Technische Universität Berlin
School IV - Electrical Engineering and Computer Science
Institute of Software Engineering and Theoretical Computer Science
Machine Learning Group

A thesis submitted for the degree of Bachelor of Science

Quantization-aware neural network training

Author:
Tarek
Akrout

First examiner:
Prof. Dr. Klaus-Robert
Müller
Second examiner:
Prof. Dr. Grégoire
Montavon
Thesis Advisor:
Jannik Wolff

September 2024

Declaration of authorship

I hereby declare that the thesis submitted is my own, unaided work, completed without any external help. Only the sources and resources listed were used. All passages taken from the sources and aids used, either unchanged or paraphrased, have been marked as such. Where generative AI tools were used, I have indicated the product name, manufacturer, the software version used, as well as the respective purpose (e.g. checking and improving language in the texts, systematic research). I am fully responsible for the selection, adoption, and all results of the AI-generated output I use. I have taken note of the Principles for Ensuring Good Research Practice at TU Berlin dated 15 February 2023. `https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf` I further declare that I have not submitted the thesis in the same or similar form to any other examination authority.

Berlin, 30.09.2024

.....

Signature

Abstract

As neural networks grow, the demand for memory and computational resources increases, making it challenging to deploy these models efficiently. To address this issue, Ma et al. [2024] proposed a quantization-aware training (QAT) method that restricts network weights to ternary values to {-1, 0, 1} during inference. This method was initially proposed in the context of large language models. This thesis contextualizes this method in the broader literature and applies it to other generative models, specifically variational autoencoders (VAEs) and diffusion models. Through our experiments on synthetic datasets and the MNIST dataset, we analyze the trade-offs between quantization and model performance, highlighting QAT's advantages and limitations. Our findings indicate that Quantization-Aware Training (QAT) yields significant memory savings (i.e., reducing the storage requirements for the model weights) and, in some instances, even enhances performance. We hypothesize that the BitNet model may act as a form of regularization, helping to prevent overfitting by limiting the expressiveness of the model. However, in other cases, the performance of the BitNet model declines (relative to the baseline), likely due to the restricted weight range, which may limit the model's capacity to capture more complex patterns in the data.

Zusammenfassung: Mit dem Wachstum neuronaler Netze steigt auch der Bedarf an Speicher- und Rechenressourcen, was den effizienten Einsatz dieser Modelle erschwert. Um dieses Problem anzugehen, hat Ma et al. [2024] eine quantisierungssensitive Trainingsmethode (QAT) vorgeschlagen, die die Netzwerkgewichte während der Inferenz auf ternäre Werte wie {-1, 0, 1} beschränkt. Diese Methode wurde ursprünglich im Zusammenhang mit großen Sprachmodellen vorgeschlagen. Diese Arbeit kontextualisiert diese Methode in der breiteren Literatur und wendet sie auf andere generative Modelle an, insbesondere auf Variational-Auto-Encoder (VAEs) und Diffusionsmodelle. Durch unsere Experimente mit synthetischen Datensätzen und dem MNIST-Datensatz analysieren wir die Kompromisse zwischen Quantisierung und Modellleistung und zeigen die Vorteile und Grenzen von QAT auf. Unsere Ergebnisse zeigen, dass quantisierungssensitives Training (QAT) signifikante Speichereinsparungen (hinsichtlich der Speicherung der Modellgewichte) ermöglicht und in einigen Fällen sogar die Leistung verbessert. Wir stellen die Hypothese auf, dass die Quantisierung als eine Art Regularisierung fungieren kann, die dazu beiträgt, eine Überanpassung an die Trainingsdaten zu verhindern, indem sie die Expressivität des Modells einschränkt. In anderen Fällen ist die Leistung des BitNet-Modells jedoch schlechter als die der Baseline, was wahrscheinlich auf die Quantisierung der Modellgewichte zurückzuführen ist, der die Fähigkeit des Modells, komplexere Muster in den Daten zu erfassen, einschränken kann.

Contents

List of figures

List of tables

Chapter 1

Introduction

General problem Neural networks have made significant strides in both capability and scale, leading to transformative advancements across numerous fields. However, as these models continue to grow in capability and scale, they present significant challenges, particularly in terms of computational demands, extended training times, high inference latency, increased energy consumption, and difficulties in deploying them within resource-constrained environments [Wan et al., 2023]. For example, OpenAI's GPT-3 [Brown et al., 2020], with its 175 billion parameters, and Meta's LLaMA 3 [Dubey et al., 2024], which scales up to 405 billion parameters, exemplify the immense scale of modern language models. These models are capable of generating human-like text, performing complex reasoning, and understanding nuanced language. However, their impressive capabilities come at a high cost, requiring vast computational resources for training and deployment. As neural networks continue to scale, the need to optimize their efficiency has become a critical area of research. Efficient neural networks are essential for enabling the deployment of advanced models across a wide range of applications, including image and video processing, natural language understanding, gaming, robotics, and medical diagnostics [Sze et al., 2017].

Overview of existing approaches and their limitations To address these challenges, several techniques have been developed. These include post-training quantization [Shen et al., 2024], which reduces the precision of model parameters after training; parameter pruning [Ma et al., 2023], which involves removing less significant weights; low-rank approximation [Hu et al., 2021], which approximates weight matrices with lower-rank structures; and knowledge distillation [Hinton et al., 2015], where a smaller model is trained to mimic the behavior of a larger, more complex model.

Despite the effectiveness of these methods, they have certain limitations. For instance, post-training quantization can lead to significant performance degradation when aggressive quantization is applied (e.g., using fewer than 8 bits), as the model has not been optimized for quantized weights during the training process. Low-rank adaptation (LoRA), for example, is used to fine-tune models and usually does not result in faster inference time. In light of these limitations, we focus on quantization-aware training (QAT), a method that integrates the quantization process into the training phase.

Quantization-aware training Quantization-aware training (QAT) enables models to account for the limitations of low-precision weights during the training phase, often leading to superior performance compared to post-training quantization methods, which apply quantization only after training is complete [Krishnamoorthi, 2018]. One of the pioneering works to incorporate quantization during training was the Binarized Neural Networks (BNN) paper

by Courbariaux et al. [2016], followed by further advancements like the DoReFa-Net framework [Zhou et al., 2016], which introduced more sophisticated quantization schemes. In recent years, QAT has garnered increasing attention, particularly in the field of large language models (LLMs), with approaches like BitNet [Wang et al., 2023] and BitNet b1.58 [Ma et al., 2024] demonstrating how QAT can facilitate efficient training of LLMs while maintaining performance at reduced precision levels. The method introduced by Ma et al. [2024] reduces the weight representation to just three values, which is equivalent to using $\log_2(3) \simeq 1.58$bits, hence the name of the model **BitNet b1.58**. During training, the model still utilizes floating-point values for its weights; however, they are adjusted to conform to the constraints of quantization, ensuring they are distributed across the quantization range. During inference, the model uses only the ternary quantized weights weights. This architecture significantly decreases the model's size and the memory needed to store it. Additionally, the reduction in bit-width transforms floating-point multiplications into simpler additions, thus potentially speeding up computation [Ma et al., 2024]. Ma et al. [2024] show that the BitNet b1.58, starting from a 3B model size, matches the full precision LLaMA model in terms of perplexity and task performance while being up to 3.55 times more memory efficient and 2.71 times faster.

Research question Our research focuses on the most recent advancements in **QAT**, specifically the use of ternary weights {-1, 0, 1} introduced by BitNet [Ma et al., 2024]. While recent advancements have shown that quantization-aware training works effectively with large, complex (language) models like GPT, which leverage vast amounts of text data, our aim is to investigate whether this approach is equally effective with alternative generative architectures, smaller models, and fewer data. Specifically, we explore its use in generative models like variational autoencoders (**VAEs**) [Kingma and Welling, 2013] and **diffusion models** [Sohl-Dickstein et al., 2015, Ho et al., 2020], which are widely used for generative tasks but differ from language models in structure and data requirements.

This thesis seeks to determine whether QAT can maintain competitive performance relative to full-precision baselines while reducing the memory footprint and potentially reducing inference time. Additionally, we investigate the internal properties of QAT-trained models, such as their weight distribution and quantization error dynamics, to further understand this model class.

Method and contributions We apply the BitNet quantization method, initially designed for large language models, to VAEs and diffusion models. We compare this model with baseline models (non-quantized models). For this, we first conduct a hyperparameter search (grid search) for both model classes and compare the respective optimal models to assess the trade-off between quantization (as measured by memory efficiency) and performance (e.g., as measured by reconstruction quality and log-likelihood) across different data types, including simple synthetic datasets and the more complex MNIST dataset [LeCun et al., 1998]. Additionally, we investigate properties of the quantized models, e.g., how quantization-aware training adapts weights toward ternary values.

Our experiments reveal that the BitNet model achieves competitive results for some data with significantly reduced model size (which we measure in terms of memory usage). We also evaluate properties of the quantized model, e.g., we observe an intriguing phenomenon where the loss function stabilizes early, but quantization error continues to decrease, suggesting ongoing adaptation of the model to quantized weights even after convergence.

Chapter 2

Related work

This section reviews the existing literature on various model compression techniques that can be used for deploying deep learning models on resource-constrained devices and, more generally, reducing latency, memory footprint, and operational costs. The primary techniques covered include post-training quantization, parameter pruning, low-rank adaptation, and knowledge distillation [Han et al., 2016, Cheng et al., 2020]. Additionally, we will explore relevant methodologies from adjacent fields like federated learning. We will analyze the strengths and shortcomings of each approach, motivating our choice of quantization technique.

2.1 Post-training quantization

Quantization is the process of reducing the (computational or memory) complexity of neural networks by lowering the number of bits used to represent the parameters of a model. Post-Training Quantization (PTQ) involves reducing the precision of pre-trained neural network's weights, typically converting 32-bit floating-point (FP32) representations to lower-precision formats such as 8-bit integers (INT8) [Jacob et al., 2018]. PTQ is applied after the initial training phase and is primarily used to compress models for deployment on resource-constrained devices [Gholami et al., 2022]. Various levels of precision are used in PTQ, each with distinct trade-offs in terms of compression rate, inference speed, and accuracy.

For instance, recent advancements have introduced FP8 quantization, which can significantly reduce precision while maintaining high accuracy. A notable example is the Llama3 model [Dubey et al., 2024], which applies FP8 quantization to all feed-forward layers (which account for 50% of the inference compute time), with negligible degradation in model performance. Another popular PTQ technique is INT8 quantization, which maps floating-point values to an integer range (from -128 to 127) and provides better model compression and faster inference on edge devices. Jacob et al. [2018].

A commonly used function for integer quantization can be expressed as follows:

$$Q(r) = \text{Int}\left(\frac{r}{S}\right) - Z, \tag{2.1}$$

where Q is the quantization function, r represents the real-valued input (either activation or weight), S is a scaling factor, and Z is the integer zero point. The Int function converts the real number to an integer by rounding (e.g., rounding to the nearest integer or truncating). Essentially, this function maps real numbers r to corresponding integer values. This type of quantization is referred to as uniform quantization, where the quantized values (or quantization levels) are evenly distributed.[Wu et al., 2020]

We can reconstruct the real values r using dequantization with the following formula [Gholami et al., 2022] :

$$\tilde{r} = S(Q(r) + Z),$$

where $\tilde{r}$ is an approximation to r but not equal due to rounding.

While post-training quantization can significantly reduce the model size, it often results in a notable loss of accuracy and might require fine-tuning. The loss in accuracy increases as the precision level gets lower, especially under 8 bits. This occurs because the model is not optimized for the quantized representation during its training phase [Ma et al., 2024].

2.2 Parameter pruning

Parameter pruning [Srinivas and Babu, 2015, Ma et al., 2023] aims to reduce the size and complexity of a neural network by, for example, identifying and eliminating redundant parameters.

Srinivas and Babu [2015] propose a popular pruning technique. They start by noticing that neurons with a weight of 0 do not impact the model's accuracy and can be removed. Similarly, they suggest that neurons with weight vectors that have small magnitude have less of an impact than other ones. They also observe that for a layer, where $W_i = W_j$ for two neurons i and j with weight vectors $W_i, W_j \in \mathbb{R}^n$, the layer's size can be reduced by one by merging the two neurons and adding their coefficients as shown in Fig. 2.1. Since, in practice, two weight vectors will not be equal, the authors minimize the expected value of $\mathbb{E}[(z_n - z_{n-1})^2]$, where z_n denotes the output of a layer with n neurons and z_{n-1} the output of layer z_n after merging two neurons. Specifically, they minimize an upper bound for $\mathbb{E}[(z_n - z_{n-1})^2]$.

Another class of pruning methods utilizes the Hessian of the loss function to identify connections that contribute negligibly to model performance, allowing for a targeted reduction in parameters without significant loss in accuracy [Cheng et al., 2020].

One drawback of pruning techniques is the need for manual configuration and fine-tuning of layer sensitivities, which ensures effective pruning without significant performance loss but adds complexity and effort to network optimization [Cheng et al., 2020]. This complexity mirrors some of the challenges seen in other techniques, such as low-rank adaptation, which also seeks to reduce model complexity, albeit in a different way.

2.3 Low-rank adaptation

Low-rank adaptation (LoRA), introduced by [Hu et al., 2021], is a powerful model compression approach used for fine-tuning pre-trained large models for specific tasks. Although it is

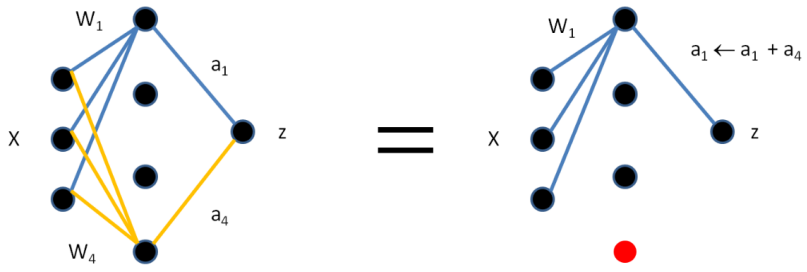


Figure 2.1: Pruning: Merging two neurons (a_1 and a_4) and updating the model's weights [Srinivas and Babu, 2015].

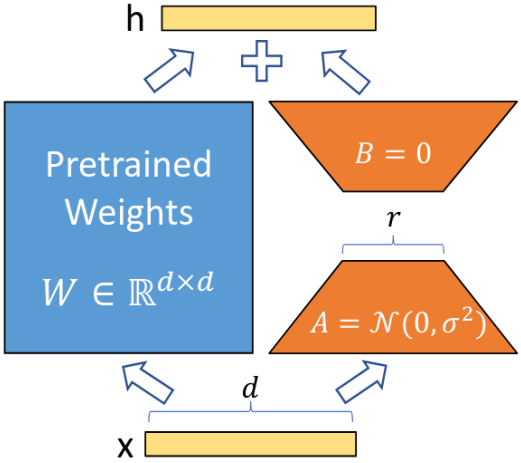


Figure 2.2: LoRA: Illustration of the use of two smaller matrices A and B with rank r to fine-tune the model's weights

commonly applied to large language models (LLMs), its utility extends beyond them. One of the key challenges in fine-tuning LLMs is their massive size, which results in a high number of parameters and requires substantial computational resources, such as GPUs and memory, making it costly and resource-intensive. LoRA addresses this by reducing the number of trainable parameters during fine-tuning, enabling efficient adaptation without needing to retrain or store the entire model, thus mitigating the issues associated with the large size of LLMs.

Motivated by the research from Aghajanyan et al. [2020], the authors Hu et al. [2021] hypothesize that the change in the weights during fine-tuning (i.e, adaptation of the network to specific tasks) has a "low intrinsic rank", meaning that there exists a low-dimensional reparametrization that is as effective as reparametrizing the full dimensional space for fine-tuning.

In a layer with weights matrix $W_0 \in \mathbb{R}^{d \times k}$, instead of updating all the parameters, LoRA freezes the weights and adds two trainable matrices of lower rank $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$ where $r \ll \min(d, k)$. The product $BA \in \mathbb{R}^{d \times k}$ represents the update to the original frozen weights, where the weights W_0 are ignored in the backward pass and not updated, which enables faster fine-tuning (Fig. 2.2). During inference, the weights are updated with $W = W_0 + \Delta W = W_0 + BA$. Since LoRA uses two lower-rank matrices, it requires fewer parameters to change and, therefore, takes less time to fine-tune. It also does not increase inference time since the model's size stays the same. The forward pass for an input x is given by:

$$h = W_0 x + \Delta W x = W_0 x + BAx. \tag{2.2}$$

A notable limitation is that the low-rank decomposition operation can be computationally expensive and requires extensive retraining to achieve convergence [Cheng et al., 2020]. Additionally, finding the optimal rank can be daunting because the ideal rank that balances compression and performance varies across different layers and models, and is often not straightforward to determine [Cheng et al., 2020].

2.4 Knowledge distillation

Knowledge distillation (KD) [Hinton et al., 2015] involves training a smaller student model to mimic the behavior of a larger, pre-trained teacher model. One common approach is to

use the teacher's softened output logits, which are passed through a softmax function with a temperature parameter to provide smoother probability distributions, for the student model to learn from [Hinton et al., 2015]. For classification problems, the loss function for the student model can consist of two parts, the L2 loss between the predicted class and the actual class, and a cross-entropy between the logits of the teacher model and the student model. Other techniques include matching the teacher model's weights [Romero et al., 2015] or features at different layers [Huang and Wang, 2017].

While KD aims to transfer the teacher's performance and generalization capabilities to a more compact and efficient student model, it has limitations. For instance, KD methods are often tied to the architecture of the teacher model, making the process less flexible across different model types [Alemdar et al., 2017]. Moreover, KD typically lags behind other model compression techniques, such as parameter pruning, in terms of performance [Cheng et al., 2020].

2.5 Federated learning

Federated learning is a collaborative machine learning approach where multiple devices train models locally on their data and then send the model updates to a global model. Sparse Ternary Compression (STC) is a method proposed by Sattler et al. [2020] that addresses the challenge of communication efficiency in federated learning, particularly from non-i.i.d. data. STC extends top-k gradient sparsification with downstream compression, ternarization, and optimal Golomb encoding of weight updates. Although STC uses ternary encoding, it is different from previously mentioned methods since the quantization is applied to the gradients and weight updates instead of the weights themselves.

The importance of compression in federated learning highlights the broader applicability of model compression techniques. Methods like Deep Context-Adaptive Binary Arithmetic Coding (CABAC) [Wiedemann et al., 2020], which is a universal algorithm that is used for compression and transmission of model weights for deep neural networks, serve as an example. The compression pipeline consists of sparsification, followed by quantization, fine-tuning, and finally, lossless compression (using CABAC) where the quantized weights are binarized through a series of operations. Examples : $1 \rightarrow 100$, -4 $\rightarrow$ 111101, $7 \rightarrow 10111010$.

However, the focus of this thesis is primarily on compression techniques applied directly to model weights during training, rather than weight updates or gradients, as seen in federated learning. While STC and similar methods are promising for distributed learning scenarios, they fall outside the scope of our work.

Chapter 3

Quantization-aware training

In the previous chapter, we explored various model compression techniques, including post-training quantization (PTQ), parameter pruning, and low-rank adaptation, each with its own strengths and limitations in reducing model complexity and maintaining performance. For example, although PTQ is effective for compressing models after training, it often results in significant performance degradation, particularly at lower precision levels. In contrast, Quantization-Aware Training (QAT), the focus of this chapter and this thesis, attempts to overcome these limitations by incorporating quantization directly into the training process [Krishnamoorthi, 2018]. Specifically, QAT allows the model to adapt to reduced precision during training, mitigating the performance drop typically seen in PTQ. During training, QAT simulates low-bit integer arithmetic for both the forward and backward passes, while retaining floating-point precision for weights and activations [Krishnamoorthi, 2018, Wu et al., 2020].

3.1 Overview of QAT approaches

Jacob et al. [2017] provided an effective QAT method for 8-bit quantization in convolutional neural networks (CNNs), enabling these models to maintain high accuracy even in reduced-precision formats. Similarly, Krishnamoorthi [2018] extended this by offering a more comprehensive understanding of low-bit quantization, particularly as it applies to deeper and larger models.

Further advancements have explored more extreme quantization methods, such as binary and ternary quantization, as seen in the work of Courbariaux et al. [2016] and Zhou et al. [2016] with Binarized Neural Networks (BNNs) and DoReFa-Net. These approaches limit model weights and activations to just 1- or 2-bit representations, drastically reducing memory and computational requirements while still achieving reasonable performance.

In the context of large language models (LLMs), the need for efficient quantization methods has become even more critical. This is where the BitNet model, introduced by Wang et al. [2023], Ma et al. [2024], stands out. We specifically focus on BitNet because it has demonstrated strong results in reducing memory-related and computational overhead while maintaining competitive performance in large-scale LLMs despite only using ternary weights ({-1, 0, 1}).

We present this novel technique in the next section as we will use it in our experiments in Chapter 5. We choose this approach, instead of other approaches such as Binarized Neural Networks (BNNs), because it has demonstrated superior efficiency and performance.

3.2 BitNet 1.58

High-level overview

- **Training**: Weights are adapted during training to operate under quantization constraints. While the weights remain in FP32 format throughout the training process, QAT ensures they are adjusted so that when quantized during inference, they are near the discrete values {-1, 0, 1}.
- **Inference**: After training, weights are fully quantized to ternary values, which reduces memory usage and potentially speeds up inference time.
- **Quantization Scheme**: Weights are fully quantized to ternary values {-1, 0, 1} during inference, while activations are quantized to 8-bits.

In the next paragraphs, we will dive into the specifics of how the BitNet model functions, providing a detailed explanation of its mechanisms and key components.

Quantization of weights The following quantization functions are used for the weights [Ma et al., 2024]:

$$\text{RoundClip}(w, a, b) = \max(a, \min(b, \text{round}(w))), \tag{3.1}$$

$$\widetilde{W} = \text{RoundClip}\left(\frac{W}{\beta + \epsilon}, -1, 1\right), \quad \text{and } \beta = \frac{1}{nm}\sum_{ij}|W_{ij}|, \tag{3.2}$$

The Roundclip function Eq. (3.1) rounds a float to the nearest integer, and the values greater than a and less than b get clipped. In Eq. (3.2), β represents the mean of the absolute values of the weight matrix W and $\widetilde{W}$ the weight matrix with quantized values to {-1,0,1}. ϵ is added to avoid division by 0.

Quantization of activations The activations (the input tensors multiplied with the weight matrices) are scaled to $[-Q_b, Q_b]$, where $Q_b = 2^{b-1}$ and b is the number of bits being used (fixed to 8 in our experiments), using the following formulas [Wang et al., 2023]:

$$\text{Clip}(x, a, b) = \max(a, \min(b, x)), \quad \gamma = \|x\|_\infty, \quad \text{and} \tag{3.3}$$

$$\tilde{x} = \text{Quant}(x) = \text{Clip}\left(x \times \frac{Q_b}{\gamma}, -Q_b + \epsilon, Q_b - \epsilon\right). \tag{3.4}$$

In the above equations, $\tilde{x}$ represents the quantized input x, and γ is the infinite norm of the input x (maximum of the absolute value of each component of x).

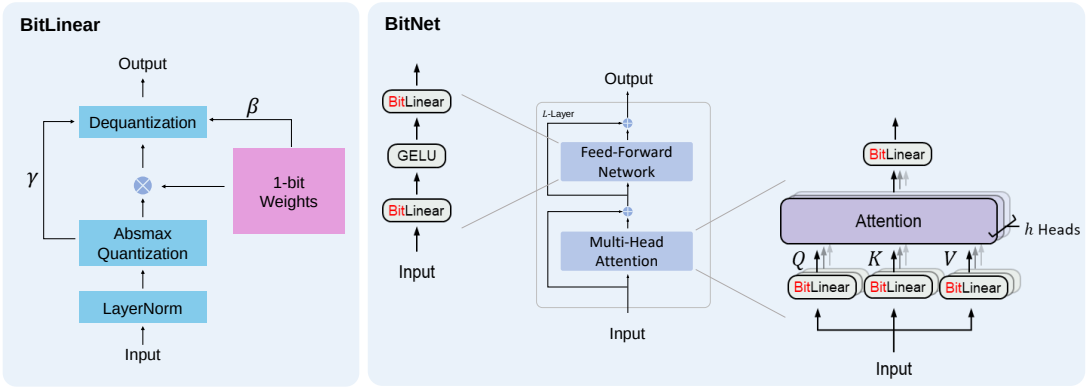


Figure 3.1: Illustration of the BitLinear layer and the BitNet model's architecture. (figure from [Wang et al., 2023])

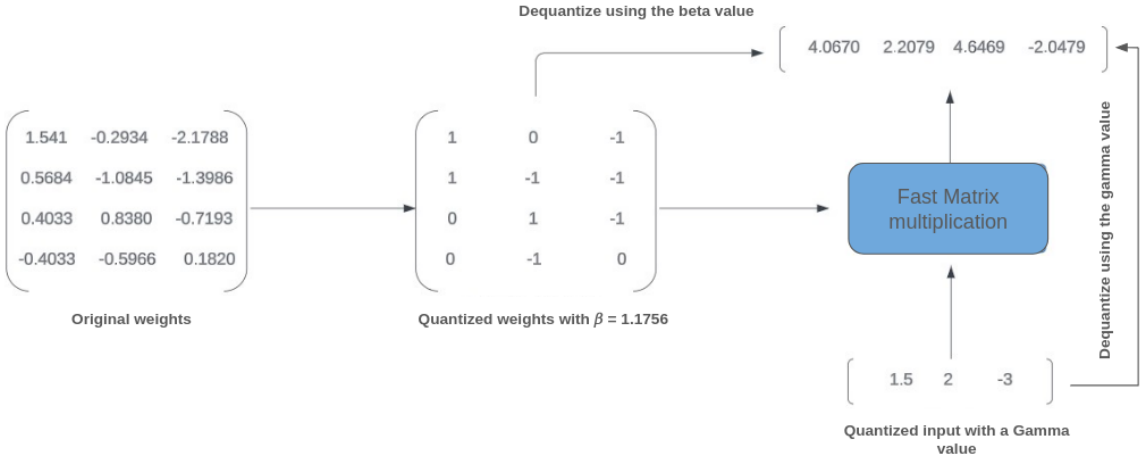


Figure 3.2: Illustration of the "fake" quantization process

Fake quantization Instead of multiplying the full-precision FP32 weight matrix W with the inputs x, the weights are first quantized to a lower-precision format (ternary values in the case of the BitNet model), then multiplied with the input (fast operation), and finally the output is dequantized using the β and γ values from the quantization functions. The operation can be modeled by the following expression:

$$W \cdot x \approx d(q(W)) \cdot d(q(x)) = \beta q(W) \cdot \gamma q(x) = q(W) \cdot q(x) \cdot \beta \cdot \gamma \tag{3.5}$$

where d and q are the dequantize and quantize operations respectively. We show an example of the fake quantization process in Fig. 3.2.

Fig. 3.1 illustrates the architecture of the BitNet model, which incorporates BitLinear layers that perform the "fake" quantization (the BitLinear layer was introduced by Wang et al. [2023], but the same architecture is used by Ma et al. [2024] with different quantization functions). The method is called "fake" quantization since the weights are still stored in full precision during training and they just simulate the quantization process.

Normalization of activations Layer normalization [Ba et al., 2016] is a widely used technique to stabilize and accelerate the training process, especially in deep neural networks. It normalizes the input features by subtracting the mean and dividing by the standard deviation.

This is followed by a learned affine transformation which ensures that the model maintains the capacity to learn different scales and shifts in the data, preserving its ability to approximate complex functions. In comparison, BitNet uses RMSNorm [Zhang and Sennrich, 2019], a variation that normalizes the input using only the root mean square (RMS) instead of the full mean and variance:

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(a)} g_i, \quad \text{and} \quad \mathrm{RMS}(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}, \tag{3.6}$$

where $\bar{a}_i$ is the normalized output for the i-th element after applying RMS normalization, a_i is the original input value or activation for the i-th element before normalization, and g_i is a scaling parameter (often referred to as a gain parameter) that is learned during training. RMSNorm tends to be more computationally efficient than layer normalization and can achieve similar performance without the dependence on the full statistical information of the input.

Training and the Straight-through Estimator Since the BitLinear layer involves operations such as clipping and rounding, calculating the gradient during backpropagation is not possible since these functions are not continuous and, therefore, not differentiable. To circumvent this problem, the BitNet model uses a straight-through estimator (STE) [Bengio et al., 2013, Yin et al., 2019].

In neural network software frameworks like PyTorch [Paszke et al., 2017], a computational graph is created that keeps track of the functions being applied to the input variables of the neural network, which allows calculating derivatives for backpropagation. When using the STE, all operations (including the quantization operations such as clipping and rounding) are applied during the forward pass, allowing the network to operate with quantized values, however, during backpropagation, the STE bypasses the non-differentiable nodes where quantization occurs. In this way, it calculates the gradient based on the unquantized input.

We present how the STE works in the following code snippet (this function is part of a "BitLinear" class):

```
def forward(self, input):

    quant_input, gamma = activation_quant(input, self.input_bits)
    # input_bits are set to 8
    quant_weight, beta = weight_quant(self.weight)

    # implementation of the STE
    # detach() removes the quantization from the computational graph
    quant_input = input + (quant_input - input).detach()
    quant_weight = self.weight + (quant_weight - self.weight).detach()

    out = F.linear(quant_input, quant_weight)/ beta / gamma
    # The F.linear should be replaced by a custom kernel function to
       observe the improvements

    if self.bias is not None:
        out += self.bias.to(device).view(1, -1).expand_as(out)

    return out
```

Listing 3.1: Forward pass implementation during training using the Straight-Through Estimator (STE)

Inference Once training is complete, the model weights are fully quantized to {-1, 0, 1}, eliminating the need for "fake" quantization processes described above. Only the quantized weights are stored with a dequantization value β and dequantization still happens after the activations are calculated. Additionally, since the weights are no longer being updated during inference, the use of the Straight-Through Estimator (STE) becomes unnecessary. At this stage, the model can efficiently perform forward passes with the quantized weights, enhancing memory and potentially computational performance since the weights are constrained to ternary values {-1, 0, 1} and the multiplication operation is effectively replaced with simple additions and subtractions, leading to a faster and more efficient computation process.

Connection to pruning techniques Ternary coding can be seen as a form of pruning, since it makes the weight's matrices more sparse. In our experiments, about 27% to 40% of the weights are set to 0 using BitNet (see Fig. 5.4 and Fig. 5.11), compared to 50% to 90% using common pruning methods [Cheng et al., 2023]. Although common pruning methods have higher sparsity ratios, using BitNet should still result in better inference time (with custom CUDA kernels and hardware), since we avoid float multiplication and replace it with integer addition.

Chapter 4

Variational Autoencoders and Diffusion Models

In this section, we will introduce two probabilistic generative models that we will be using in our experiments: Variational Autoencoders (VAEs) [Kingma and Welling, 2013, Kingma et al., 2019] and diffusion models [Sohl-Dickstein et al., 2015, Ho et al., 2020]. These models have a wide range of applications including computer vision, natural language generation, text-to-image generation, medical image reconstruction, and molecular graph modeling as described in the survey from Yang et al. [2024].

Generative models are a class of machine learning models that learn the distribution of the data in order to generate new samples that resemble the original ones, where x is a random variable with an underlying true distribution $p^*(x)$ (Fig. 4.1). Since we do not have access to the analytical form of the true distribution, we aim to approximate it with a machine learning model $p_\theta(x)$, parameterized by θ:

$$x \sim p_\theta(x).$$

Once we have successfully approximated the true distribution $p^*(x)$ using $p_\theta(x)$, we can then sample from the model to generate new data (Fig. 4.1).

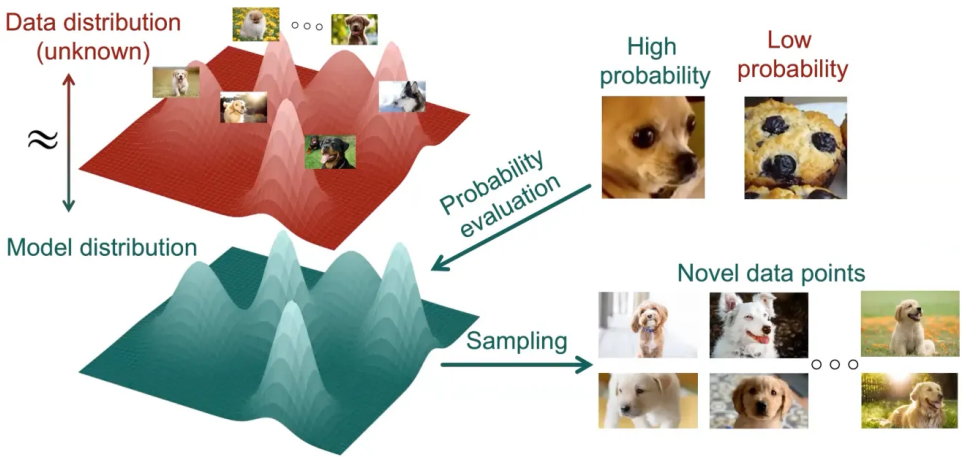


Figure 4.1: Illustration of the learned model distribution approximating the unknown true data distribution, followed by sampling new data points from the model [Charlie, 2023].

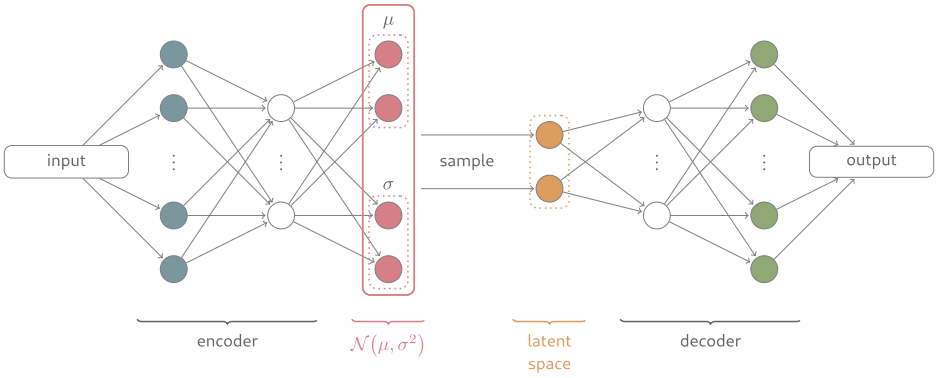


Figure 4.2: Exemplary architecture of a VAE Spinner et al. [2018]

4.1 Variational Autoencoders

VAEs [Kingma and Welling, 2013] address the challenge of modeling the complex and high-dimensional data distribution $p_\theta(x)$ by introducing a low-dimensional latent space z, where the variable z captures the essential features of the data. We may want to estimate the true posterior $p_\theta(z \mid x)$, which describes how the latent variable is distributed given the observed data. Trying to compute it directly via

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(x)} \tag{4.1}$$

involves integrating over the latent variables z:

$$p_\theta(x) = \int p_\theta(x \mid z)p_\theta(z)dz, \tag{4.2}$$

which is intractable [Kingma et al., 2019]. To overcome this problem, the method introduces an approximate posterior distribution $q_\phi(z \mid x)$, parameterized by ϕ, the variational parameters. This model is referred to as the encoder and $p_\theta(x \mid z)$ is called a probabilistic decoder, since it maps samples from the latent space back to the data space, allowing the model to generate new data points.

The parameters of the VAE are optimized by maximizing the Evidence Lower Bound (ELBO), which provides a lower bound for the log-likelihood of the data. The ELBO is given by:

$$\mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z \mid x) \parallel p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] \leq p_\theta(x), \tag{4.3}$$

where θ are the parameters of the decoder network, ϕ the parameters of the encoder network, z the latent space variable, $q_\phi(z \mid x)$ is the approximate posterior distribution, and $p_\theta(z)$ is the prior distribution over the latent variable which is usually chosen to be a simple distribution like a standard normal distribution. The derivation can be found in Appendix A.1.
The ELBO is composed of two terms:

- **The KL divergence term :** Ideally, for each data point, $q_\phi(z|x)$ should encode meaningful and relevant features of x, providing a useful latent representation. The D_{KL} term ensures that, on average across all data points, the posterior distribution $q_\phi(z|x)$ is close to the prior $p_\theta(z)$, often a standard normal distribution $\mathcal{N}(0, I)$. It can be computed using a closed formula (see Appendix A.2).

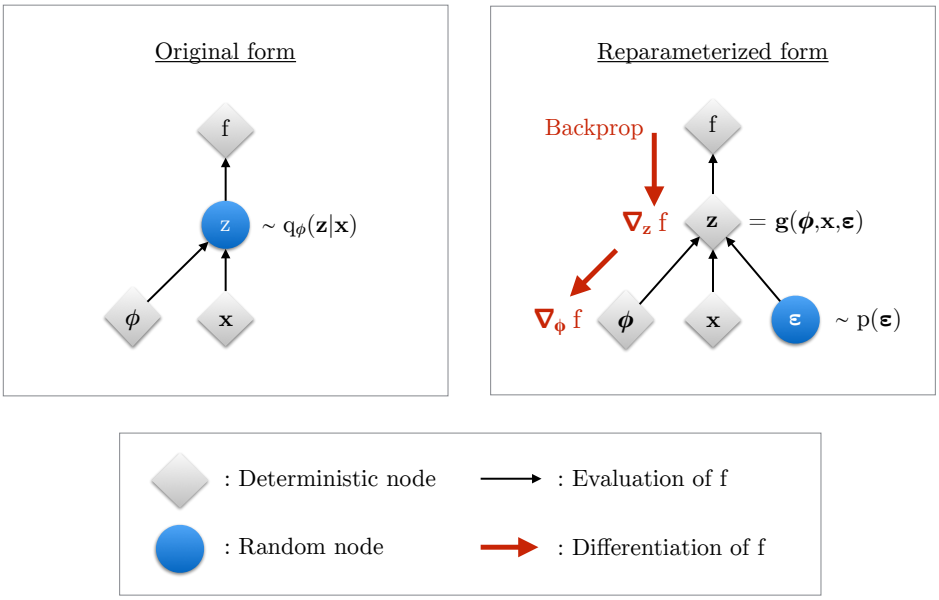


Figure 4.3: Illustration of the reparameterization trick which introduces a stochastic node ϵ so that we can compute the gradients of f w.r.t ϕ. [Kingma et al., 2019]

- The **reconstruction loss** measures the difference between the original input image and the reconstructed one (after passing the image through the encoder and decoder). This ensures that the reconstructed image is accurate and close to the original one. In our experiments (Chapter 5), we can use the mean squared error (MSE) as a measure of the reconstruction loss when the decoder outputs only the mean vector (Sections A.3 and A.4).

The encoder and decoder can both be modeled as neural networks, which are scalable and can be optimized efficiently using stochastic gradient descent (SGD) [Kingma et al., 2019]. For example, the (variational) encoder may output two latent vectors μ and σ, where σ is the diagonal of the covariance matrix Σ (the covariance matrix here is just a diagonal matrix), which parameterize a Gaussian distribution representing the region in the latent space where the input data is mapped, as shown in Fig. 4.2.

Reparametrization trick Directly sampling from the latent distribution $z \sim \mathcal{N}(\mu, \sigma)$ makes it difficult to compute gradients for backpropagation. Hence, VAEs employ the reparameterization trick (Fig. 4.3, Kingma et al. [2019]): we sample from an auxiliary variable $\epsilon \sim \mathcal{N}(0, I)$ and reparameterize the latent variable as:

$$z = \mu + \sigma \cdot \epsilon.$$

This formulation allows the sampling operation to be expressed as a deterministic function of the network parameters μ and σ. The reparameterization trick is crucial for training VAEs using standard gradient-based methods. For example, it helps reduce the variance of the gradient estimator during stochastic gradient descent, which can make training "smoother" and improve performance [Kingma et al., 2019].

Although VAEs are powerful generative models, they often encounter challenges, such as generating blurry or less detailed images. Despite this limitation, VAEs remain an effective framework for capturing complex data distributions by combining deep learning and probabilistic modeling. Their versatility makes them a key tool in generative modeling, and their

foundational principles have influenced the development of more advanced models, such as hierarchical VAEs and diffusion models, which we will introduce in the next section.

4.2 Diffusion models

Diffusion models, inspired by non-equilibrium thermodynamics, are based on the idea of progressively corrupting data with noise, which is then reversed to recover the original data distribution [Sohl-Dickstein et al., 2015]. Several classes of diffusion models exist, including Noise Conditional Score Networks (NCSNs) [Song and Ermon, 2019], and Score Stochastic Differential Equations (SDEs) SDE [2021], however, this work focuses on Denoising Diffusion Probabilistic Models (DDPMs) [Ho et al., 2020]. We chose DDPMs for their simplicity and robustness in generating high-quality samples across diverse datasets, making them ideal for our experiments. These models involve two primary stages: the forward diffusion process and the reverse generative process.

The forward diffusion process Given an image $x_0 \sim q(x_0)$, where q represents the underlying distribution of the dataset and x_0 a sample from the dataset, diffusion models gradually add Gaussian noise to the image over T steps according to a predefined noise schedule, referred to as the beta schedule (a sequence of noise levels). Specifically, at each time step t, the model generates an image x_t from the previous image x_{t-1} by adding noise proportional to a predefined variance β_t, as shown in Fig. 4.4. This process is mathematically expressed as:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I). \tag{4.4}$$

Here, $q(x_t|x_{t-1})$ is the probability distribution for the noisy image at step t, where the mean is a scaled version of the previous image x_{t-1}, and $\beta_t I$ represents the added Gaussian noise with variance β_t. The full forward diffusion process over T steps can then be described as a sequence of conditional distributions:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}). \tag{4.5}$$

The values β_t for $t \in \{1, 2, .., T\}$ are hyperparameters, fixed from the beginning, and represent the amount of added noise at time t. An example of a commonly used schedule is a linear schedule, where β_t starts small and gradually increases with a constant rate as t approaches T, allowing for controlled diffusion of the original image.

Note that there is a closed formula for computing $q(x_t|x_0)$ which allows us to predict the noisy image at time t directly from the original image. This bypasses the need to compute the intermediate steps from x_0 to x_t, allowing faster computation. The formula can be derived using induction (A.5), and can be written as follows:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I), \tag{4.6}$$

where

$$\alpha_t := 1 - \beta_t \quad \text{and} \quad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s. \tag{4.7}$$

A fundamental assumption in DDPMs is that progressively adding Gaussian noise to an image over T time steps results in the image becoming indistinguishable from pure noise. Specifically, the distribution $q(x_{1:T}|x_0)$ is expected to converge to a standard normal distribution $\mathcal{N}(0, I)$ after T-timesteps [Weng, 2021].

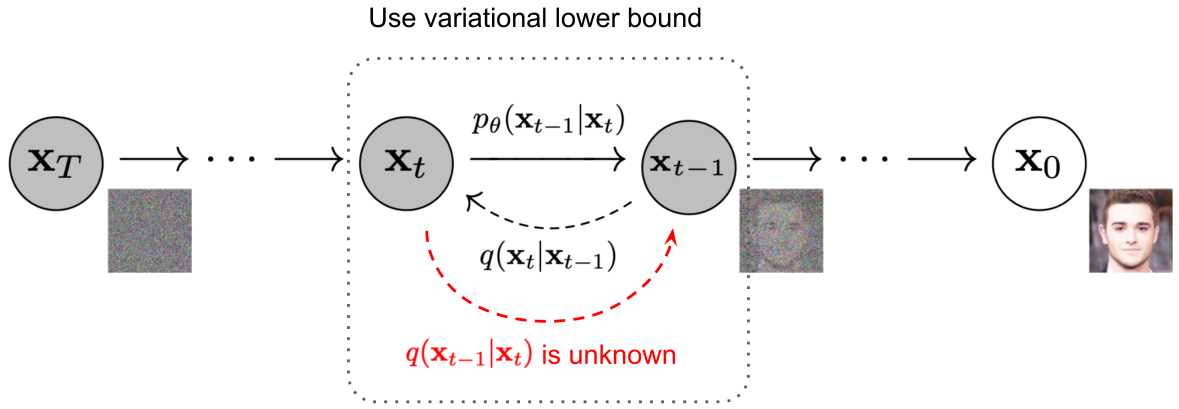


Figure 4.4: Illustration of the forward and reverse process in diffusion models [Ho et al., 2020].

The reverse diffusion process If $q(x_{t-1}|x_t)$ can be determined, then sampling $x_T \sim \mathcal{N}(0, I)$ and using $q(x_{t-1}|x_t)$ for each t would allow us to generate new images. However, this term is intractable. Therefore, the idea is to build a model with parameters θ that approximates this function. For small changes in β_t (i.e., a large number of timesteps), the reverse of the Gaussian $q(x_t|x_{t-1})$ will also be a Gaussian [Ho et al., 2020], which is why $p_\theta(x_{t-1}|x_t)$ is modeled as a Gaussian with parameters μ_θ and Σ_θ:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t), \tag{4.8}$$

where

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \tag{4.9}$$

Training objective Similar to VAEs, diffusion models use a lower bound for the log-likelihood that can be maximized. The lower bound is defined as (see Appendix A.6)

$$L_{\text{VLB}} = \mathbb{E}_{q(x_{0:T})} \left[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})} \right], \tag{4.10}$$

which can be rewritten as:

$$L_{VLB} = \mathbb{E}_q \left[\underbrace{D_{KL}(q(x_T|x_0) \parallel p_\theta(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{KL}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))}_{L_{t-1}} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0} \right]. \tag{4.11}$$

L_T can be ignored for the optimization since it is a constant (does not depend on θ). L_0 is the negative log-likelihood loss for the final denoising step, L_t for $t \in \{1, 2, .., T-1\}$ is the KL-Divergence between two normal distributions, $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_t, x_0)$ and can therefore be computed efficiently. Note that $q(x_{t-1}|x_t)$ is intractable, but can be computed when it is also conditioned on x_0, where

$$q(x_{t-1}|x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}(x_t, x_0), \tilde{\beta}_t I), \tag{4.12}$$

with

$$\tilde{\mu_t} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right), \text{ where } \epsilon_t \text{ is the noise added at time } t, \text{ and} \tag{4.13}$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t. \tag{4.14}$$

After plugging in the values in the KL Divergence term and further simplifications (e.g. omitting weighting factors β_t [Ho et al., 2020]), we arrive at a simpler form for L_t:

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[\|\epsilon_t - \epsilon_\theta(x_t,t)\|^2\right]. \tag{4.15}$$

This equation describes how learning $\mu_\theta(x_t,t)$ and $\Sigma_\theta(x_t,t)$ is equivalent to learning the added noise at each step denoted by $\epsilon_\theta(x_t,t)$.

It is important to note that this simplification leads to a theoretically less accurate approximation as the loss function no longer serves as an exact evidence lower bound (ELBO), which was the objective in the original formulation. Nevertheless, this simplified form is widely used in practice due to its computational efficiency and comparable performance in generating high-quality samples.

Diffusion models and VAEs A useful way to conceptualize diffusion models is as a type of hierarchical variational autoencoder (HVAE) with several constraints, as explored by Luo [2022]. These constraints include:

- The latent dimension is equal to the data dimension.
- The structure of the latent encoder is predefined as a linear Gaussian model with fixed parameters, centered around the previous time step's output.
- The Gaussian parameters of the encoder vary over time, with the final latent distribution being standard Gaussian.

Chapter 5

Experiments

Originally developed for large language models (LLMs), the BitNet model has shown strong performance when trained on large datasets and deployed with large-scale (Transformer-based) architectures [Ma et al., 2024]. In this chapter, we evaluate how well the BitNet method generalizes beyond the LLM space by applying and analyzing its impact on variational autoencoders (VAEs) and diffusion models (Chapter 4). For example, we explore how drastically restricting the expressivity of model weights (i.e., limiting them to ternary values {-1, 0, 1}) impacts model performance, efficiency, and quantization error. All experiments were implemented in Python using the PyTorch framework [Paszke et al., 2017].

5.1 Datasets

We evaluate the models on two datasets: 2D real-valued synthetic data and the handwritten-digits MNIST dataset LeCun et al. [1998].

Synthetic data We use the synthetic data to test the capability of the models to learn and represent simple geometric structures. We use the following distributions: anisotropic Gaussian distribution, spirals, mixture of Gaussian, moons, circles, and a more complex "Dino" dataset from Pärnamaa [2023] (Fig. 5.1). The Dino dataset consists of 8,000 data points, whereas the other datasets each contain 5,000 data points.

MNIST The MNIST dataset consists of 70,000 handwritten digits (60,000 training and 10,000 test samples), from 0 to 9. The images are grayscale, with a resolution of 28x28 pixels, and each pixel contains a value between 0 (black) and 255 (white).

5.2 Models

Here we provide more details about our implementation of the VAE and the diffusion model.

VAEs The baseline VAE model consists of a simple architecture with two fully connected layers for both the encoder and decoder, respectively. We experiment with varying the number of units in each layer, the number of hidden layers, and the dimension of the latent space to evaluate the impact of these architectural choices on model performance. The loss function is the sum of the KL divergence between the Gaussian approximate posterior $q_\phi(z \mid x)$ and the prior $p_\theta(z)$, which we choose to be a standard normal distribution $\mathcal{N}(0, I)$, and the reconstruction loss. The quantized VAE model is identical to the baseline model, except that the fully connected

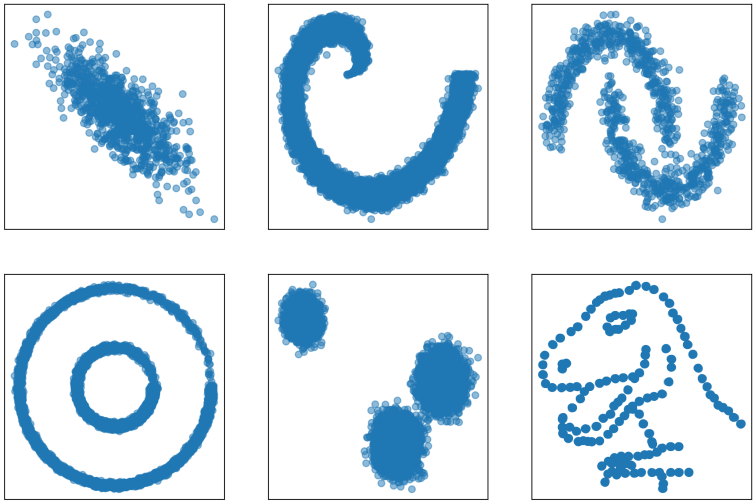


Figure 5.1: Visualizations of the synthetic datasets used for model training: (top left to bottom right) Anisotropic Gaussian, Spiral, Moons, Two Circles, Mixture of Gaussians, and Dino datasets. All data is 2D and real-valued.

layers are replaced with BitLinear layers [Ma et al., 2024], as described in Chapter 3. The general architecture of these layers is visualized in Fig. 3.1 (the figure is from Wang et al. [2023], but we use the same layers with different quantization functions as described in Chapter 3). The hyperparameters used for the VAE on both the synthetic dataset and the MNIST dataset are determined through grid search and summarized in Table B.1 and Table B.2, respectively.

We experiment with VAE decoders that either only learn the mean vector μ or learn both the mean vector μ and the variance σ. The latter models have the same architecture as the former ones but with an additional output vector for the variance.

Diffusion Models We use the Tiny Diffusion repository [Pärnamaa, 2023] as the basis for our diffusion model implementation, which is a simplified version of the Denoising Diffusion Probabilistic Models (DDPMs) framework, omitting components such as attentions blocks. The model takes as input the noisy image x_t and a timestep t then predicts the residual noise at each timestep as in Eq. (4.11). The model uses the Mean Squared Error (MSE) as the loss function for training as in Eq. (4.15). The detailed architecture of the model is outlined in Table B.3.

5.3 Results: Variational Autoencoders

This section presents the experimental results obtained by applying the baseline VAE and the BitNet VAE to both a synthetic dataset and the MNIST dataset. Since VAEs are simpler models compared to diffusion models, we begin our experimental evaluation using this model class. The experiments are designed to evaluate the trade-offs between quantization (measured by memory savings) and model performance (measured in terms of precision, recall, and approximate log-likelihood) when using quantization-aware training (Chapter 3).

5.3.1 Synthetic data

The first experiment aims to explore the behavior of the VAE with QAT on a simplified, synthetic dataset.

Hyperparameter search In order to evaluate the performance of the BitNet model and compare it to the baseline model, we conducted a grid search over the hyperparameters to select both models, respectively. We found that the Sigmoid activation function yielded the best results for both VAEs on the spiral dataset, while ReLU failed to produce viable reconstructions. Other hyperparameters, such as the learning rate, batch size, normalization layer, number of hidden layers, and units per layer, had minimal impact on the model's ability to generate data unconditionally. Additionally, applying layer normalization (RMSNorm) did not seem to affect the unconditional sampling of the BitNet model.

Results Figure 5.2 visualizes (from left to right) the initial data, the latent representation for $q_\phi(z|x)$ during reconstruction, the decoder output during reconstruction, and the decoder output during unconditional sampling for both VAE models.

For the anisotropic Gaussian data and the spiral data, both models learn an aggregated posterior that closely approximates the prior, and both successfully capture the structure of the data. However, the spiral data presents challenges to both models, as the spirals are not perfectly generated, showing the model's limitations in capturing more complex data.

In the case of the two-circles dataset, the BitNet model fails to learn the complete structure, outputting only a single circle during reconstruction. In contrast, the baseline model is able to generate two circles but with a significant amount of noise and without learning an aggregated posterior close to a Gaussian distribution. Instead, the learned function resembles an identity function, indicating that the baseline model may not fully capture the data's underlying complexity, which explains why the reconstruction is better than the unconditional samples.

Ideally, the model should balance both objective -accurately capturing the data's structure while aligning with the prior- but neither the BitNet model nor the baseline fully achieves this balance.

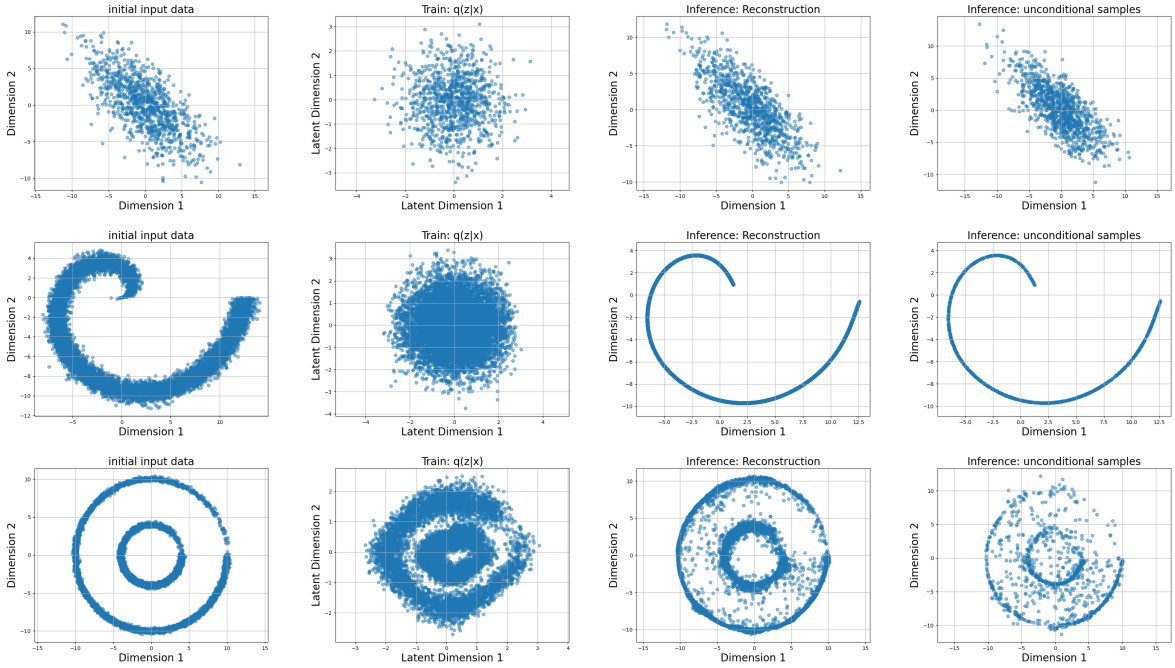
For the two-circles dataset, the baseline model struggles to fit the prior, leading to noisier reconstructions, while the BitNet model, though failing to accurately reconstruct the data, aligns more closely with the prior. This suggests that the limited expressivity of the BitNet model may serve as a regularizer, preventing overfitting and the formation of overly complex solutions.

Approximate Log-Likelihood To quantitatively evaluate the performance of both models, we compute the log-likelihood using the following equation:

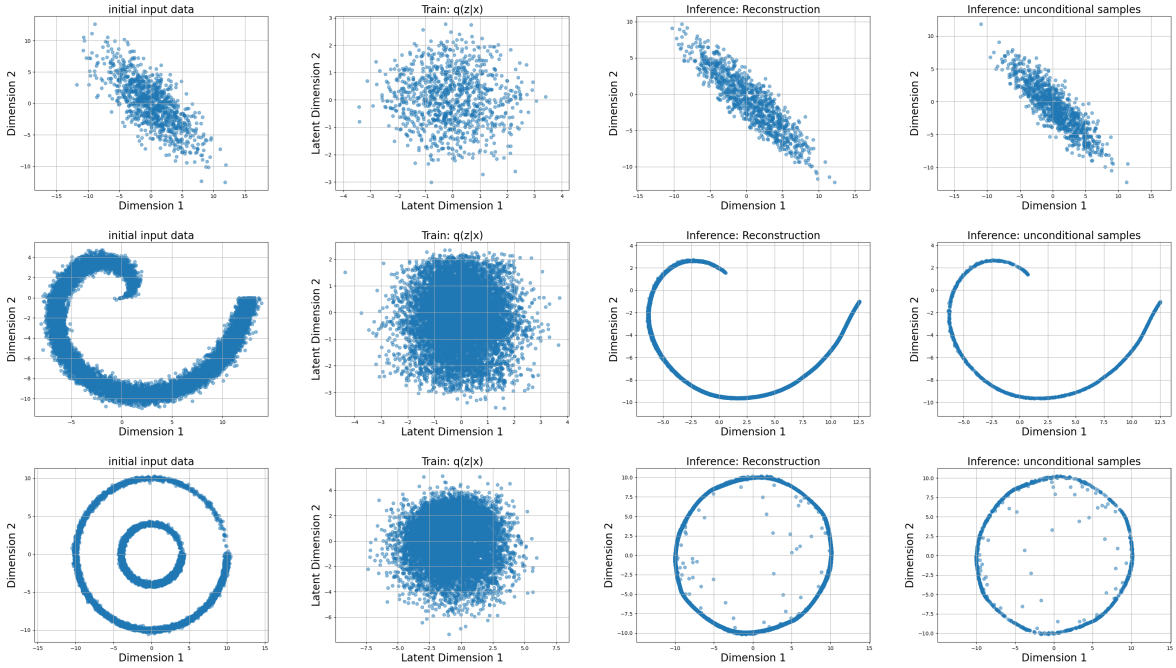
$$\log p_\theta(x) = \log \mathbb{E}_{z\sim q_\phi(z|x)}\left[\frac{p_\theta(x,z)}{q_\phi(z|x)}\right] \simeq \log \frac{1}{K}\sum_{i=1}^{K}\frac{p_\theta(x,z^{(i)})}{q_\phi(z^{(i)}|x)}, \text{ where } z^{(i)} \sim q_\phi(z|x). \tag{5.1}$$

This equation quantifies how well a variational autoencoder approximates the data distribution. Specifically, we encode each image to produce a mean vector (μ) and a variance vector (σ), sample $K = 1,000$ data points from the normal distribution defined by these parameters, decode these samples, and report the average approximate log-likelihood in Table 5.1.

The log-likelihoods regarding the anisotropic data are close, however, when it comes to the two-circles dataset, the BitNet model outperforms the baseline, even though it only was able



(a) Baseline Model



(b) BitNet Model

Figure 5.2: Comparison of the initial input data, latent space during reconstruction, output during reconstruction, and unconditional sampling using the BitNet and baseline models.

Model	Anisotropic	Spiral	two circles
Baseline	-5.276	-35.159	-30.079
BitNet (using QAT)	-4.804	-4.77	-8.533

Table 5.1: Estimated log-likelihood (higher is better) using both models on different synthetic datasets

to generate one circle during unconditional sampling. This is because the aggregated variational posterior for the baseline model does not match the Gaussian prior since the model only approximated the identity function. Regarding the spiral dataset: although the unconditional sampling outputs of both models look visually similar, the difference in log-likelihood can be attributed to the model's fit to the prior distribution: the BitNet model aligns better with the prior.

The log-likelihood results suggest that the limited weight expressivity of the BitNet model may act as a form of regularization. We hypothesize that this regularization results in a closer fit of the aggregated posterior to the prior, enabling BitNet to generate more coherent (unconditional) reconstructions despite having ternary weights.

Quantization Error and Weight Distribution To further explore the impact of the BitNet model's quantized architecture, we analyze the quantization error at specific intervals during training. We define the quantization error as the absolute difference between the unquantized FP32 weights and their quantized equivalents:

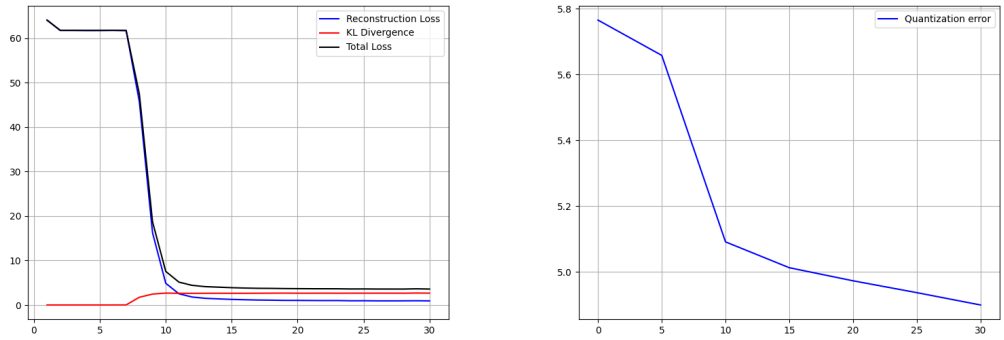
$$\text{qauntization error} = \frac{1}{N}\sum_{i=1}^{N}|w_i - Q(w_i)|, \tag{5.2}$$

where N is the number of weights, and Q the quantization function (described in Chapter 3).

We plot the quantization error and the loss for the spiral dataset in Fig. 5.3. As shown in Fig. 5.3, this error decreases consistently as training progresses, suggesting that the model adapts to the quantized weight structure. It is interesting to note that even though the loss stabilizes around the 10th epoch, quantization error continues to decline, indicating ongoing improvements to the weight quantization.

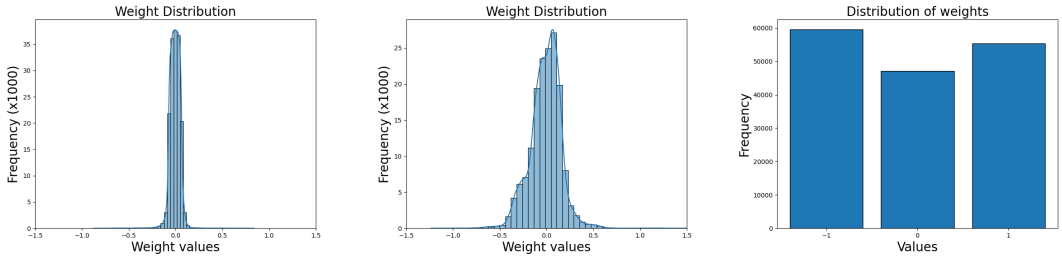
For the BitNet model, Fig. 5.4 visualizes the distribution of the weights at two key stages: after the end of the training phase (while the model still uses high-precision FP32 values for the weights) and during inference (when weights are quantized), as described in Chapter 3. The weights of the BitNet model exhibit a higher variance and are closer to the quantized values $\{-1, 0, 1\}$ compared to the baseline model, where weights tend to cluster near zero. During inference, as shown in Fig. 5.4 (c), the weights are fully quantized to the values {-1, 0, 1}. We argue that the BitLinear layer plays a crucial role in ensuring that the weight distribution is near these quantized values.

For all datasets tested in this section, the quantization error as well as the distribution of the weights follow a similar pattern.



(a) *(b)*

Figure 5.3: Training statistics for the BitNet model: (a) KL divergence, reconstruction loss, and total loss over epochs; (b) Quantization error over epochs



(a) *(b)* *(c)*

Figure 5.4: Comparing the distribution of the weights of the baseline and BitNet models during training and inference, where: (a) Baseline model using full precision; (b) BitNet model after training and before switching to inference mode; (c) BitNet model during inference

While the training loss stabilizes early, the quantization error continues to decrease. This suggests ongoing optimization of the quantized weights, ultimately leading to a weight distribution that better aligns with the target ternary values. Additionally, the BitNet model exhibits a higher weight variance compared to the baseline, which prevents the weights from collapsing to a single quantized value, encouraging a more balanced distribution across the quantization range.

Failure cases While both the baseline and BitNet VAE models effectively capture simpler geometric shapes, they encounter significant difficulties when generating more complex structures, such as mixtures of Gaussian distributions (Fig. 5.5) or multiple distinct shapes (e.g., two overlapping circles Fig. 5.2). These failure cases are particularly insightful, as they highlight the limitations of both models when it comes to handling data distributions that deviate from simple, structured patterns.

To circumvent these issues we used a decoder that not only outputs the mean μ but also the variance σ. This solved the problems for the baseline VAE but did not lead to any improvements in the BitNet VAE (Fig. 5.5).

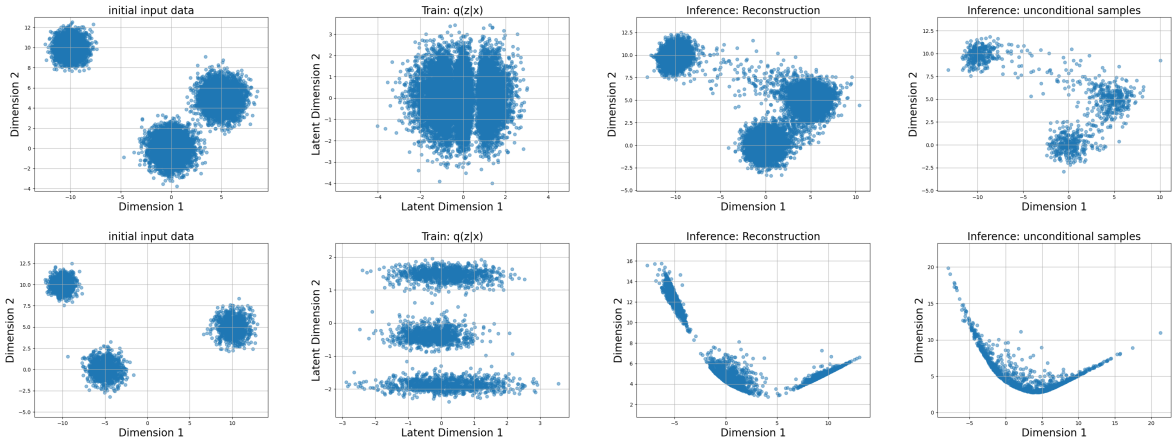


Figure 5.5: Visualization of the initial input data (mixture of Gaussians), latent space during reconstruction, the output during reconstruction, and the output during unconditional sampling using the baseline model in the first row and the BitNet model in the second one.

The mixture of Gaussian data highlights the limitations of the BitNet model when applied to synthetic datasets. Despite using a more expressive decoder that outputs both the mean and variance, the model struggles to capture the structure of more complex data distributions accurately.

5.3.2 MNIST Dataset

Motivation The MNIST dataset presents a more challenging and higher-dimensional task compared to the synthetic dataset used in earlier experiments. We want to determine whether the BitNet model can maintain high-quality image reconstruction, meaningful latent representation, and improved (memory) efficiency when applied to a larger, more realistic dataset.

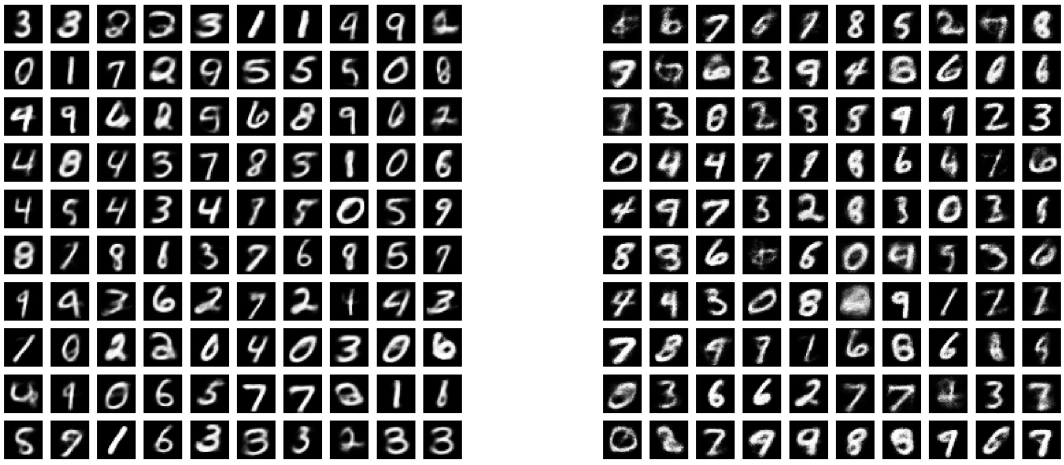
Approximate Log-Likelihood We calculate the approximate log-likelihood analogous to the previous section 5.3.1 and report the values in Table 5.2.

Despite the use of ternary weights, the BitNet model demonstrates competitive log-likelihood performance, indicating its ability to model the data distribution effectively. However, the baseline model surpasses BitNet in log-likelihood, suggesting a more accurate fit to the data distribution.

Precision To evaluate how well both VAEs capture the discriminative features of the data, we qualitatively and quantitatively assess the precision of generated images.

To estimate the precision for unconditionally generated images, we randomly sample 100 vectors from the prior $p(z)$ and pass them through the decoder, where Fig. 5.6 visualizes the generated images. The baseline model produces qualitatively better images, which is in alignment with the baseline's higher approximate log-likelihood that we reported earlier. The BitNet model is able to generate most digits but some of them are indistinct and blurry.

To calculate the precision for conditionally generated images, we utilize a pre-trained classifier for the MNIST dataset [Kabir et al., 2023], which achieves a 98% accuracy on the test set. We start by classifying the original images using this model and treat its output as the ground truth. Next, we reconstruct the images using the baseline or the BitNet model and classify the



(a) *(b)*

Figure 5.6: 100 unconditionally generated images: (a) baseline model; (b) BitNet

Model	log-likelihood	Precision (in%)
Baseline	-791.865	61.24
BitNet (using QAT)	-888.584	71.51

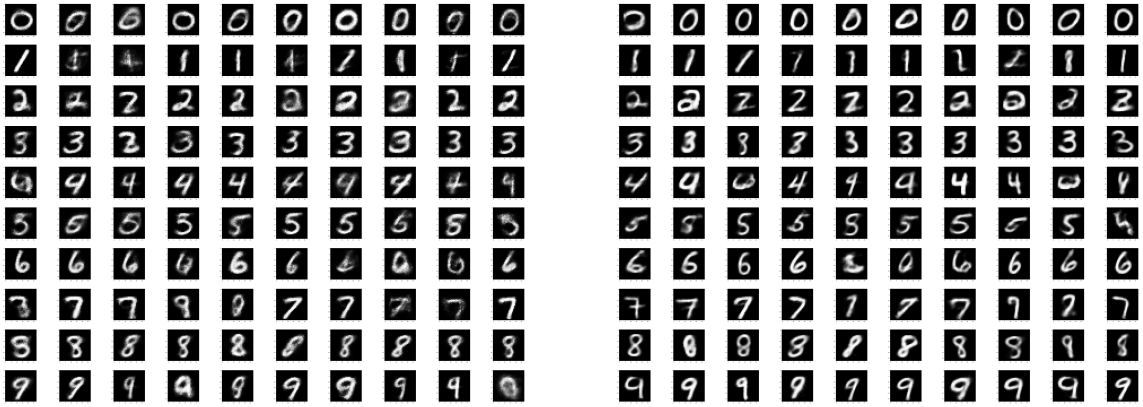
Table 5.2: Estimated log-likelihood (higher is better) and precision (higher is better) for the baseline and the BitNet model

reconstructions with the pre-trained classifier. The ratio of correctly classified reconstructed images is reported in Table 5.2.

Interestingly, despite the lower log-likelihood, the BitNet model achieves higher precision compared to the baseline model. We hypothesize that this is due to BitNet's larger latent space dimensionality, which we had specified by hyperparameter optimization when training the models, as described in Table B.2. We assume that such a higher capacity of the latent variable may improve reconstruction accuracy but lead to worse performance in the unconditional generation, as the latent space becomes more "sparse".

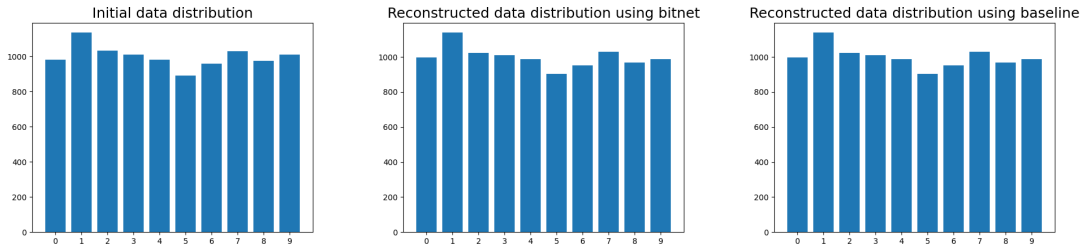
Recall Recall measures the diversity of generated images, indicating how well the model captures the full distribution of the data.

For conditionally generated images, we plot 10 samples from each class, as shown in Fig. 5.7. While most digits generated by the BitNet model demonstrate reasonable recall, the images are more indistinct and blurry compared to those from the baseline model. This suggests that although the BitNet model captures the broad structure of each digit, it struggles with fine details, leading to less sharp and clear outputs. Additionally, we reconstruct the test set and compare the frequency distributions of classified reconstructed images against the original test data in Fig. 5.8. The distribution of the frequencies of reconstructed digits for both VAEs seems to match that of the initial data distribution.



(a) *(b)*

Figure 5.7: Ten generated images for each digit: (a) using the BitNet model; (b) using the baseline model



(a) *(b)* *(c)*

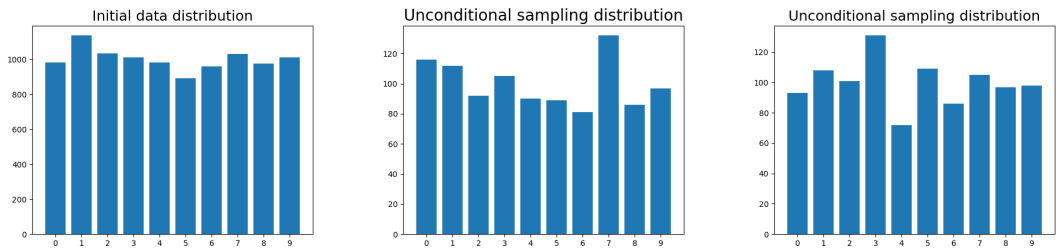
Figure 5.8: (a) Initial distribution of the data; (b) Frequency of each digit after reconstruction using the BitNet model; (c) Frequency of each digit after reconstruction using the baseline model

These results indicate that the BitNet model is capable of generalizing across classes, but the reduced clarity in the generated images points to limitations in the model's ability to capture fine-grained features.

For unconditionally generated images, we sample 1,000 data points from the prior, decode them, and classify the outputs as visualized in Fig. 5.9. The results indicate that the BitNet model over-represents certain digits, such as the digit "3", while under-representing others, such as the digit "4", compared to the original data. The baseline model shows a bias towards generating the digit "7". These biases indicate that while neither model perfectly replicates the true dataset distribution, the baseline model more closely approximates it. This suggests that the baseline model may capture more features and patterns in the data than the BitNet model, which exhibits greater bias due to its limited weight expressivity.

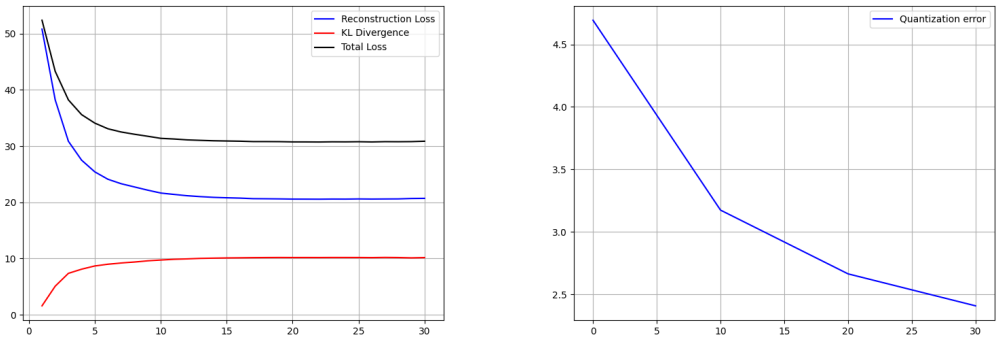
Quantization Error and Weight Distribution Analogous to the experiments from the synthetic data (Section 5.3.1), we analyze the weight distribution and the quantization error for the MNIST dataset.

We observe a similar behavior to synthetic data where the loss converges relatively quickly (around the 10th epoch) while the quantization error still decreases throughout the training process (Fig. 5.10). We also report the distribution of weights for both models in Fig. 5.11. Again, we observe a similar trend for the weights to have higher variance in the distribution for



(a) *(b)* *(c)*

Figure 5.9: (a) Initial distribution of the data; (b) Frequency of each digit for unconditionally generated images using the baseline model; (c) Frequency of each digit for unconditionally generated images using the BitNet model



(a) *(b)*

Figure 5.10: Training statistics for the BitNet model: (a) KL divergence, reconstruction loss and total loss over epochs; (b) Quantization error over epochs

BitNet compared to a narrower distribution for the baseline model centered around 0.

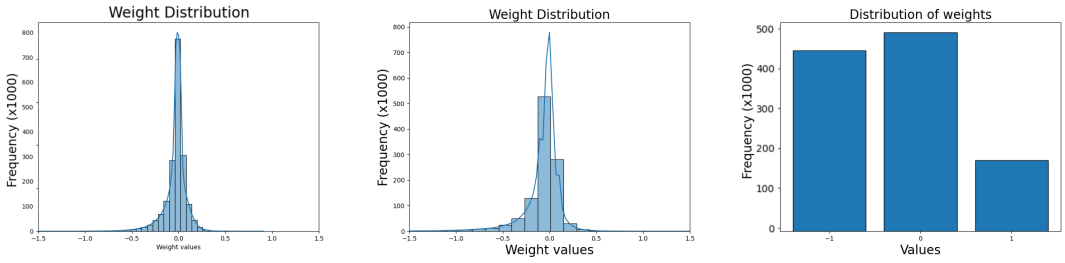
Interestingly, after quantization, the weight distribution is not uniform but remains close to the unquantized distribution. Compared to the synthetic data, where the quantized weights were more evenly spread, the MNIST dataset shows a higher proportion of -1s and 0s than 1s.

Memory improvements We evaluate the memory efficiency of the BitNet model by comparing the number of bits required to store the model. The baseline model requires 32,964,608 bits, while the BitNet model requires only 1,637,335.04 bits due to its quantized weights, representing a roughly 20-fold reduction in memory usage.

The BitNet model demonstrates a significant memory reduction compared to the baseline, requiring roughly **20 times** fewer bits for storing weights.

5.4 Results: Diffusion models

This section presents the experimental results obtained by applying both the baseline diffusion model and the BitNet diffusion model to synthetic data. The goal is to evaluate how quantization-aware training (QAT) impacts the model's ability to reconstruct and generate



(a) *(b)* *(c)*

Figure 5.11: Comparing the distribution of the weights of the baseline and BitNet models during training and inference, where: (a) Baseline model using full precision; (b) BitNet model after training and before switching to inference mode; (c) BitNet model during inference

synthetic data through the diffusion process. We focus on analyzing the trade-off between quantization (as measured by memory efficiency) and generative quality.

5.4.1 Synthetic data

To select the models for comparison, we start by a grid search for hyperparameters.

Hyperparameter search We investigate the impact of various hyperparameters, including learning rate, number of hidden layers, and hidden layer size, on the performance of BitNet. The figures 5.12, 5.13, and B.1 depict how changes in these hyperparameters affected the models' ability to generate accurate images. A low learning rate for example (10^{-5}) as well as a high learning rate (10^{-2}) led to the model not being able to accurately learn the distribution of the data. Also, the hidden layer's size played a major role in the quality of the generated images. Lower sizes (16 and 32) had blurry outputs while high sizes (512) did not seem to reconstruct the data accurately.

Similar experiments were performed using the original implementation without quantization (see Figs. B.2 and B.3). Both models exhibited strong performance with a similar range of learning rates. However, a key difference emerged in the optimal size of the hidden layers. While the baseline model produced good-quality images with hidden layer sizes of 16 and 32 neurons, the BitNet model required larger layers (ranging from 64 to 256 neurons) to achieve comparable results. This is likely due to the ternarization of the weights, as the BitNet model requires additional neurons to effectively represent the data and compensate for the reduced precision.

To achieve comparable results to the baseline model, the BitNet model requires a larger number of neurons per layer, 64 or more, whereas the baseline performs well with just 16 or 32 neurons. We argue that this increase in model capacity compensates for the reduced precision of ternary weights, allowing BitNet to maintain competitive performance. Note that the BitNet still has a significantly lower memory footprint than the baseline (see below).

Results We compare baseline and BitNet on synthetic data as shown in Fig. 5.14. The results of our experiments on synthetic data demonstrate that the BitNet model performs comparably to the baseline in generating synthetic images, albeit with some minor visual noise. As shown in Figure 5.13, the BitNet model was able to reconstruct images reasonably well, but with a slight increase in artifacts compared to the baseline.

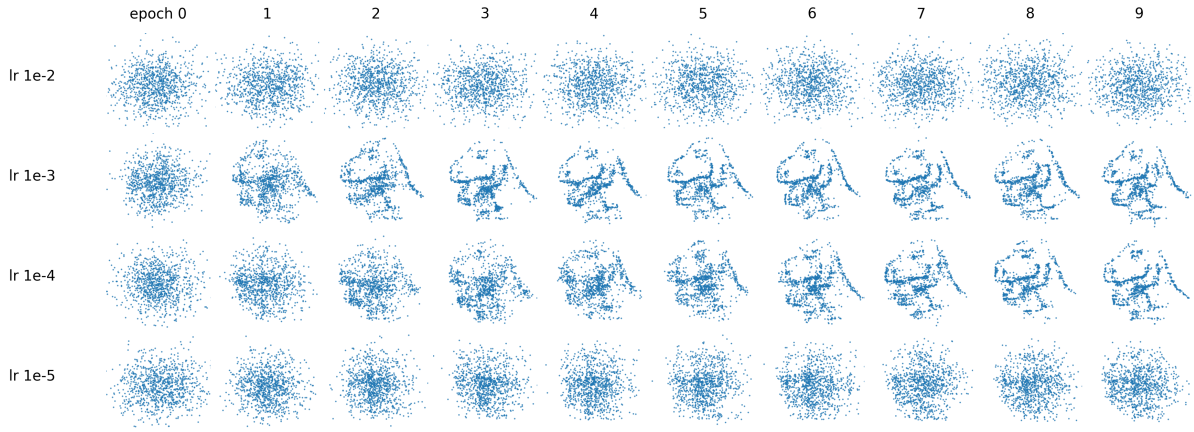


Figure 5.12: Effect of the learning rate on image reconstruction across epochs for the BitNet diffusion model.

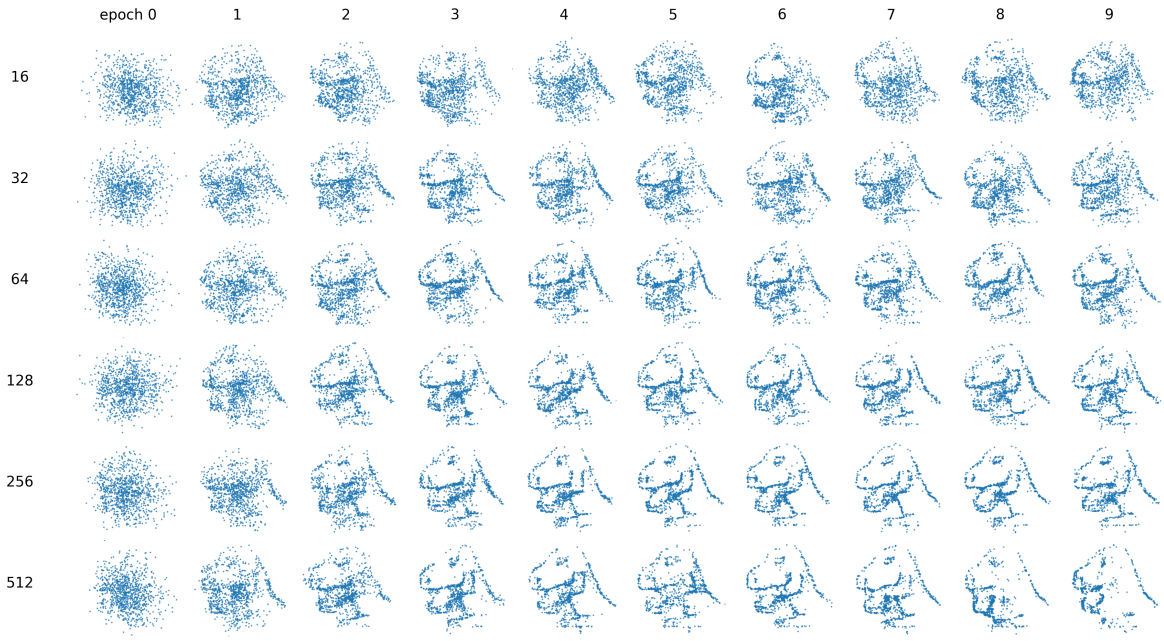


Figure 5.13: Effect of the hidden layer's size on image reconstruction across epochs for the BitNet diffusion model.

Despite using only ternary weights, the BitNet diffusion model is able to generate the synthetic data well, with a visual quality comparable to the baseline full-precision model. However, while the general structure of the synthetic images is well-reconstructed, finer details are less clear compared to the baseline.

Memory improvements In our experiments, we applied the BitLinear layer only within the MLP component of the diffusion model (the embeddings were not quantized). The baseline model required 3,178,496 bits to store the weights, whereas the BitNet model reduced this to 180,300.8 bits. This resulted in a significant memory reduction, approximately 17-fold compared to the baseline.

The application of the BitLinear layer in the diffusion model's MLP reduced memory usage by approximately **17 times** compared to the baseline model without a significant drop in the accuracy of the generated images.

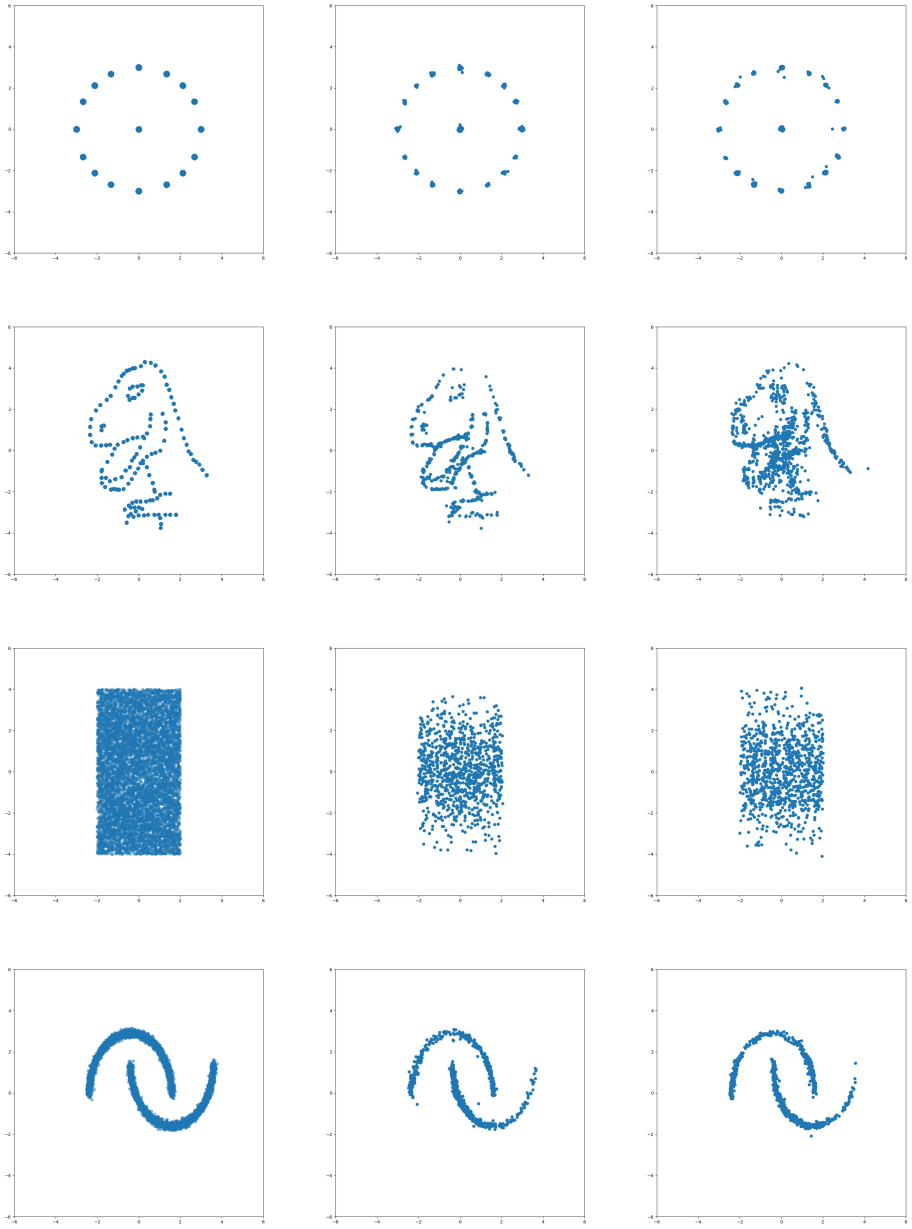


Figure 5.14: (From left to right) Original input image, generated image by the baseline model, and generated image using BitNet

Chapter 6

Conclusions

In this chapter, we will summarize our key findings and insights from the experiments, while also discussing the limitations of the BitNet model and potential directions for future work.

6.1 Summary and conclusions

The results of this thesis demonstrate that quantization-aware training (QAT) can effectively reduce the memory footprint of generative models, such as VAEs and diffusion models, without a dramatic loss of performance on simpler datasets. Our experiments confirm that QAT allows the BitNet models to adapt their weight structures towards ternary values {-1, 0, 1} during training, leading to significant memory savings while maintaining competitive performance regarding measures like log-likelihood and image reconstruction quality during inference.

In some cases, the performance of the quantized model improves the performance of the full-precision baseline, which we attribute to the regularization effect imposed by quantization, where lower model capacity through weights encoded with reduced precision can help to avoid overfitting. Nevertheless, the quantized model's performance is sometimes worse than the baseline, likely due to the reduced model capacity.

While the BitNet model successfully represents most of the data, it struggled with more complex patterns, such as producing two distinct circles or accurately modeling a mixture of Gaussians. Overall, it produces less detailed images compared to the baseline, particularly on more complex datasets.

Finally, we found interesting properties of quantized models. For example, the ongoing reduction in quantization error throughout training suggests that the models are still refining their quantized weights even after convergence in the loss function. This shows that QAT adapts well to the constraints imposed by quantization.

6.2 Future work and limitation

The current implementation of the BitNet model is built using conventional Python and PyTorch [Paszke et al., 2017]. Although QAT effectively reduces the memory footprint, the potential inference speedups from quantization have not yet been fully realized. As observed in similar work [Ma et al., 2024], true speedup benefits require the development or use of specialized (CUDA) kernels and hardware accelerators, which are not yet fully developed or widely available in existing ecosystems. Future work could focus on optimizing inference speed by leveraging specialized hardware designed for low-precision operations. This includes developing custom kernels for quantized operations or utilizing emerging hardware architectures that

support efficient inference with ternary weights.

Additionally, more research is needed to evaluate the capabilities of QAT to more complex generative tasks, such as handling multimodal distributions or real-world datasets with higher complexity. Another potential area for exploration is combining QAT with other model compression techniques, such as pruning or low-rank adaptation, to further reduce the model size and enhance efficiency without sacrificing performance.

Despite these limitations, this thesis provides a step toward developing more efficient generative models, highlighting the importance of QAT in the broader context of model compression and resource-constrained environments.

Bibliography

Score-based generative modeling through stochastic differential equations, 2021. URL https://arxiv.org/abs/2011.13456.

A. Aghajanyan, L. Zettlemoyer, and S. Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, 2020. URL https://arxiv.org/abs/2012.13255.

H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot. Ternary neural networks for resource-efficient ai applications. In *2017 international joint conference on neural networks (IJCNN)*, pages 2547–2554. IEEE, 2017.

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

H. Charlie. Diffusion models in generative chemistry for drug design. *medium.com*, Jun 2023. URL https://medium.com/@cch57/exploring-the-promise-of-generative-models-in-chemistry-an-introduction-to-diffusion-mode

H. Cheng, M. Zhang, and J. Q. Shi. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations. *arXiv preprint arXiv:2308.06767*, 2023.

Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks, 2020. URL https://arxiv.org/abs/1710.09282.

M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann,

I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. E. Tan, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Grattafiori, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Vaughan, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Franco, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Wyatt, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Ozgenel, F. Caggioni, F. Guzmán, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Thattai, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, I. Damlaj, I. Molybog, I. Tufanov, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Prasad, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Huang, K. Chawla, K. Lakhotia, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Tsimpoukelli, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. P. Laptev, N. Dong, N. Zhang, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Li, R. Hogan, R. Battey, R. Wang, R. Maheswari, R. Howes, R. Rinott, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay,

S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Kohler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Albiero, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wang, X. Wu, X. Wang, X. Xia, X. Wu, X. Gao, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Hao, Y. Qian, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, and Z. Zhao. The llama 3 herd of models, 2024. URL `https://arxiv.org/abs/2407.21783`.

A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016. URL `https://arxiv.org/abs/1510.00149`.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. URL `https://arxiv.org/abs/1503.02531`.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models, 2020. URL `https://arxiv.org/abs/2006.11239`.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Z. Huang and N. Wang. Like what you like: Knowledge distill via neuron selectivity transfer, 2017. URL `https://arxiv.org/abs/1707.01219`.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL `https://arxiv.org/abs/1712.05877`.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

H. M. D. Kabir, M. Abdar, A. Khosravi, S. M. J. Jalali, A. F. Atiya, S. Nahavandi, and D. Srinivasan. Spinalnet: Deep neural network with gradual input. *IEEE Transactions on Artificial Intelligence*, 4(5):1165–1177, Oct. 2023. ISSN 2691-4581. doi: 10.1109/tai.2022.3185179. URL `http://dx.doi.org/10.1109/TAI.2022.3185179`.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

D. P. Kingma, M. Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

C. Luo. Understanding diffusion models: A unified perspective, 2022. URL `https://arxiv.org/abs/2208.11970`.

S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

X. Ma, G. Fang, and X. Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

T. Pärnamaa. Tiny-diffusion, 2023. URL `https://github.com/tanelp/tiny-diffusion`. GitHub repository.

A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets, 2015. URL `https://arxiv.org/abs/1412.6550`.

F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9):3400–3413, 2020. doi: 10.1109/TNNLS.2019.2944481.

X. Shen, Z. Kong, C. Yang, Z. Han, L. Lu, P. Dong, C. Lyu, C.-h. Li, X. Guo, Z. Shu, et al. Edgeqat: Entropy and distribution guided quantization-aware training for the acceleration of lightweight llms on the edge. *arXiv preprint arXiv:2402.10787*, 2024.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

T. Spinner, J. Körner, O. Deussen, and J. Görtler. Towards an interpretable latent space. *tspinner.de*, 2018. URL `https://tspinner.de/towards-an-interpretable-latent-space/`.

S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Z. Wan, X. Wang, C. Liu, S. Alam, Y. Zheng, Z. Qu, S. Yan, Y. Zhu, Q. Zhang, M. Chowdhury, et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 1, 2023.

H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei. Bitnet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

L. Weng. What are diffusion models? *lilianweng.github.io*, Jul 2021. URL `https://lilianweng.github.io/posts/2021-07-11-diffusion-models/`.

S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek. Deepcabac: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):700–714, 2020. doi: 10.1109/JSTSP.2020.2969554.

H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.

L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang. Diffusion models: A comprehensive survey of methods and applications, 2024. URL `https://arxiv.org/abs/2209.00796`.

P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi, and J. Xin. Understanding straight-through estimator in training activation quantized neural nets. *arXiv preprint arXiv:1903.05662*, 2019.

B. Zhang and R. Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Appendices

Appendix A

Mathematical Derivations

A.1 Derivation of the ELBO

The goal is to find parameters θ under which the likelihood of the observed data is maximal. We therefore want to maximize the log-likelihood defined as:

$$\underset{\theta}{\operatorname{argmax}} \log p_\theta(x). \tag{A.1}$$

Since directly maximizing the log-likelihood is infeasible, we instead seek a tractable lower bound for $p_\theta(x)$ and maximize it instead [Kingma et al., 2019]:

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x)] && \text{(A.2)}\\
&= \int q_\phi(z|x) \log p_\theta(x) dz && \text{(A.3)}\\
&= \int q_\phi(z|x) \log \frac{p_\theta(x,z)}{p_\theta(z|x)} dz && \text{(A.4)}\\
&= \int q_\phi(z|x) \log \left(\frac{p_\theta(x,z)}{p_\theta(z|x)} \cdot \frac{q_\phi(z|x)}{q_\phi(z|x)} \right) dz && \text{(A.5)}\\
&= \int q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} dz + \int q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x)} dz && \text{(A.6)}\\
&= \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)} \right] + D_{KL}(q_\phi(z|x) \parallel p_\theta(z|x)). && \text{(A.7)}
\end{aligned}$$

Since the KL Divergence is always positive, it holds that:

$$\log p_\theta(x) \geq \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)} \right]}_{\text{this is defined as the ELBO}}. \tag{A.8}$$

To arrive at the expression 4.3 :

$$\begin{aligned}
\mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)} \right] &= \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log \frac{p_\theta(x|z) p_\theta(z)}{q_\phi(z|x)} \right] && \text{(A.9)}\\
&= \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z) \right] + \int q_\phi(z|x) \log \frac{p_\theta(z)}{q_\phi(z|x)} dz && \text{(A.10)}\\
&= \mathbb{E}_{z\sim q_\phi(z|x)} \left[\log p_\theta(x|z) \right] - D_{KL}(q_\phi(z|x) \parallel p_\theta(z)) && \text{(A.11)}
\end{aligned}$$

A.2 Closed formula for the KL Divergence term

The KL Divergence between $p(x)$ and $q(x)$ is given by:

$$D_{KL}(p(x) \parallel q(x)) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right), \tag{A.12}$$

where p and q are assumed Gaussian distributions with means μ_1,μ_2 and standard deviations σ_1, σ_2 given by:

$$p(x) = \frac{1}{\sqrt{(2\pi)^N |\sigma_1|}} \exp\left(-\frac{1}{2}(x-\mu_1)^T \sigma_1^{-1}(x-\mu_1)\right), \tag{A.13}$$

$$q(x) = \frac{1}{\sqrt{2\pi}^N |\sigma_2|} \exp\left(-\frac{1}{2}(x-\mu_2)^T \sigma_2^{-1}(x-\mu_2)\right), \tag{A.14}$$

Substituting these in (A.12) and solving it will lead to:

$$D_{KL}(p(x) \parallel q(x)) = \frac{1}{2}\left[\log\frac{|\sigma_2|}{|\sigma_1|} + \operatorname{tr}(\sigma_2^{-1}\sigma_1) + (\mu_1-\mu_2)^T\sigma_2^{-1}(\mu_1-\mu_2) - N\right]. \tag{A.15}$$

If $\sigma_2 = I$, and $\mu_2 = 0$, i.e., $q(x) = \mathcal{N}(0, I)$, it holds that:

$$\begin{aligned}
D_{KL}(p(x) \parallel q(x)) &= \frac{1}{2}\left[\log\frac{1}{|\sigma_1|} + \operatorname{tr}(\sigma_1) + (\mu_1)^T(\mu_1) - N\right] && \text{(A.16)}\\
&= \frac{1}{2}\left[-\log|\sigma_1| + \operatorname{tr}(\sigma_1) + \mu_1^2 - N\right] && \text{(A.17)}\\
&= \frac{1}{2}\left[-\log\prod_N \sigma_1 + \sum_N \sigma_1 + \sum_N \mu_1^2 - \sum_N 1\right] && \text{(A.18)}\\
&= \frac{1}{2}\left[-\sum_N \log(\sigma_1) + \sum_N \sigma_1 + \sum_N \mu_1^2 - \sum_N 1\right] && \text{(A.19)}\\
&= \frac{1}{2}\sum_N \left[-\log(\sigma_1) + \sigma_1 + \mu_1^2 - 1\right]. && \text{(A.20)}
\end{aligned}$$

Thus, the KL divergence term can be efficiently computed for the loss function.

A.3 MSE as the reconstruction loss

To estimate the expectation $\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$, we sample M latent vectors $z_1, \ldots, z_M$ from the variational posterior distribution $q_\phi(z \mid x)$, and compute the average of the log-likelihoods:

$$\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] \approx \frac{1}{M}\sum_{i=1}^{M} \log p_\theta(x \mid z_i).$$

We assume that $p_\theta(x|z_i)$ is normally distributed with variance I,

$$p_\theta(x|z_i) = \mathcal{N}(x, f_\theta(z_i), I), \tag{A.21}$$

where $f_\theta(z_i)$ is the output of the decoder.

$$\log p_\theta(x|z_i) = \log\left(\frac{1}{(2\pi)^{d/2}}\exp\left(-\frac{1}{2}\|x - f_\theta(z_i)\|^2\right)\right). \tag{A.22}$$

$$= \log(\frac{1}{(2\pi)^{d/2}}) - \frac{1}{2}\|x - f_\theta(z_i)\|^2. \tag{A.23}$$

Therefore, maximizing the log-likelihood is equivalent to minimizing the mean squared error.

A.4 NLL as reconstruction loss

Similar to Section Section A.3, we can express the conditional likelihood $p_\theta(x|z_i)$ as a normal distribution:

$$p_\theta(x|z_i) = \mathcal{N}(x, \mu_{\text{recon}}, \Sigma_{\text{recon}}), \tag{A.24}$$

where μ_{recon} and Σ_{recon} represent the output of the decoder.

Consequently, the negative log-likelihood (NLL) is defined as:

$$-\log p_\theta(x|z_i) = \frac{d}{2}\log(2\pi) + \frac{1}{2}\sum_{i=1}^{d}\log\sigma^2_{\text{recon},i} + \frac{1}{2}\sum_{i=1}^{d}\frac{(x_i - \mu_{\text{recon},i})^2}{\sigma^2_{\text{recon},i}}. \tag{A.25}$$

This is equivalent to maximizing the following form, since $\frac{d}{2}\log(2\pi)$ is a constant:

$$\frac{1}{2}\sum_{i=1}^{d}\left(\log\sigma^2_{\text{recon},i} + \frac{(x_i - \mu_{\text{recon},i})^2}{\sigma^2_{\text{recon},i}}\right). \tag{A.26}$$

In our experiments, we use this simplified expression as the reconstruction loss.

A.5 Derivation of $q(x_t|x_0)$

In this section, we derive the expression for $q(x_t|x_0)$, which represents the distribution of the data at an intermediate step t conditioned on the initial data x_0. This derivation is achieved by iteratively applying the recursive definition of x_t.

Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, [Luo, 2022]:

$$
\begin{aligned}
x_t &\sim q(x_t|x_{t-1}) && \text{(A.27)}\\
&= \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\epsilon^*_{t-1} && \text{(A.28)}\\
&= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}}\epsilon^*_{t-2}\right) + \sqrt{1-\alpha_t}\epsilon^*_{t-1} && \text{(A.29)}\\
&= \sqrt{\alpha_t\alpha_{t-1}} x_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\epsilon^*_{t-2} + \sqrt{1-\alpha_t}\epsilon^*_{t-1} && \text{(A.30)}\\
&= \sqrt{\alpha_t\alpha_{t-1}} x_{t-2} + \sqrt{\sqrt{\alpha_t - \alpha_t\alpha^2_{t-1}} + \sqrt{1-\alpha_t}^2}\epsilon_{t-2} && \text{(A.31)}\\
&= \sqrt{\alpha_t\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon_{t-2} && \text{(A.32)}\\
&= \cdots && \text{(A.33)}\\
&= \sqrt{\prod_{i=1}^{t}\alpha_i} x_0 + \sqrt{1-\prod_{i=1}^{t}\alpha_i}\epsilon_0 && \text{(A.34)}\\
&= \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_0 && \text{(A.35)}\\
&\sim \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t)\mathbf{I}). && \text{(A.36)}
\end{aligned}
$$

A.6 Variational lower bound for diffusion models

This section focuses on deriving the variational lower bound for diffusion models, similar to the approach used in Variational Autoencoders (VAEs). The goal is to find a tractable lower bound for the log-likelihood, $\log p_\theta(\mathbf{x}_0)$, which can then be maximized during model training (this is equivalent to finding an upper bound for $-\log p_\theta(\mathbf{x}_0)$and minimizing it).

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{KL}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0))\\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right]\\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right]\\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right].
\end{aligned}
$$

We define $L_{VLB} = \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right]$, [Weng, 2021].

Appendix B

Additional Visualizations

Hyperparameter	Value
Encoder layers	3 fully connected layers (200 neurons each)
Decoder layers	3 fully connected layers (200 neurons each)
Activation function	ReLU / Sigmoid
Learning rate	0.001
Batch size	64
Latent space dimension	2
Optimizer	Adam (default parameters)
Epochs	30

Table B.1: Hyperparameters used for training the VAEs on the synthetic dataset. (Same for both the baseline and the quantized model)

Hyperparameter	Value
Encoder layers	2 fully connected layers (512 and 256 neurons)
Decoder layers	2 fully connected layers (256 and 512 neurons)
Activation function	ReLU and Sigmoid for the last layer
Learning rate	0.001
Batch size	64
Latent space dimension	8 for the baseline and 32 for BitNet
Optimizer	Adam (default parameters)
Epochs	30

Table B.2: Hyperparameters used for training the VAEs on the MNIST dataset.

Component	Description
Positional Embeddings	3 sinusoidal embeddings (time, x, y)
Input dimension	384
Hidden layers	4 fully connected layers
Hidden layer dimensions	128 neurons per hidden layer
Activation function	GELU
Output dimension	2
Optimizer	AdamW

Table B.3: Model architecture and hyperparameters for the diffusion model.

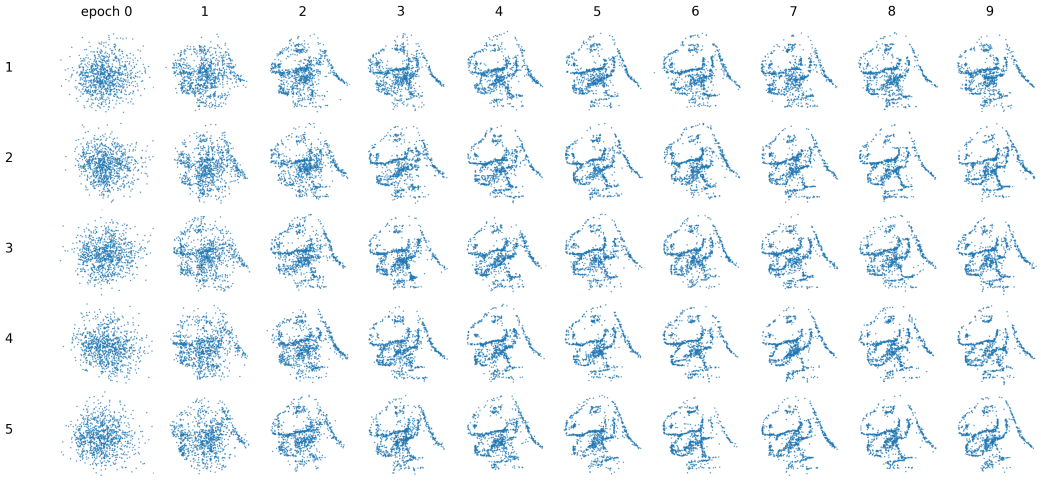


Figure B.1: Effect of the number of hidden layers on image reconstruction across epochs for the BitNet diffusion model.

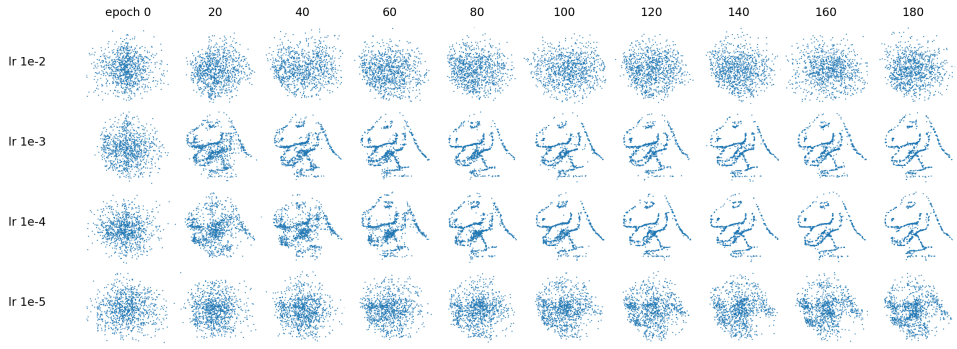


Figure B.2: Effect of the learning rate on image reconstruction across epochs for the baseline diffusion model [Pärnamaa, 2023].

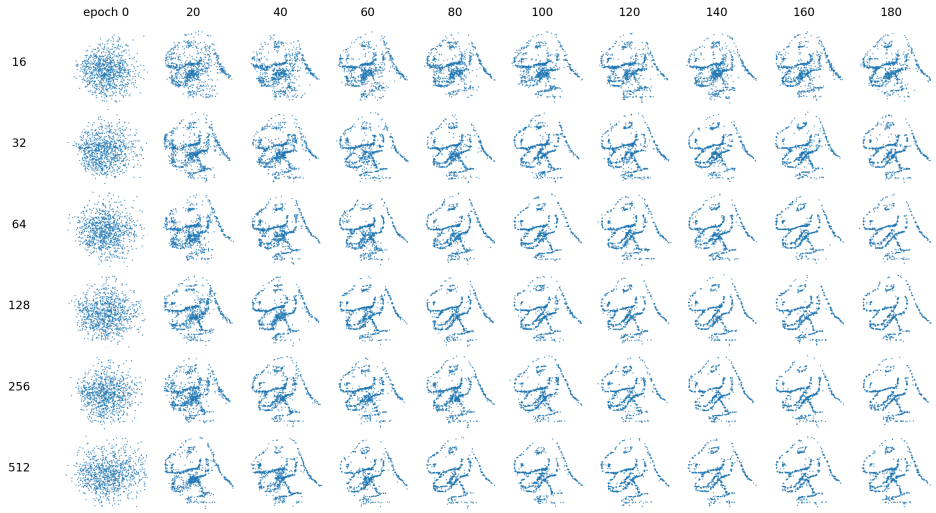


Figure B.3: Effect of the number of the size of hidden layers on image reconstruction across epochs for the baseline diffusion model [Pärnamaa, 2023].

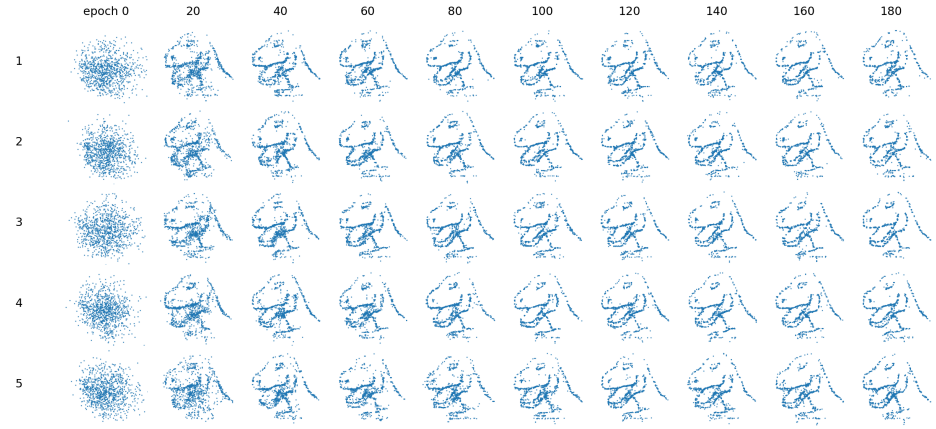


Figure B.4: Effect of the number of hidden layers on image reconstruction across epochs for the baseline diffusion model [Pärnamaa, 2023].