significantly larger than FP4/FP8 with rounding learning.

Since the currently available hardware (Nvidia H100) supports only E4M3 and E5M2 encodings for the FP8 format, we evaluate our quantization method with this limited encoding search space. As shown in Table 4.5, the FP8/FP8 model outperforms the INT8/INT8 model in three out of four

metrics, with only a negligible difference in Recall. Additionally, the FP4/FP8 model surpasses the INT4/INT8 model in all four metrics. Compared to the results with a broader search space including four encoding candidates (E2M5, E3M4, E4M3, E5M2) as shown in Table 4.4, reducing the encoding candidates has led to a slight degradation in generated image quality. Nevertheless, the floating-point quantized models continue to outperform the integer quantized models.

Moreover, we analyze the encodings selected by our quantization method for LDM (LSUN-Bedroom) and Stable Diffusion when all the aforementioned encodings are in the search space for FP8 and FP4. Figure 4.3 (a) illustrates that when weights are quantized to FP8, both LDM and Stable Diffusion predominantly select encodings with more bits allocated to the mantissa, as evidenced by the near-zero percentages of E5M2 and E4M3 for both models. Additionally, 66.3% of the weights in Stable Diffusion are encoded as E2M5, whereas 62.6% of the weights in LDM are encoded as E3M4. This suggests that Stable Diffusion requires higher precision rather broader range to preserve the task performance of the full-precision model compared to LDM. When weights are quantized to FP4, both models predominantly use the E2M1 encoding, as shown in Figure 4.3 (b).

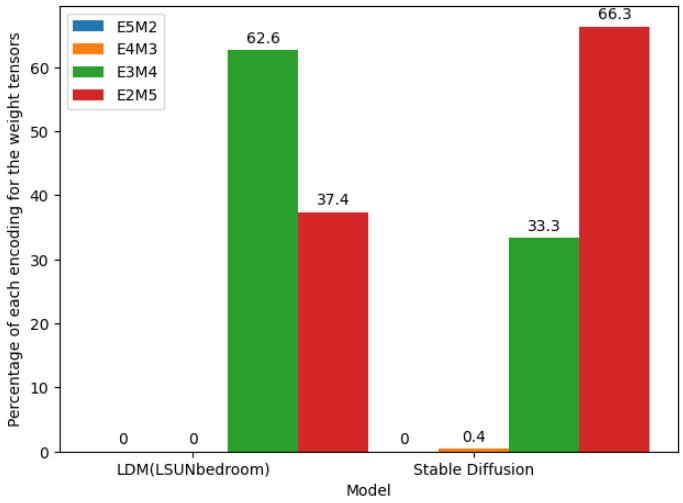
Table 4.5: Stable Diffusion Quantitative Evaluation (Reference: Full-Precision Model Generated Images), encoding candidates: E4M3, E5M2

Reference Dataset	**Full Precision Model Generated Images**			
Metric / Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision	0.00	0.00	1.00	1.00
INT8/INT8	5.53	33.38	0.8293	0.8659
FP8/FP8 (Ours)	5.34	30.96	0.9508	0.865
INT4/INT8	5.85	33.97	0.8141	0.8474
FP4/FP8 (Ours)	5.81	32.61	0.8403	0.892

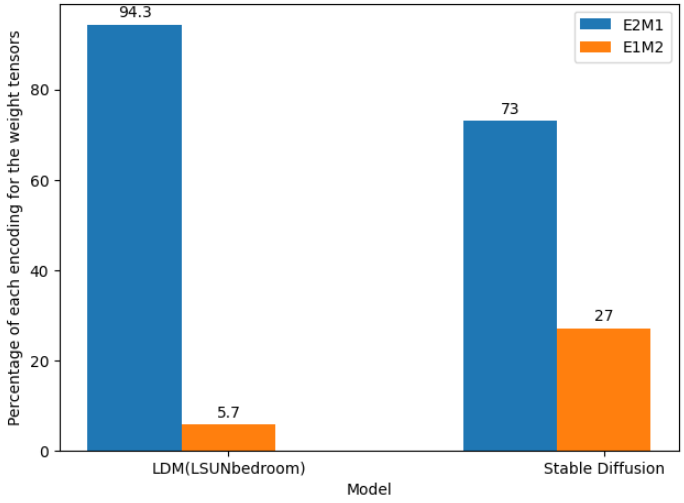
SDXL is the largest open-source image generative model, with a U-Net that is approximately three times larger than that of Stable Diffusion. Table 4.6 presents the image generation quality metrics for FP8 and INT8 quantized models. The FP8/FP8 model significantly outperforms the INT8/INT8 model across all four metrics. Figure 4.4 provides an example of the images generated by each model. The image generated by the FP8/FP8 model closely resembles the one produced by the full-precision model. In contrast, the image generated by the INT8/INT8 model is vastly different and lacks many details (e.g., the stop sign is absent).

Table 4.6: SDXL Quantitative Evaluation

Reference Dataset	**Full-precision Generated Images**			
Bitwidth (W/A)	***FID*↓**	***sFID*↓**	***Precision*↑**	***Recall*↑**
Full Precision	0.00	0.00	1.00	1.00
INT8/INT8	94.22	247.42	0.135	0.681
FP8/FP8 (Ours)	39.52	229.21	0.5125	0.894



(a) FP8



(b) FP4

Figure 4.3: Percentage of each encoding selected by our quantization method in the weights

(a) full-precision (b) FP8/FP8 (c) INT8/INT8

Figure 4.4: SDXL Qualitative Evaluation

4.6 CLIP Score

Thus far, the metric reported measures the output image quality with respect to a reference image set. Since Stable Diffusion is a text-to-image model, it is crucial to also measure that the generated images closely align with the input prompts. The CLIP score is the best practice metric used for this purpose.

Figure 4.5 reports the CLIP score for various bitwidths. The differences across all methods are relatively small, even when compared to the full-precision model, suggesting that all models perform reasonably well according to the CLIP score. However, our FP8/FP8 and FP4/FP8 quantized models consistently exhibit better CLIP scores compared to integer quantized models, with the FP4/FP8 model achieving a slightly better CLIP score than the full-precision model.

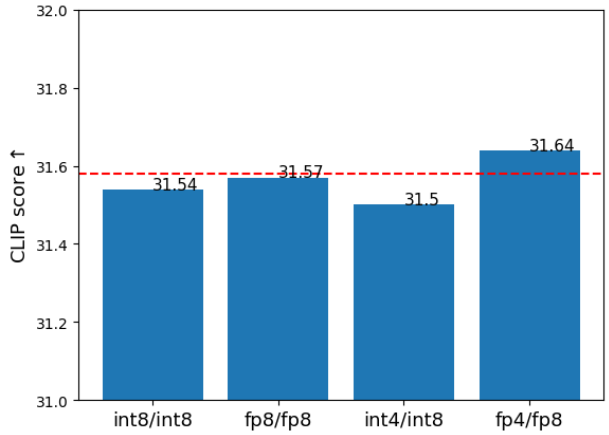


Figure 4.5: Stable Diffusion CLIP Score. The dotted red line indicates the CLIP score of images generated by the full-precision model

4.7 Sparsity

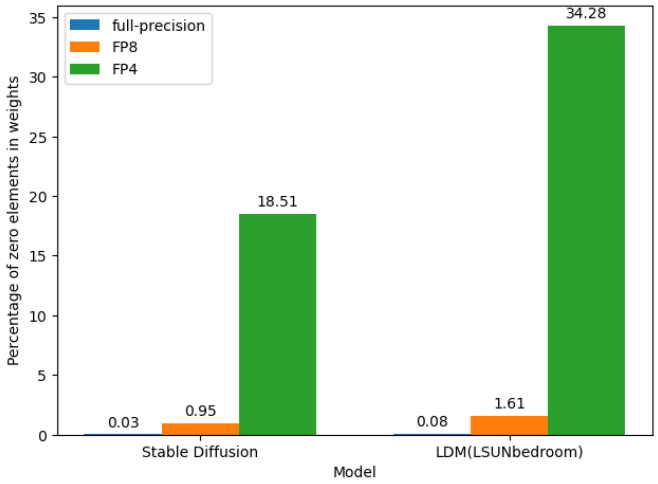


Figure 4.6: Percentage of weights that are zero in Stable Diffusion and LDM(LSUNbedroom)

Since ultimately the goal of methods such as quantization is to reduce overall memory and compute costs, we also study the effect of quantization on another behavior that neural networks often exhibit, i.e., sparsity. Sparsity, that is the fraction of values in a tensor that are zero, can be used to reduce memory footprint and transfers via appropriate encoding, and to reduce computations by skipping multiplications.

Since quantization reduces data precision, it can naturally introduce sparsity by forcing values that are close to zero to become zero. Many previous works [6, 37, 24, 58, 17] exploit sparsity in deep neural networks to reduce compute and memory costs. For example, NVidia GPUs via cuSCNN [6] add hardware support for *structural sparsity*, allowing for up to a certain pattern and number of zeros in subgroups of a tensor [37].

We measure the sparsity in floating-point quantized models. Figure 4.6 illustrates that for the weights of Stable Diffusion, our methods introduce a 31.6× and 617× increase in sparsity, when quantizing to FP8 and FP4, respectively, compared to the full-precision model. For LDM (LSUNBedroom), our methods introduce a 20.1× and 428.5× increase in sparsity, when quantized to FP8 and FP4, respectively, compared to the full-precision model. Leveraging the methods that exploit sparsity mentioned above, the inference latency and energy efficiency of diffusion models can be further improved together with our quantization methods.

4.8 Summary

The experiments evaluate our quantization method using state-of-the-art diffusion models (DDIM, LDM, Stable Diffusion, SDXL) on two tasks: unconditional generation and text-to-image generation. The models are evaluated using metrics such as FID, sFID, Precision, and Recall. The experiments demonstrate that FP8/FP8 quantization often matches or outperforms the full-precision models in quality, while FP4/FP8 quantization shows more degradation unless rounding learning is applied. Visual inspections support these findings, especially highlighting the importance of rounding learning for FP4 quantization. The study also fixes seeds across runs to ensure consistent comparisons.

Quantitative results show that FP8/FP8 models perform better than INT8/INT8, with FP4/FP8 outperforming INT4/INT8 in most metrics when rounding learning is used. Without rounding learning, FP4/FP8 models produce poor results, often close to random noise. Stable Diffusion is one of the most popular applications of diffusion models. FP8/FP8 Stable Diffusion achieves 2.77 lower(better) FID, 14.88 better(lower) sFID, 14.5% higher in precision and 10.9% higher recall compared to INT8/INT8 model. Notably, the FP4/FP8 model ahieves 0.32 lower FID, 2.24 lower sFID, 2.8% higher precision and 4.5% higher recall compared to the INT8/INT8 model. With rounding learning, the FID and sFID is reduced by 247.72 and 188.8 respectively for the FP4/FP8 model.

The findings indicate that floating-point quantization, particularly with rounding learning, is effective in maintaining image quality while reducing memory and computation costs. Additionally, our floating-point quantization method increases the sparsity of the quantized models by an order of magnitude, which offers further optimization opportunities.

Chapter 5

Conclusion

This thesis focuses on quantizing diffusion models to achieve high-performance inference. Our work is the first to quantize the diffusion model weights to FP8 or FP4 values, and the activations to FP8. By exploring the application of floating-point quantization at these specific bitwidths, we provide a comprehensive analysis and comparison against traditional integer quantization techniques. Our findings reveal that the task performance degradation on both unconditional and text-to-image generation caused by our floating-point quantization is significantly less than that caused by integer quantization at the same bitwidth, demonstrating a clear advantage in preserving generated image fidelity. In addition to this, we present a novel methodology to evaluate the effects of quantization on diffusion models, which offers a more accurate and nuanced understanding of the trade-offs involved.

Our work not only advances the current state of research but also lays a strong foundation for future exploration by identifying key areas where quantization can be further optimized. One of the most promising avenues for future research involves leveraging the sparsity introduced in floating-point numbers through quantization to enhance the efficiency of diffusion model inference. This could involve investigating advanced techniques for sparse matrix computations, exploring hardware accelerations specifically designed for sparse data processing, and developing adaptive algorithms capable of dynamically responding to the sparsity patterns that emerge in quantized models. By pursuing these avenues, we aim to push the boundaries of high-performance diffusion model inference, ultimately contributing to more efficient and effective deep learning models.

Bibliography

[1] Michael Andersch. *NVIDIA Hopper Architecture In-Depth*. March 22, 2022. 2022. URL: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML]. URL: https://arxiv.org/abs/1607.06450.

[3] Mart van Baalen, Andrey Kuzmin, Suparna S Nair, Yuwei Ren, Eric Mahurin, Chirag Patel, Sundar Subramanian, Sanghyuk Lee, Markus Nagel, Joseph Soriaga, and Tijmen Blankevoort. *FP8 versus INT8 for efficient deep learning inference*. 2023. arXiv: 2303.17951 [cs.LG]. URL: https://arxiv.org/abs/2303.17951.

[4] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. *Low-bit Quantization of Neural Networks for Efficient Inference*. 2019. arXiv: 1902.06822 [cs.LG].

[5] Prafulla Dhariwal and Alex Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG].

[6] Mohamed A. Elgammal, Omar Mohamed Awad, Isak Edo Vivancos, Andreas Moshovos, and Vaughn Betz. "cuSCNN: an Efficient CUDA Implementation of Sparse CNNs". In: *Proceedings of the 13th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. HEART '23. ¡conf-loc¿, ¡city¿Kusatsu¡/city¿, ¡country¿Japan¡/country¿, ¡/conf-loc¿: Association for Computing Machinery, 2023, pp. 107–113. ISBN: 9798400700439. DOI: 10.1145/3597031.3597057. URL: https://doi.org/10.1145/3597031.3597057.

[7] Hongxiang Fan, Gang Wang, Martin Ferianc, Xinyu Niu, and Wayne Luk. "Static Block Floating-Point Quantization for Convolutional Neural Networks on FPGA". In: *2019 International Conference on Field-Programmable Technology (ICFPT)*. 2019, pp. 28–35. DOI: 10.1109/ICFPT47387.2019.00012.

[8] Alex Finkelstein, Ella Fuchs, Idan Tal, Mark Grobman, Niv Vosco, and Eldad Meller. "QFT: Post-training Quantization via Fast Joint Finetuning of All Degrees of Freedom". In: *Computer Vision – ECCV 2022 Workshops*. Ed. by Leonid Karlinsky, Tomer Michaeli, and Ko Nishino. Cham: Springer Nature Switzerland, 2023, pp. 115–129. ISBN: 978-3-031-25082-8.

[9] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. 2023. arXiv: 2210.17323 [cs.LG]. URL: https://arxiv.org/abs/2210.17323.

[10] Sam Greydanus. *Three Perspectives on Deep Learning*. 2016.

[11] Song Han, Huizi Mao, and William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2016. arXiv: 1510.00149 [cs.CV]. URL: https://arxiv.org/abs/1510.00149.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[13] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. *PTQD: Accurate Post-Training Quantization for Diffusion Models*. 2023. arXiv: 2305.10657 [cs.CV].

[14] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. *CLIPScore: A Reference-free Evaluation Metric for Image Captioning*. 2022. arXiv: 2104.08718 [cs.CV].

[15] 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*. 2018. arXiv: 1706.08500 [cs.LG].

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[17] Shen-Fu Hsiao, Kun-Chih Chen, Chih-Chien Lin, Hsuan-Jui Chang, and Bo-Ching Tsai. "Design of a Sparsity-Aware Reconfigurable Deep Learning Accelerator Supporting Various Types of Operations". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10.3 (2020), pp. 376–387. DOI: 10.1109/JETCAS.2020.3015238.

[18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. *Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference*. 2017. arXiv: 1712.05877 [cs.LG]. URL: https://arxiv.org/abs/1712.05877.

[19] Leela S. Karumbunathan. *NVIDIA Jetson AGX Orin Series*. July 2022. 2022. URL: https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf.

[20] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[22] Andrey Kuzmin, Mart Van Baalen, Yuwei Ren, Markus Nagel, Jorn Peters, and Tijmen Blankevoort. *FP8 Quantization: The Power of the Exponent*. 2024. arXiv: 2208.09225 [cs.LG].

[23] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. *Improved Precision and Recall Metric for Assessing Generative Models*. 2019. arXiv: 1904.06991 [stat.ML].

[24] Alberto Delmas Lascorz, Mostafa Mahmoud, Ali Hadi Zadeh, Milos Nikolic, Kareem Ibrahim, Christina Giannoula, Ameer Abdelhadi, and Andreas Moshovos. "Atalanta: A Bit is Worth a "Thousand" Tensor Values". In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS '24. La Jolla, CA, USA: Association for Computing Machinery, 2024, pp. 85–102. ISBN: 9798400703850. DOI: 10.1145/3620665.3640356. URL: https://doi.org/10.1145/3620665.3640356.

[25] Haoying Li, Yifan Yang, Meng Chang, Huajun Feng, Zhihai Xu, Qi Li, and Yueting Chen. *SRDiff: Single Image Super-Resolution with Diffusion Probabilistic Models*. 2021. arXiv: 2104.14951 [cs.CV].

[26] Xin Li, Yulin Ren, Xin Jin, Cuiling Lan, Xingrui Wang, Wenjun Zeng, Xinchao Wang, and Zhibo Chen. *Diffusion Models for Image Restoration and Enhancement – A Comprehensive Survey*. 2023. arXiv: 2308.09388 [cs.CV].

[27] 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 (ICCV)*. Oct. 2023, pp. 17535–17545.

[28] Yanjing Li, Sheng Xu, Xianbin Cao, Xiao Sun, and Baochang Zhang. "Q-DM: An Efficient Low-bit Quantized Diffusion Model". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 76680–76691. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/f1ee1cca0721de55bb35cf28ab95e1b4-Paper-Conference.pdf.

[29] 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*. 2021. arXiv: 2102.05426 [cs.LG].

[30] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. *AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration*. 2024. arXiv: 2306.00978 [cs.CL]. URL: https://arxiv.org/abs/2306.00978.

[31] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[32] Jiaqi Liu, Peng Hang, Xiaocong Zhao, Jianqiang Wang, and Jian Sun. *DDM-Lag : A Diffusion-based Decision-making Model for Autonomous Vehicles with Lagrangian Safety Enhancement*. 2024. arXiv: 2401.03629 [cs.RO].

[33] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. "LLM-FP4: 4-Bit Floating-Point Quantized Transformers". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, pp. 592–605. DOI: 10.18653/v1/2023.emnlp-main.39. URL: http://dx.doi.org/10.18653/v1/2023.emnlp-main.39.

[34] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. *RePaint: Inpainting using Denoising Diffusion Probabilistic Models*. 2022. arXiv: 2201.09865 [cs.CV].

[35] Calvin Luo. *Understanding Diffusion Models: A Unified Perspective*. 2022. arXiv: 2208.11970 [cs.LG].

[36] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. *SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations*. 2022. arXiv: 2108.01073 [cs.CV].

[37] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. *Accelerating Sparse Deep Neural Networks*. 2021. arXiv: 2104.08378 [cs.LG].

[38] Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. *Up or Down? Adaptive Rounding for Post-Training Quantization*. 2020. arXiv: 2004.10568 [cs.LG].

[39] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. *A White Paper on Neural Network Quantization*. 2021. arXiv: 2106.08295 [cs.LG].

[40] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W. Battaglia. *Generating Images with Sparse Representations*. 2021. arXiv: 2103.03841 [cs.CV].

[41] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. *SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis*. 2023. arXiv: 2307.01952 [cs.CV]. URL: https://arxiv.org/abs/2307.01952.

[42] Ethan Pronovost, Meghana Reddy Ganesina, Noureldin Hendy, Zeyu Wang, Andres Morales, Kai Wang, and Nicholas Roy. *Scenario Diffusion: Controllable Driving Scenario Generation With Diffusion*. 2023. arXiv: 2311.02738 [cs.LG].

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV].

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[45] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[46] Mehdi S. M. Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. *Assessing Generative Models via Precision and Recall*. 2018. arXiv: 1806.00035 [stat.ML].

[47] Dave Salvator. *H100 Transformer Engine Supercharges AI Training, Delivering Up to 6x Higher Performance Without Losing Accuracy*. March 22, 2022. 2022. URL: https://blogs.nvidia.com/blog/h100-transformer-engine/#:~:text=Tensor%20Core%20operations%20in%20FP8,of%20smaller%2C%20faster%20numerical%20formats..

[48] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. *LAION-5B: An open large-scale dataset for training next generation image-text models*. 2022. arXiv: 2210.08402 [cs.CV].

[49] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. "Post-training Quantization on Diffusion Models". In: *CVPR*. 2023.

[50] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. *OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models*. 2024. arXiv: 2308.13137 [cs.LG]. URL: https://arxiv.org/abs/2308.13137.

[51] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[52] Junhyuk So, Jungwon Lee, Daehyun Ahn, Hyungjun Kim, and Eunhyeok Park. *Temporal Dynamic Quantization for Diffusion Models*. 2023. arXiv: 2306.02316 [cs.CV]. URL: https://arxiv.org/abs/2306.02316.

[53] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. 2022. arXiv: 2010.02502 [cs.LG].

[54] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

[55] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[57] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. *HAQ: Hardware-Aware Automated Quantization with Mixed Precision*. 2019. arXiv: 1811.08886 [cs.CV]. URL: https://arxiv.org/abs/1811.08886.

[58] Ziheng Wang. "SparseRT: Accelerating Unstructured Sparsity on GPUs for Deep Learning Inference". In: *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. PACT '20. Virtual Event, GA, USA: Association for Computing Machinery, 2020, pp. 31–42. ISBN: 9781450380751. DOI: 10.1145/3410463.3414654. URL: https://doi.org/10.1145/3410463.3414654.

[59] Yuxin Wu and Kaiming He. *Group Normalization*. 2018. arXiv: 1803.08494 [cs.CV]. URL: https://arxiv.org/abs/1803.08494.

[60] Haojun Xia, Zhen Zheng, Xiaoxia Wu, Shiyang Chen, Zhewei Yao, Stephen Youn, Arash Bakhtiari, Michael Wyatt, Donglin Zhuang, Zhongzhu Zhou, Olatunji Ruwase, Yuxiong He, and Shuaiwen Leon Song. *FP6-LLM: Efficiently Serving Large Language Models Through FP6-Centric Algorithm-System Co-Design*. 2024. arXiv: 2401.14112 [cs.LG].

[61] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. 2024. arXiv: 2211.10438 [cs.CL]. URL: https://arxiv.org/abs/2211.10438.

[62] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. *LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop*. 2016. arXiv: 1506.03365 [cs.CV].