Low-Bit Power-of-Two Quantization for Large Language Models

Xinyu Wang

School of Computer Science

McGill University, Montreal

November, 2024

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

This thesis introduces a Power-of-Two (PoT) quantization method designed for efficient low-bit quantization of large language models (LLMs), targeting configurations as low as 2 or 3 bits. As LLMs expand in scale, the computational and memory demands required for deployment increase, driving the need for effective quantization strategies. While traditional quantization methods perform well at higher bit levels (e.g., 4 bits), their accuracy and efficiency diminish significantly at more extreme low-bit settings. Our proposed two-step PoT quantization approach addresses these limitations by focusing on maintaining high model fidelity while achieving substantial reductions in memory and compute requirements.

The method begins with a data-agnostic scale initialization step to set an optimized scaling factor without relying on calibration data. This is followed by a data-dependent fine-tuning step that uses a learnable parameter to adjust the scale based on calibration data, minimizing the alignment error between the original and quantized model activations. Unlike uniform quantization methods, which typically require zero-point offsets, our PoT quantization employs symmetric scaling, simplifying the quantization process while maintaining the performance.

We validate our method through extensive experiments on LLaMA model families, including LLaMA1 and LLaMA2 models with up to 30 billion parameters, achieving state-of-the-art perplexity on benchmarks such as WikiText-2, C4, and PTB datasets. While traditional methods excel in 4-bit quantization, our results demonstrate that PoT quantization is particularly effective at 3-bit and 2-bit levels, where traditional methods struggle. Furthermore,

our approach's dequantization scheme yields a modest speedup, highlighting its computational efficiency as an additional benefit.

In conclusion, this work shows that PoT quantization, with the proposed enhancements, provides an effective solution for deploying LLMs under stringent resource constraints, balancing accuracy and performance for real-world applications on resource-limited hardware.

Abrégé

Cette thèse propose une méthode de quantification en puissance de deux (PoT) destinée à la quantification efficace à faible nombre de bits des grands modèles de langage (LLMs), avec des configurations aussi basses que 2 ou 3 bits. À mesure que les LLMs croissent en taille, les exigences en calcul et en mémoire pour leur déploiement s'intensifient, accentuant ainsi le besoin de stratégies de quantification adaptées. Bien que les méthodes de quantification traditionnelles soient performantes à des niveaux de bits plus élevés (par exemple, 4 bits), leur précision et leur efficacité diminuent considérablement dans des configurations extrêmes à faible nombre de bits. Notre approche de quantification PoT en deux étapes répond à ces limitations en se concentrant sur le maintien d'une haute fidélité du modèle tout en réduisant significativement les besoins en mémoire et en calcul.

La méthode débute par une étape d'initialisation de l'échelle sans données, qui définit un facteur de mise à l'échelle optimisé sans recourir à des données de calibration. Cette étape est suivie d'une étape de réglage fin dépendant des données, qui utilise un paramètre apprenable pour ajuster l'échelle en fonction des données de calibration, minimisant l'erreur d'alignement entre les activations du modèle original et celles du modèle quantifié. Contrairement aux méthodes de quantification uniforme, qui nécessitent généralement des décalages de point zéro, notre quantification PoT utilise une mise à l'échelle symétrique, simplifiant le processus de quantification tout en maintenant les performances.

Nous validons notre méthode à travers des expériences approfondies sur les familles de modèles LLaMA, y compris les modèles LLaMA1 et LLaMA2 allant jusqu'à 30 milliards de paramètres, atteignant une perplexité de pointe sur des ensembles de données de référence

tels que WikiText-2, C4 et PTB. Bien que les méthodes traditionnelles excellent dans la quantification en 4 bits, nos résultats montrent que la quantification PoT est particulièrement efficace à des niveaux de 3 et 2 bits, où les méthodes traditionnelles rencontrent des difficultés. De plus, notre schéma de déquantification apporte une accélération modeste, soulignant son efficacité computationnelle comme un avantage supplémentaire.

En conclusion, ce travail montre que la quantification PoT, avec les améliorations proposées, offre une solution efficace pour le déploiement des LLMs dans des environnements à ressources limitées, équilibrant précision et performance pour des applications réelles sur du matériel contraint en ressources.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Prof. Xiao-wen Chang, for granting me the opportunity to study under his guidance and for his unwavering support both in academia and in my personal life. I still fondly recall the day I decided to study with you at the BBQ party—it was a pivotal moment in my journey.

I am deeply thankful to my research colleagues, Xinlin Li, Vahid Partovi Nia, Yufei Cui, Adrian Zhao, and Yu Gu, who introduced me to the field of language model quantization and guided me along the way. Working together has been an invaluable experience.

I would also like to express my sincere thanks to my former girlfriend, Yuhe Fan, who was a tremendous source of support in my life, encouraging me to confront challenges with strength and resilience. Although we have parted ways, I genuinely wish her a healthy and joyful life.

Finally, to my family and Miss Shurui Deng, whom I consider as my second mother, thank you for your unconditional love and support. Your encouragement means everything to me. I will continue to strive for my PhD with the hope of making you proud and becoming a better person—one with wisdom, courage, responsibility, and a strong sense of duty.

Table of Contents

Abstract 1

Abrégé 3

Acknowledgements 5

List of Figures 10

List of Tables 12

List of Abbreviations 13

1 Introduction **1**

1.1 Problem Statement 1

1.1.1 Increasing Computational Demands of large language models (LLM) 1

1.1.2 Quantization: Overview and Evolution 2

1.1.3 Challenges of Weight-activate Quantization (WAT) and the Rationale for Weight-only Quantization (WOQ) 3

1.1.4 Challenges of Quantization-aware Training (QAT) and the Feasibility of Post-training Quantization (PTQ) 4

1.1.5 Summary: Need for Low-Bit Weight-only Quantization (WOQ) with Efficient Dequantization 5

1.2 Literature Review 6

1.2.1 Quantization Techniques in Deep Learning 6

1.2.2 PTQ Methods in LLM 7

1.2.3 Dequantization Challenges and Power-of-Two Quantization for Efficient Inference 9

1.3 Goal, Contribution, and Organization 10
1.3.1 Objective 10
1.3.2 Two-Step PoT Quantization and Dequantization Approach 11
1.3.3 Advantages of the Proposed Approach 11
1.4 Organization of the Thesis 12
1.5 Notation 12

2 Background and Quantization Techniques 14
2.1 LLM Architecture and Layer-wise Computation 15
2.1.1 Matrix Representations in LLaMA Models 15
2.2 Introduction to Quantization 18
2.2.1 Rounding Techniques 19
2.2.2 Symmetric and Non-Symmetric Quantization 21
2.2.3 Uniform and Non-Uniform Quantization 24
2.2.4 Single-Precision vs. Mixed-Precision Quantization 27
2.2.5 Quantization Training Paradigms: QAT and PTQ 28
2.2.6 Weight-Activation Quantization vs. Weight-Only Quantization 30
2.3 State-of-the-Art PTQ Weight-Only Quantization Methods 32
2.3.1 OBQ and GPTQ 32
2.3.2 AWQ: Activation-Aware Weight Quantization 36
2.3.3 OmniQuant: Block-Wise Quantization with Learnable Weight Clipping 40

3 Two Step Power-of-Two Quantization Method 43
3.1 Assessment of Low-Bit Quantization Methods for LLaMa Models 44
3.1.1 Quantization Performance Summary for LLaMa Models 44
3.1.2 Need for Efficient Low-Bit Quantization with Fast Dequantization . . 46
3.2 Motivation for Power-of-Two Quantization in LLMs 46
3.2.1 Alignment with Weight Distribution in LLMs 47
3.2.2 Efficient Dequantization for Inference Acceleration 47

3.2.3 Conclusion 49
3.3 Proposed method 49
3.3.1 Explore and Preliminaries 49
3.3.2 Proposed Two-Step Quantization Method 54
3.3.3 Efficient Dequantization for Power-of-Two Quantized Weights 61

4 Experiments and Results **64**
4.1 Experimental Settings 64
4.2 Perplexity Calculation 65
4.3 Baselines 65
4.4 Quantization Results on PoT and Uniform Quantization Baselines 66
4.4.1 WikiText-2 Quantization Comparison 66
4.4.2 C4 and PTB Quantization Comparison 67
4.5 Ablation Studies 68
4.5.1 Efficiency of Step 2 (Section 3.3.2) : Epoch-Wise Perplexity and Loss Reduction 69
4.6 Quantization Efficiency 70
4.7 Dequantization Efficiency 71
4.8 Conclusion of Experiments 71

5 Conclusion and Future Work **72**
5.1 Conclusion 72
5.2 Limitation 73
5.3 Future Work 73

List of Figures

2.1 Illustration of symmetric quantization (top) and non-symmetric quantization (bottom). In symmetric quantization, the quantization range is centered around a zero point (e.g., $[-127, 127]$ for an 8-bit symmetric range), while in non-symmetric quantization, the range is shifted to better accommodate an unbalanced data distribution. 22

2.2 Comparison between uniform quantization (left) and non-uniform quantization (right). Real values in the continuous domain r are mapped into discrete, lower precision values in the quantized domain Q, which are marked with the orange bullets. Note that the distances between the quantized values (quantization levels) are the same in uniform quantization, whereas they can vary in non-uniform quantization. 24

2.3 Weight distribution for LLaMa1 7B, illustrating the typical dense mid-range and sparse extreme values found in neural network weights. 27

2.4 Comparison between Quantization-Aware Training (QAT, a) and Post-Training Quantization (PTQ, b). In QAT (a), a pre-trained model undergoes quantization followed by fine-tuning with training data to adjust parameters and mitigate accuracy degradation. In PTQ (b), a pre-trained model is calibrated using a small subset of data (calibration data) to compute the clipping ranges and scaling factors without additional fine-tuning. 28

2.5 Inference flow for (a) Weight-Only Quantization and (b) Weight-Activation Quantization. 32

3.1 Illustrations of LLM weight distribution and power-of-two quantization alignment. Power-of-two quantization aligns closely with LLM weight characteristics, especially in the low-bit setting, enhancing accuracy. 48
3.2 Overview of the proposed two-step quantization method. The first step involves data-agnostic scale initialization, while the second step refines the scaling using activation data from a calibration set. 54
3.3 Error curves over 200 grid points in the range [0,2] for both uniform and power-of-two scaling. Power-of-two scale results demonstrate less smooth loss curves compared to uniform scaling. 56
3.4 Distribution of optimal scaling values across 200 grid points in the range [0,2] for both uniform and power-of-two scaling. The results show that optimal power-of-two scaling values are widely distributed and not centered around 1, deviating from the naive RTN scale. 57
3.5 Multiplication between FP16 scale and power-of-two weights for a 3-bit example, implemented through efficient bit manipulation and fixed-point integer addition for accelerated PoT dequantization. 63

List of Tables

2.1 Summary of symmetric and non-symmetric quantization equations, with notation for low-bit and high-bit components. 23

3.1 WikiText-2 perplexity of 2-, 3-, and 4-bit quantized LLaMa family models (LLaMa-1 and LLaMa-2) using the studied quantization methods. Results are based on group-wise quantization with a group size of 128. Lower perplexity values indicate better model performance, and the table highlights the performance of Round-to-Nearest (RTN), GPTQ, AWQ, and OmniQuant across different bit levels. 44

3.2 Performance in perplexity comparison of various quantization methods on different LLaMa models and sizes, with a group size of 128. 50

4.1 Performance comparison of various quantization methods adapted to power-of-two (PoT) format on different LLaMa models and sizes, with a group size of 128. Our PoT method shows superior results, overcoming the limitations of applying PoT to existing uniform quantization methods. 67

4.2 WikiText-2 perplexity of 2- and 3-bit quantized LLaMa family using uniform quantization for baseline methods and PoT quantization for our method. . . 67

4.3 Perplexity comparison across LLaMa models at various bit levels on C4. . . . 68

4.4 PTB Perplexity results for different LLaMa models using OMNI (g128) and PoT (g64) configurations. 68

4.5 Ablation study on perplexity for different configurations of the PoT method for LLaMA models. 69
4.6 WikiText-2 Perplexity and Loss of LLaMA 13B model during Step 2(Section 3.3.2) fine-tuning. 70
4.7 LLaMA 4-bit Quantization Time 70
4.8 Dequantization efficiency measured in warp cycles on NVIDIA Tesla V100 and RTX 4090 GPUs. 71

List of Abbreviations

FP16 16-bit floating-point.

GEMM General Matrix Multiplications.

GPT generative pre-trained transformer.

HBM High-bandwidth Memory.

LLaMA Large Language Model Meta AI.

LLM large language models.

NLP natural language processing.

NNs neural networks.

PoT Power-of-Two.

PTQ Post-training Quantization.

QAT Quantization-aware Training.

RTN round to nearest.

SRAM Static Random-access Memory.

WAT Weight-activate Quantization.

WOQ Weight-only Quantization.

Chapter 1

Introduction

1.1 Problem Statement

1.1.1 Increasing Computational Demands of large language models (LLM)

In recent years, the development of large language models (LLM) has transformed the field of natural language processing (NLP). These models, which are trained to predict and generate text, excel at various language-related tasks such as translation, summarization, and question answering. Among the most impactful advancements has been the development of models based on the Transformer architecture [48], a deep learning structure introduced by Vaswani et al. in 2017, which has become the backbone for many state-of-the-art NLP models.

A prominent subclass of Transformer models is the generative pre-trained transformer (GPT), which is pre-trained on vast amounts of text data to capture general language patterns. Notable examples of GPT models include OpenAI's GPT-2 and GPT-3 [3, 37], and Meta AI's OPT [58]. These models belong to a family of LLMs that have demonstrated remarkable abilities in generating coherent and contextually relevant text based on an initial prompt, with applications ranging from essay generation to basic conversations.

Despite their powerful performance, deploying large-scale models like GPT-3, which has 175 billion parameters, presents significant computational challenges. Training such models can require tens to hundreds of GPU years [58]. Even after training, inference—using the model to generate text or answer queries—remains computationally expensive. For example, GPT-3's parameters occupy approximately 350 GB in 16-bit floating-point (FP16) format [3], far exceeding the memory capacity of standard high-end GPUs (typically 16–80 GB). This necessitates specialized hardware setups with multiple GPUs working in parallel, making deployment costly and complex.

The high storage and computation demands limit the practical deployment of large language models. Each pass of data through the model (known as a forward pass) involves substantial matrix calculations, especially in the attention and feedforward layers that form the core of the Transformer architecture. For real-world applications, these models face significant constraints in terms of latency (speed), energy consumption, and cost–especially when deployed in resource-constrained environments, such as edge devices or time-sensitive cloud services.

As large language models (LLM)s continue to grow in size, the need for efficient methods to compress and deploy them without sacrificing accuracy becomes critical. This need drives the exploration of quantization techniques, which reduce the computational and storage requirements of these models by representing their parameters in fewer bits, thereby enabling the deployment of large-scale language models in a wider range of settings.

1.1.2 Quantization: Overview and Evolution

Quantization is a process that maps input values from a large (often continuous) set to a smaller (often finite) set, thereby reducing the bit precision of numerical representations. Historically, quantization methods like rounding and truncation were introduced to approximate continuous quantities and manage numerical stability in early computational tasks. Traces of quantization can be seen as far back as the early 1800s, in methods related to least-squares

approximation and large-scale data analysis [44], and as early as 1867, when discretization was used to approximate integrals [39].

In modern applications, quantization has become fundamental to optimizing computational efficiency in neural networks (NNs). Early NNs were typically smaller, but as models grew more complex and over-parameterized, the need for more efficient representations became clear. Quantization gained traction in deep learning for reducing model size and accelerating inference, with initial applications in convolutional neural networks for tasks such as image classification and object detection.

The recent advent of large language models (LLM) has further elevated the importance of quantization. As these models scale in size and complexity, quantization is now applied to convert high-precision weights into low-bit representations, compressing model sizes and enabling deployment on resource-constrained devices. By approximating high-precision weights and activations with lower-precision values, quantization helps decrease memory requirements, reduce computational costs, and improve inference speed, making LLM viable for real-time applications on edge devices and in cloud environments with limited computational resources.

1.1.3 Challenges of Weight-activate Quantization (WAT) and the Rationale for Weight-only Quantization (WOQ)

Quantization reduces both the memory and storage requirements of deep learning models by converting high-precision weights to lower-precision formats, such as 8-bit [7] or even 2-bit [5] integer values. This compression decreases the model's memory footprint, making it feasible to store and transfer model weights more efficiently. However, achieving high compression rates with minimal loss in model accuracy is challenging, particularly for extremely low-bit representations.

In practice, quantization can reduce memory demands and improve data transfer efficiency. By storing weights in low-precision formats, the model's memory access and data transfer between memory types, such as High-bandwidth Memory (HBM) and Static Random-access

Memory (SRAM), become faster. This accelerated data transfer enables faster loading of the model weights for inference or even low-bit training setups.

Weight-activate Quantization (WAT) [53], which reduces the precision of both weights and activations to the same low-bit level, is especially effective at 8-bit precision, where it enables matrix multiplications to occur directly in low precision, accelerating computations by using 8-bit tensor core [27]. This approach yields both memory and computation savings in hardware designed for low-precision operations, and the quantized results generally remain acceptable for large language models. However, at bit depths below 4 bits, weight-activation quantization tends to result in a significant accuracy drop [35], limiting its applicability in ultra-low-bit quantization.

In Weight-only Quantization (WOQ), weights are stored in low precision while activations remain in higher precision. This requires that weights be dequantized back to FP16 before performing General Matrix Multiplications (GEMM) with activation values [11]. Thus, accelerating the dequantization process itself is crucial for maintaining computational efficiency. By speeding up dequantization, the compressed weights can be accessed and utilized more quickly in computations, reducing the overall latency and enabling faster inference. Effective dequantization is particularly important in low-bit quantization setups, where the focus is on efficient data handling and maximizing throughput.

1.1.4 Challenges of Quantization-aware Training (QAT) and the Feasibility of Post-training Quantization (PTQ)

Quantization techniques may generally be divided into Post-training Quantization (PTQ) and Quantization-aware Training (QAT) [28]. QAT enhances model accuracy by incorporating quantization directly into the training process. However, this approach is less practical due to the excessive costs associated with managing vast numbers of parameters in pre-training and fine-tuning steps. Post-training Quantization (PTQ) is a more pragmatic approach; it is faster and requires no or few calibration data.

One challenge when quantizing large-scale LLM is that conventional Quantization-aware Training (QAT) is often impractical. Quantization-aware Training (QAT) involves incorporating quantization directly during training, which can improve accuracy by allowing the model to adjust to lower precision. However, retraining LLM with QAT is infeasible for models with billions of parameters due to the excessive computational costs involved. This limitation is compounded by the high resource demands of large LLM, making QAT unmanageable in practice.

An alternative is PTQ, a one-shot method that does not require retraining, making it a more feasible solution for LLM. PTQ allows compression of pretrained weights without fine-tuning, achieving a balance between compression and accuracy. Although PTQ methods offer this advantage, many techniques—such as round-to-nearest quantization [7, 55]—struggle to maintain accuracy for extreme compression, especially in low-bit configurations like 2-bit or 3-bit weight quantization.

For extremely large models like GPT-3 and Large Language Model Meta AI (LLaMA) [46], where every parameter reduction has a significant impact on memory usage, advanced PTQ methods are essential to enable higher compression rates while preserving accuracy. The primary aim is to find an optimal PTQ method that enables effective model compression while supporting efficient dequantization. This focus is critical since faster dequantization directly enhances computational throughput by expediting the conversion of low-precision weights for GEMM operations. However, achieving this balance remains an open challenge, as quantization methods capable of preserving accuracy at high compression rates are still underdeveloped.

1.1.5 Summary: Need for Low-Bit Weight-only Quantization (WOQ) with Efficient Dequantization

As discussed in 1.1.1, there is an increasing demand for compressing LLM to address their high memory and computational requirements, particularly through quantization. While

WAT (1.1.3) has shown effectiveness at higher bit levels, such as 8-bit, it faces significant accuracy challenges in low-bit configurations, limiting its applicability below 4 bits.

In response to these limitations, WOQ offers a promising alternative for ultra-low-bit configurations. However, as discussed in 1.1.4, QAT is generally impractical for LLMs due to the substantial computational resources required. Consequently, Post-training Quantization (PTQ) emerges as a feasible solution, yet current PTQ methods tend to achieve optimal performance only at 4 bits. Lower-bit settings, such as 3-bit and 2-bit quantization, remain challenging and require further exploration to ensure model accuracy.

Furthermore, to fully leverage low-bit quantization, an efficient dequantization process is essential for speeding up both inference and training. Therefore, this thesis focuses on developing a low-bit (3-bit, 2-bit) weight-only PTQ method that maintains accuracy while enhancing dequantization efficiency, enabling faster and more practical deployment of quantized LLM.

1.2 Literature Review

1.2.1 Quantization Techniques in Deep Learning

Quantization technique generally falls into two main categories: Quantization-aware Training (QAT) and Post-training Quantization (PTQ) [14, 28].

QAT maintains model performance by simulating quantization during the training process itself, allowing the model to adjust weights to lower precision levels [2, 6]. This approach leverages backpropagation to update quantized weights and can be effective in minimizing accuracy loss. However, QAT requires substantial computational resources and is therefore less feasible for large language models (LLM), which consist of billions of parameters. The high computational costs make it challenging to apply QAT directly to LLM, as the training process must accommodate the quantization constraints, adding to the overall training time and resource demand [14, 28].

PTQ, on the other hand, is a training-free method that quantizes models after training is completed, requiring no retraining and minimal calibration data [13, 32]. PTQ methods such as AdaRound [32] and BRECQ [25] use gradient-based optimization to determine optimal rounding thresholds for weights, aiming to minimize the quantization error. While these techniques reduce time-intensive computations by bypassing the need for full model retraining, tuning all weights remains computationally intensive for very large models. Due to these limitations, most LLM quantization methods prioritize training-free PTQ approaches [8, 11, 19, 51, 53], which sacrifice some accuracy in lower-bit precision scenarios.

1.2.2 PTQ Methods in LLM

Post-training Quantization (PTQ) has proven effective in large language models (LLM) by providing a practical approach to reduce model precision without the high costs of retraining. Notable PTQ methods include GPTQ [11], AWQ [26], and OmniQuant [42]. Each of these methods contributes unique techniques to address the challenges of quantizing weights in LLM, such as minimizing quantization error, adjusting scaling factors, and managing outliers.

GPTQ GPTQ builds on the Optimal Blockwise Quantization (OBQ) algorithm [10], which quantizes each row of the weight matrix based on the reconstruction error relative to the Hessian matrix. OBQ iteratively adjusts unquantized weights to mitigate errors, but this requires frequent updates to the Hessian matrix, which can increase computational complexity. GPTQ simplifies this by adopting a uniform left-to-right quantization order for each row, effectively reducing the need for constant Hessian updates. By computing the Hessian only during the quantization of one row and then reusing it for subsequent rows, GPTQ reduces computational demands. This approach enables near-lossless quantization at 4-bit precision, though its performance diminishes when generalized to lower precisions like 3-bit and 2-bit [11].

AWQ Recognizing that certain weight channels are more critical for preserving model performance, particularly those aligned with activation outliers, AWQ [26] implements a reparameterization method to address these differences in channel importance. This approach uses grid search to determine reparameterization coefficients, which effectively minimizes reconstruction error by adjusting the scale of important channels. Although AWQ performs well at 4-bit and shows acceptable performance at 3-bit, it struggles to maintain accuracy at 2-bit precision due to limitations in preserving fine-grained weight information. AWQ's focus on critical channels makes it robust in higher precision, but it faces difficulties as bit precision decreases.

OmniQuant To improve quantization in low-bit formats, OmniQuant [42] introduces block-wise scaling adjustments. OmniQuant optimizes quantization boundaries within blocks, allowing finer control over weight distributions, which can help retain model accuracy at extreme low-precision levels. However, OmniQuant's reliance on block-wise adjustments presents challenges, as scaling for extremely low-bit quantization remains difficult to balance between model accuracy and memory efficiency. OmniQuant's approach reflects an attempt to address limitations in both uniform and non-uniform quantization for LLM.

Other PTQ Methods Several other PTQ-based methods extend the capabilities of GPTQ and AWQ. For example, SpQR [8] assigns higher precision to weight outliers while quantizing the rest of the weights to 3 bits. OWQ [19] and LLM-MQ [20] offer mixed-precision strategies, selectively retaining weights in higher precision to accommodate outliers or assigning bit-widths dynamically using integer programming.

Summary Overall, these PTQ methods collectively aim to minimize quantization error by tuning weight precision, either through uniform adjustments (e.g., GPTQ) or adaptive strategies that recognize channel importance (e.g., AWQ). Although methods like OmniQuant and AWQ have shown improvements at higher precision levels, challenges remain for maintaining model fidelity at extremely low bit-widths, especially in 2-bit or sub-4-bit configurations.

Future work in PTQ-based quantization could integrate gradient-based optimizations from QAT for enhanced performance in ultra-low-bit scenarios while retaining PTQ's efficiency benefits.

1.2.3 Dequantization Challenges and Power-of-Two Quantization for Efficient Inference

In Weight-only Quantization (WOQ), particularly within PTQ, integer weights must be dequantized back to FP16 to ensure compatibility with GPU-based General Matrix Multiplications (GEMM) operations during inference. This dequantization step introduces significant computational costs, as it requires mixed-precision multiplications combined with zero-point additions, which add latency and degrade inference speed.

Mixed-precision quantization methods, such as AWQ, SpQR [8], and LLM.int8() [7], further complicate the dequantization process by retaining some critical weights in FP16 format. This adds to the computational complexity, as special kernels are needed to determine whether each weight is in FP16 or quantized format. GPUs are generally optimized for parallel dequantization, yet mixed-precision weights pose a challenge, as this structure disrupts parallel processing efficiencies, making it difficult to streamline dequantization for performance improvements.

Power-of-Two (PoT) quantization has recently emerged as a promising alternative to traditional uniform quantization, especially in contexts where dequantization efficiency is essential. By quantizing weights to values that are powers of two, PoT quantization enables dequantization with shift and addition operations, rather than costly mixed-precision multiplications [36, 55]. This shift-based approach can align effectively with the bell-shaped distribution of weights in deep neural networks, providing higher resolution for small weight values and speeding up the dequantization step significantly.

While PoT quantization offers notable computational efficiency, its application to large language models (LLM) introduces unique challenges. Originally developed for computer vision tasks, DenseShift [21] successfully applies PoT quantization to models such as ResNet50,

demonstrating performance gains in these traditional deep networks. However, when applied to LLM, naive PoT quantization—such as round to nearest (RTN) methods or adaptations to state-of-the-art techniques like GPTQ, AWQ, and OmniQuant—results in significant accuracy degradation due to the broader and more varied weight distributions of LLM. The coarse granularity of PoT quantization at higher weight magnitudes exacerbates this degradation, indicating that direct adaptations from Computer Vision models are insufficient for LLMs.

Recently, ShiftAddLLM [57] has explored the use of PoT quantization specifically tailored for LLM, taking a different approach by implementing column-wise scaling and sequential quantization. This technique allows each subsequent quantized value to compensate for the error introduced by previous values, enhancing accuracy preservation. By quantizing weights to PoT values, ShiftAddLLM enables dequantization using efficient shift and addition operations, presenting a promising direction for LLM in WOQ settings.

These advancements highlight the potential of PoT quantization for but also underscore the need for further refinement. While the efficiency of PoT quantization in reducing dequantization complexity is advantageous, future approaches must carefully address the accuracy challenges posed by the unique characteristics of LLM weight distributions.

1.3 Goal, Contribution, and Organization

1.3.1 Objective

This thesis aims to develop an effective Post-training Quantization (PTQ) method that leverages Power-of-Two (PoT) quantization to address limitations in existing quantization approaches, particularly in extremely low-precision formats like 2-bit and 3-bit. Our goal is to enhance accuracy and reduce memory costs, facilitating efficient deployment of large language models (LLM) on GPUs.

1.3.2 Two-Step PoT Quantization and Dequantization Approach

We propose a novel two-step PoT quantization approach specifically designed for LLM. The first step involves finding an optimal initial scale factor based on the weight distribution, which significantly improves quantization quality compared to round-to-nearest methods. In the second step, we fine-tune the scaling by introducing a learnable coefficient on the clipping bounds, enabling partial Quantization-aware Training (QAT) with minimal GPU resource requirements. This process requires backpropagating only a minimal number of parameters, making it computationally feasible for a single GPU.

In addition to the quantization approach, we provide a customized dequantization kernel tailored to the PoT format. This kernel leverages the power-of-two representation to perform dequantization using efficient bitwise shifts and additions instead of mixed-precision multiplications. By exploiting the parallel processing capabilities of GPUs, our kernel achieves significant speedup over traditional methods. Unlike previous PoT approaches, which are primarily optimized for sequential processing on ARM CPUs, our method is optimized for GPU parallelism.

1.3.3 Advantages of the Proposed Approach

Our approach offers multiple advantages:

- **Improved Quantization Efficiency:** Our method enables rapid quantization of LLM with minimal resource requirements, making it feasible on a single GPU.
- **Enhanced Accuracy at Low Bit Levels:** The two-step approach effectively addresses rounding errors, allowing PoT quantization to perform well at extreme low-bit levels, such as 2-bit and 3-bit.
- **Optimized Dequantization Speed:** Our customized GPU dequantization kernel accelerates inference by 3.6x on the NVIDIA V100 and 1.5x on the NVIDIA RTX 4090, taking full advantage of bitwise operations and GPU parallelism.

1.4 Organization of the Thesis

The organization of this thesis is as follows:

- **Chapter 2** provides a comprehensive overview of symmetric and non-symmetric quantization techniques, including the dequantization process. It discusses existing PTQ methods such as GPTQ, AWQ, and OmniQuant, which will be compared with our proposed approach.
- **Chapter 3** describes our PTQ method, focusing on WOQ for PoT quantization and the design of an optimized dequantization kernel.
- **Chapter 4** summarizes the results and key observations of our study, followed by a discussion of potential future research directions.
- **Chapter 5** gives a discussion of potential future research directions.

1.5 Notation

In this thesis, we adopt the following conventions for notations. Boldface lowercase letters, such as $\mathbf{x}$, denote vectors, and boldface uppercase letters, such as $\mathbf{X}$, denote matrices. Scalars are represented using normal lowercase and uppercase letters, such as x and X, respectively. Superscripts, such as (l), are used to indicate quantities corresponding to the l-th layer in a neural network.

For an $m \times n$ matrix $\mathbf{A}$, its (i, j)-th element is denoted by a_{ij}, or $\mathbf{A}_{ij}$. The submatrix containing rows i to j is denoted by $\mathbf{A}_{i:j,:}$, and the submatrix containing columns i to j is denoted by $\mathbf{A}_{:,i:j}$. The transpose of a matrix $\mathbf{A}$ is denoted as $\mathbf{A}^\mathsf{T}$.

We use $\mathbb{R}$ to denote the set of real numbers and $\mathbb{Z}$ to denote the set of integers. For a set $\mathcal{S}$, $\mathcal{S}^n$ represents the set of n-dimensional column vectors with entries from $\mathcal{S}$, and $\mathcal{S}^{m\times n}$ represents the set of $m \times n$ matrices with entries from $\mathcal{S}$.

For a scalar x, $\widetilde{x}_Q$ represents its quantized value, while $\widetilde{x}$ denotes its dequantized value. Similarly, for a vector $\mathbf{x}$, $\widetilde{\mathbf{x}}_Q$ and $\widetilde{\mathbf{x}}$ represent its quantized and dequantized vectors, respectively. For matrices, $\widetilde{\mathbf{X}}_Q$ and $\widetilde{\mathbf{X}}$ denote the quantized and dequantized matrices. These representations are used consistently throughout this thesis.

For sets, such as a quantization grid, we use notations like $\mathcal{G}_i$ to denote the grid for a specific column i. The set typically contains elements derived from quantization parameters such as scaling factors and quantization levels. The specific structure of such sets is introduced in the respective sections.

The sign of a scalar x is denoted by $\text{sign}(x)$, which returns 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$. The clamp operation limits a scalar x to a specified range $[\min, \max]$ and is defined as

$$\text{clamp}(x, \min, \max) = \begin{cases} \min, & \text{if } x < \min, \\ x, & \text{if } \min \leq x \leq \max, \\ \max, & \text{if } x > \max. \end{cases}$$

The component-wise product of two vectors or matrices of the same dimensions is denoted by $\mathbf{A} \circ \mathbf{B}$, where $(\mathbf{A} \circ \mathbf{B})_{ij} = a_{ij} \cdot b_{ij}$ for all valid i, j. This operation applies multiplication individually to corresponding elements.

For a matrix $\mathbf{A}$, $|\mathbf{A}| = (|a_{ij}|)$. The Euclidean norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$. The Frobenius norm of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is $\|\mathbf{A}\|_F$, i.e., $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$.

Finally, for a random variable $\mathbf{x}$, $\mathbb{E}[\mathbf{x}]$ represents its expectation. If a random vector $\mathbf{x}$ follows a normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, we write $\mathbf{x} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Chapter 2

Background and Quantization Techniques

This chapter covers essential quantization techniques for efficient deployment of large language models (LLMs). We start by introducing the architecture of LLMs, detailing the layer-wise computations that underpin model functionality. Following this, we explain core concepts, distinguishing between uniform and non-uniform schemes, symmetric and non-symmetric formats, and single- vs. mixed-precision strategies. We then introduce the primary quantization paradigms, Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ), focusing on PTQ for its practicality in LLMs. Finally, we explore weight-activation and weight-only quantization, examining their impact on memory efficiency and computational complexity, with particular attention to the advantages of weight-only quantization in low-bit scenarios.

Then, we explore state-of-the-art PTQ weight-only quantization methods, including GPTQ [11], AWQ [26], and OmniQuant [42], providing a detailed overview of each method's strengths and limitations.

2.1 LLM Architecture and Layer-wise Computation

2.1.1 Matrix Representations in LLaMA Models

In transformer-based models such as LLaMA [46], designed for language processing tasks, each layer contains essential parameter matrices that enable the model to generate coherent text, summarize, translate, and understand context by processing large volumes of text data. These matrices include the query ($\mathbf{W}_q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$), key ($\mathbf{W}_k \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$), value ($\mathbf{W}_v \in \mathbb{R}^{d_{\text{model}} \times d_{\text{head}}}$), and output ($\mathbf{W}_o \in \mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$) matrices within the attention module, as well as matrices in the feedforward (FFN) module, specifically $\mathbf{W}_{\text{down}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $\mathbf{W}_{\text{gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, and $\mathbf{W}_{\text{project}} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$. Here, d_{model} is the embedding dimension, d_{head} is the dimensionality of each attention head, and d_{ff} is the intermediate dimension in the feedforward network which is consistent across layers.

Let T be the length of the input sequence of the ordered tokens in a segment of text processed by the model. In layer l, the input is the matrix $\mathbf{H}^{(l-1)} \in \mathbb{R}^{T \times d_{\text{model}}}$, representing the hidden state output from layer $l-1$. The matrix $\mathbf{H}^{(0)}$ is the initial input embeddings, where each token in the sequence is mapped to a d_{model}-dimensional vector in the embedding space. To compute the output of layer l, the model first performs the self-attention operation. The attention scores are computed as follows:

1. **Self-Attention Layer**: For each token's representation within the input matrix $\mathbf{H}^{(l-1)}$, which serves as the hidden states passed into layer l, compute the **Query (Q)**, **Key (K)**, and **Value (V)** transformations:

$$\mathbf{Q}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_q^{(l)}, \quad \mathbf{K}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_k^{(l)}, \quad \mathbf{V}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}_v^{(l)}, \tag{2.1}$$

   where $\mathbf{W}_q^{(l)}$, $\mathbf{W}_k^{(l)}$, and $\mathbf{W}_v^{(l)}$ are the respective weight matrices for layer l that project the input into the query, key, and value spaces, with dimensions as previously noted.

The attention scores are computed using scaled dot-product attention:

$$\text{Attention}(\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)}) = \text{softmax}\left(\frac{\mathbf{Q}^{(l)}\mathbf{K}^{(l)\mathsf{T}}}{\sqrt{d_{\text{head}}}}\right)\mathbf{V}^{(l)}, \tag{2.2}$$

where softmax is applied along the sequence dimension to normalize the scores, and d_{head} is used to scale the dot product for stable gradients.

The attention result is then projected through the **Output (O)**:

$$\mathbf{H}_{\text{attn}}^{(l)} = \text{Attention}(\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)})\mathbf{O}^{(l)}, \tag{2.3}$$

where $\mathbf{O}^{(l)} \in \mathbb{R}^{d_{\text{head}} \times d_{\text{model}}}$ is derived from the weight matrix $\mathbf{W}_o^{(l)}$ in layer l. This projection ensures that the attention output aligns with the original embedding dimension, enabling the integration of results into subsequent layers.

2. **Feed-Forward Network (FFN) with SwiGLU Activation**:

   After the self-attention operation, the output is processed through a feed-forward network, commonly referred to as a **Multilayer Perceptron (MLP)**. An MLP consists of fully connected layers, which serve to transform the input embeddings and add non-linearity, enhancing the model's expressive power. The MLP in each Transformer layer utilizes a **SwiGLU** (Switchable Gated Linear Unit) activation function, which is defined as follows:

$$\mathbf{H}_{\text{act}}^{(l)} = \text{SwiGLU}(\mathbf{H}_{\text{attn}}^{(l)}\mathbf{W}_{\text{down}}^{(l)}, \mathbf{H}_{\text{attn}}^{(l)}\mathbf{W}_{\text{gate}}^{(l)}) = \text{ReLU}(\mathbf{H}_{\text{attn}}^{(l)}\mathbf{W}_{\text{gate}}^{(l)}) \odot (\mathbf{H}_{\text{attn}}^{(l)}\mathbf{W}_{\text{down}}^{(l)}), \tag{2.4}$$

   where $\mathbf{W}_{\text{down}}^{(l)}$ and $\mathbf{W}_{\text{gate}}^{(l)}$ are the MLP weights for layer l, and $\odot$ denotes element-wise multiplication. Here, **ReLU** (Rectified Linear Unit) is an activation function that outputs the input if it is positive and zero otherwise, introducing non-linearity to the model. The activated output is then projected back to the original dimension with a

bias matrix included:

$$\mathbf{H}_{\text{mlp}}^{(l)} = \mathbf{H}_{\text{act}}^{(l)} \mathbf{W}_{\text{project}}^{(l)} + \mathbf{B}_{\text{project}}^{(l)}, \tag{2.5}$$

where $\mathbf{W}_{\text{project}}^{(l)}$ and $\mathbf{B}_{\text{project}}^{(l)}$ are the projection weight and bias for layer l.

3. **Layer Output with Residual Connection**:

   Each layer l in the Transformer model applies residual connections to preserve information from previous computations. This process combines the input from the previous layer with the outputs of the self-attention and MLP components, allowing the model to iteratively refine its representations across layers:

$$\mathbf{H}^{(l)} = \mathbf{H}^{(l-1)} + \mathbf{H}_{\text{attn}}^{(l)} + \mathbf{H}_{\text{mlp}}^{(l)}, \tag{2.6}$$

   where $\mathbf{H}^{(l)}$ represents the output of layer l, $\mathbf{H}_{\text{attn}}^{(l)}$ is the output from the attention mechanism, and $\mathbf{H}_{\text{mlp}}^{(l)}$ is the output from the feed-forward network in layer l. These residual connections help maintain the gradient flow across layers and support efficient information propagation, ultimately enabling the model to capture complex patterns over multiple layers.

   The entire process for layer l can be summarized as a transformation function:

$$\mathbf{H}^{(l)} = \mathcal{F}^{(l)}(\mathbf{W}^{(l)}, \mathbf{H}^{(l-1)}), \tag{2.7}$$

   where $\mathcal{F}^{(l)}$ represents the composite transformation of the attention and MLP modules for layer l, and $\mathbf{W}^{(l)}$ denotes the set of weight matrices:

$$\mathbf{W}^{(l)} = \left\{ \mathbf{W}_q^{(l)}, \mathbf{W}_k^{(l)}, \mathbf{W}_v^{(l)}, \mathbf{W}_o^{(l)}, \mathbf{W}_{\text{down}}^{(l)}, \mathbf{W}_{\text{gate}}^{(l)}, \mathbf{W}_{\text{project}}^{(l)} \right\}.$$

   This layered structure, connected by residual links, allows the LLaMA model to iteratively refine its understanding of the input sequence across multiple layers.

2.2 Introduction to Quantization

Quantization is a critical technique for reducing the memory and computational footprint of large language models (LLMs) by converting high-precision values into lower-precision representations [11, 26]. A typical quantization method consists of two primary steps: *mapping* and *rounding*.

In quantization, several key quantities are involved:

- X: the original high-precision value.
- $\widetilde{X}_{\mathrm{Q}}$: the quantized value, typically stored in a low-bit integer format.
- $\widetilde{X}$: the dequantized value that approximates the original X.
- S: the scale factor, which maps the original value range to a quantized range.
- Z: the zero point, an offset to adjust the quantized range for alignment with the data distribution.

In the *mapping* phase, high-precision discrete values within the IEEE 754 floating-point range [34] are projected onto a finite interval suitable for the desired bit precision.

Quantization methods can be classified based on the mapping range as either *symmetric* or *non-symmetric*. In symmetric quantization (see Section 2.2.2), the quantized range is centered around a zero point, making it suitable for distributions that are approximately symmetric. Non-symmetric quantization (see Section 2.2.2) is preferred when the data distribution has a significant bias.

Additionally, quantization can be divided based on value distribution within the quantized range into *uniform* and *non-uniform quantization* (see Section 2.2.3 and Section 2.2.3), as further explored in Section 2.2.3. Uniform quantization maintains equal distances between quantization levels, while non-uniform quantization varies level spacing to align more closely with data distributions.

The mapping approach can also vary based on precision. In *single-precision quantization*, values are mapped to one consistent low-bit precision, whereas *mixed-precision quantization* assigns varying precision levels to different parts of the model, allowing for targeted optimization. These approaches are discussed in Section 2.2.4.

Once values are mapped, the *rounding* phase adjusts these values to discrete integers within the specified range. Rounding methods commonly include rounding to nearest (see Section 2.2.1) and stochastic rounding (see Section 2.2.1). Here, the quantized value $\widetilde{X}_{\mathrm{Q}}$ represents a discrete approximation of the original high-precision value X, based on the chosen rounding technique.

In practical model quantization tasks, different quantization methodologies are selected based on specific requirements and objectives. These include *Quantization-Aware Training (QAT)* and *Post-Training Quantization (PTQ)*, as discussed in Section 2.2.5, along with weight-activation and weight-only quantization approaches covered in Section 2.2.6. Since rounding plays a fundamental role in all quantization methods, we begin by introducing rounding techniques in the following subsection.

2.2.1 Rounding Techniques

In quantization, after mapping the values to the desired range, a *rounding* method is applied to convert floating-point values to integers. The choice of rounding method significantly impacts the quantization error, particularly in low-bit quantization scenarios. The two predominant rounding techniques are *Rounding to Nearest* and *Stochastic Rounding*.

Rounding to Nearest

Rounding to Nearest (RTN) is the simplest and most commonly used rounding technique. In this method, each value is rounded to the nearest integer:

$$\text{round}(X) = \begin{cases} \lfloor X \rfloor, & \text{if } X - \lfloor X \rfloor < 0.5, \\ \lceil X \rceil, & \text{if } X - \lfloor X \rfloor \geq 0.5, \end{cases} \tag{2.8}$$

where $\lfloor X \rfloor$ represents the floor function, or the largest integer less than or equal to X, and $\lceil X \rceil$ represents the ceiling function, or the smallest integer greater than or equal to X. This method minimizes quantization error for individual values but can lead to significant precision loss in low-bit quantization, especially when accumulated over multiple layers in a deep neural network.

Stochastic Rounding

To address the limitations of RTN, *Stochastic Rounding* introduces randomness in the rounding decision. Instead of deterministically rounding up or down, each value is rounded probabilistically, based on its fractional part:

$$\text{round}(X) = \begin{cases} \lfloor X \rfloor, & \text{with probability } p(X) = \lceil X \rceil - X, \\ \lceil X \rceil, & \text{with probability } 1 - p(X) = X - \lfloor X \rfloor, \end{cases} \tag{2.9}$$

This ensures that the expected value of the rounded result remains close to the original value, i.e., $\text{E}\{\text{round}(x)\} = x$ (see [52] and [12]). Stochastic rounding has been shown to perform well in low-bit quantization settings by spreading rounding errors more evenly across computations, reducing the risk of catastrophic error accumulation in deep neural networks [12, 52].

Stochastic rounding is particularly useful in settings where preserving statistical properties across layers is important, such as in post-training quantization for LLMs. However, relying solely on element-wise rounding methods like RTN or Stochastic Rounding is often insufficient for capturing the cross-effects of accumulated quantization losses across multiple layers in deep neural networks, such as LLMs. These methods do not account for interdependencies between parameters, which can lead to significant precision degradation when accumulated

over many layers. Advanced quantization techniques have been developed to address these limitations by incorporating layer-wise or block-wise adjustments to better handle the compounded effects of rounding errors. These approaches will be introduced in more detail in Section 2.3.

2.2.2 Symmetric and Non-Symmetric Quantization

Quantization can be classified as either *symmetric* or *non-symmetric*, depending on whether the quantized range is centered around a zero point, which serves as the midpoint of the quantization range. Symmetric quantization is effective for data distributions centered around a zero point, often zero, while non-symmetric quantization is advantageous for data with a significant positive or negative skew. These approaches are illustrated in Figure 2.1.

In our quantization framework, quantization is applied to a set of scalar values, denoted by $\{X\}$. The scalars $X_{\max}$ and $X_{\min}$ refer to the maximum and minimum values in $\{X\}$, respectively. The quantization parameters, specifically the scale S and (if needed) the zero point Z, are shared across this group $\{X\}$, ensuring a consistent quantization scheme for all values in the set.

Symmetric Quantization

In symmetric quantization, the quantized range is centered around a zero point $Z = 0$, with values quantized symmetrically around this midpoint. This approach is well-suited for weight distributions often centered near zero, which is common due to regularization in neural networks [1, 16, 18].

For N-bit symmetric quantization, the quantized value $\widetilde{X}_{\text{symmetric\_Q}}$ is computed as:

$$\widetilde{X}_{\text{symmetric\_Q}} = \text{round}\left(\frac{X}{S}\right), \tag{2.10}$$

where the scale factor S is given by

$$S = \frac{\max(|X_{\max}|, |X_{\min}|)}{2^{N-1} - 1}. \tag{2.11}$$

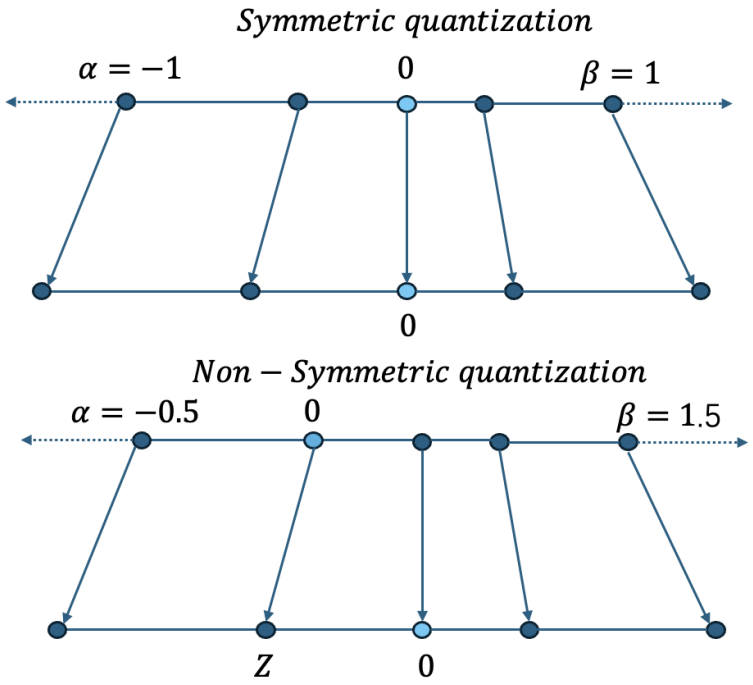


Figure 2.1: Illustration of symmetric quantization (top) and non-symmetric quantization (bottom). In symmetric quantization, the quantization range is centered around a zero point (e.g., $[-127, 127]$ for an 8-bit symmetric range), while in non-symmetric quantization, the range is shifted to better accommodate an unbalanced data distribution.

Here, 2^{N-1} values are possible for the absolute magnitude of the data, with the remaining bit reserved for the sign, yielding $2^{N-1} - 1$ intervals between values. This format typically stores values in a sign-and-modulus representation, where the first bit stores the sign, and the remaining bits represent the modulus of the value.

The dequantized value $\widetilde{X}_{\text{symmetric}}$, an approximation of the original value X, is then computed as:

$$\widetilde{X}_{\text{symmetric}} = \widetilde{X}_{\text{symmetric\_Q}} \cdot S. \tag{2.12}$$

Non-Symmetric Quantization

Non-symmetric quantization is useful for handling skewed data distributions by allowing an adjustable zero point Z, aligning the quantized range more closely with the natural distribution of values in $\{X\}$ [30, 33]. For non-symmetric quantization, the quantized value

$\widetilde{X}_{\text{non-symmetric\_Q}}$ is calculated as

$$\widetilde{X}_{\text{non-symmetric\_Q}} = \text{round}\left(\frac{X}{S} - Z\right), \tag{2.13}$$

where the scale factor S and the zero point Z are given by:

$$S = \frac{X_{\max} - X_{\min}}{2^N - 1}, \tag{2.14}$$

$$Z = \text{round}\left(\frac{X_{\min}}{S}\right). \tag{2.15}$$

In non-symmetric quantization, all N bits are allocated to represent the value, typically stored in a two's complement format, which yields $2^N - 1$ intervals in the quantized range.

The dequantized value $\widetilde{X}_{\text{non-symmetric}}$, approximating X, is given by:

$$\widetilde{X}_{\text{non-symmetric}} = \left(\widetilde{X}_{\text{non-symmetric\_Q}} + Z\right) \cdot S. \tag{2.16}$$

Summary of Symmetric and Non-Symmetric Quantization

The table below summarizes the key differences between symmetric and non-symmetric quantization, including the quantization and dequantization equations.

	Symmetric Quantization	Non-Symmetric Quantization
Scale Factor (High Bit)	$\frac{\max(\lvert X_{\max}\rvert, \lvert X_{\min}\rvert)}{2^{N-1}-1}$	$\frac{X_{\max}-X_{\min}}{2^N-1}$
Zero Point Z (Low Bit)	0	$\text{round}\left(\frac{X_{\min}}{S}\right)$
Quantized Value (Low Bit)	$\text{round}\left(\frac{X}{S}\right)$	$\text{round}\left(\frac{X}{S} - Z\right)$
Dequantized Value (High Bit)	$\widetilde{X}_{\text{symmetric\_Q}} \cdot S$	$\left(\widetilde{X}_{\text{non-symmetric\_Q}} + Z\right) \cdot S$

Table 2.1: Summary of symmetric and non-symmetric quantization equations, with notation for low-bit and high-bit components.

In summary, symmetric quantization provides computational efficiency and simplicity, especially for data centered around zero, while non-symmetric quantization offers greater accuracy by adjusting for data skew with a flexible zero point.

2.2.3 Uniform and Non-Uniform Quantization

Quantization techniques can be further categorized based on the *mapping* procedure into *uniform* and *non-uniform* quantization (see Figure 2.2). These categories define how the continuous values are distributed over the quantized range.

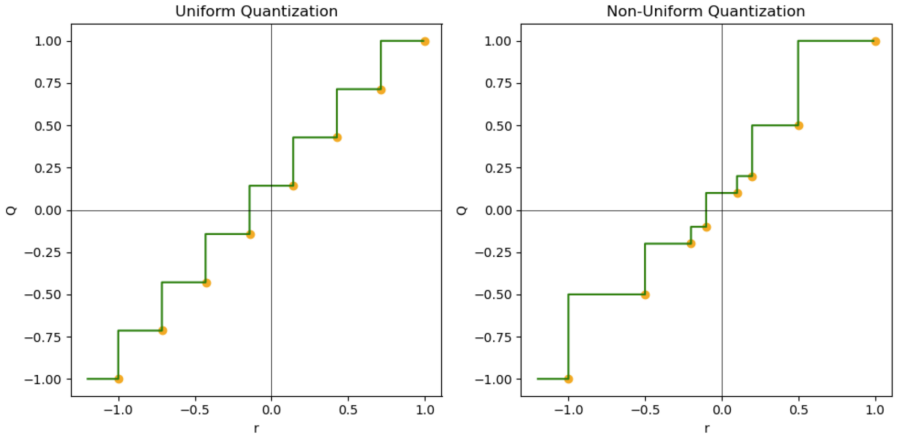


Figure 2.2: Comparison between uniform quantization (left) and non-uniform quantization (right). Real values in the continuous domain r are mapped into discrete, lower precision values in the quantized domain Q, which are marked with the orange bullets. Note that the distances between the quantized values (quantization levels) are the same in uniform quantization, whereas they can vary in non-uniform quantization.

Uniform Quantization

Uniform quantization divides the value range into equal intervals, making it computationally efficient and straightforward for hardware implementation. This process, also known as range-based or min-max quantization [17, 40], maps continuous values to a discrete integer range using a fixed scale S and, in non-symmetric cases, a zero point Z.

In symmetric quantization, the quantized value $\widetilde{X}_{\text{uniform\_Q}}$ can be obtained using Equation (2.10), while in non-symmetric quantization, Equation (2.13) is applied.

The dequantized value $X_{\text{uniform\_Q}}$ can be reconstructed from $\widetilde{X}_{\text{uniform\_Q}}$ by applying the corresponding dequantization equations—Equation (2.12) for symmetric quantization and Equation (2.16) for non-symmetric quantization.

For non-symmetric quantization, the choice of zero point Z is discussed in Section 2.2.2, providing alignment with the data's range. Uniform quantization is effective for evenly distributed values, resulting in minimal precision loss compared to the full-precision model [49].

Non-Uniform Quantization

Non-uniform quantization allocates quantized values based on the frequency distribution of the data, providing more precision to frequently occurring values and less precision to rare ones. This approach is beneficial for values that follow a non-uniform or heavy-tailed distribution, which is often seen in trained neural network weights [15, 54]. As illustrated in Figure 2.3, neural network weights are typically denser around zero and sparser at the extremes, aligning well with the concept of non-uniform quantization.

- **Look-Up Table (LUT) Quantization**: In LUT-based quantization, a set of quantized values is stored in a look-up table. Each original input value is mapped to the closest entry in this table, representing a quantized approximation of the original value [4, 45, 50]. This approach allows for efficient storage and retrieval, as the look-up table only contains a reduced set of representative quantized values, minimizing memory usage and speeding up computations.

- **Power-of-Two (PoT) Quantization**: PoT quantization typically uses symmetric quantization, where the quantized values are restricted to powers of two without a zero point [23,36]. This approach allows for efficient computations by enabling bitwise shift operations instead of multiplications.

  Different from Equation (2.11), the scale factor S in PoT quantization is calculated as:

$$S = \frac{\max(|X_{\max}|, |X_{\min}|)}{2^{q_{\max}} - 1}, \tag{2.17}$$

where $q_{\max} = 2^{N-1} - 1$ for an N bits quantization. The quantized value $\widetilde{X}_{\text{PoT\_Q}}$ is obtained by:

$$\widetilde{X}_{\text{PoT\_Q}} = \text{round}\left(\log_2\left(\frac{|X|}{S}\right)\right), \tag{2.18}$$

The dequantized value $\widetilde{X}_{\text{PoT}}$ is then computed as:

$$\widetilde{X}_{\text{PoT}} = 2^{\widetilde{X}_{\text{PoT\_Q}}} \cdot \text{sign}(X) \cdot S, \tag{2.19}$$

where $\text{sign}(X)$ represents the sign of X.

In general, the dequantization process in Equation (2.19) can be faster than in Equation (2.16) because it enables the use of bitwise shifts instead of additions and multiplications, providing a significant speed-up in dequantization. Additionally, if S is specifically designed to be a power of two, such as $S = 2^k$ for some integer k, then the dequantized weight $\widetilde{X}_{\text{PoT}}$ will also be a power of two, **regardless of the original value of** X. This enables further optimization, as multiplications in inference can be replaced by shift operations, leading to substantial computational savings in the inference steps of the model.

Non-uniform quantization methods like LUT and PoT are advantageous over uniform quantization methods when the original values are not uniformly distributed, as they allow for more efficient representation. Additionally, they can potentially enable faster computations if customized computation kernels are designed to leverage the non-uniform characteristics of these methods.

For example, in LUT quantization, the retrieval process can be faster than performing computations, as values are directly mapped to a stored set of representative quantized values. In Power-of-Two (PoT) quantization, the dequantization process can be accelerated if the algorithm is optimized to use bitwise shift operations in place of multiplications. However, the dequantization for non-uniform methods like LUT and PoT needs to be carefully designed to avoid performance issues, as these methods do not maintain a simple

linear relationship between the quantized and original values. Without optimized dequantization, these methods could potentially be slower than the standard uniform quantization.

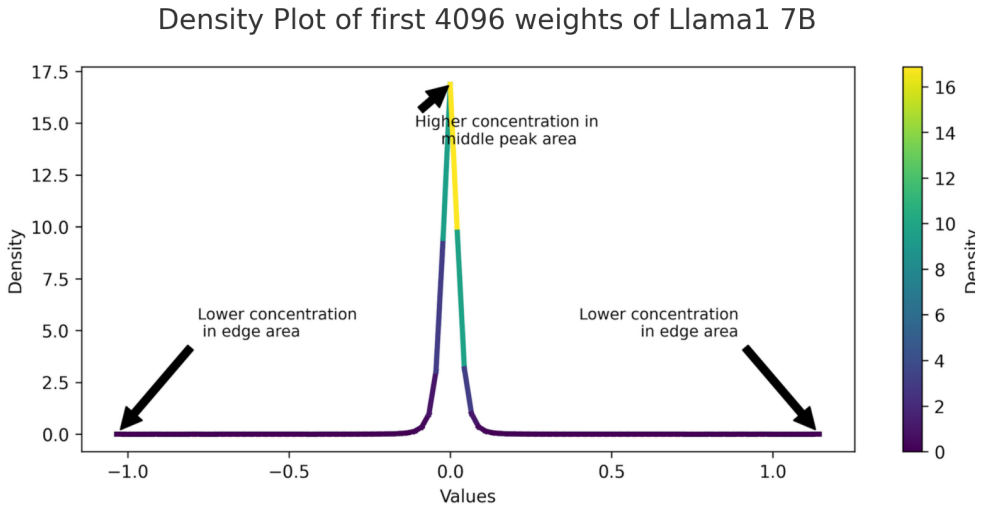


Figure 2.3: Weight distribution for LLaMa1 7B, illustrating the typical dense mid-range and sparse extreme values found in neural network weights.

2.2.4 Single-Precision vs. Mixed-Precision Quantization

Single-precision quantization refers to applying a consistent bit-width across all model weights, where each weight is quantized to the same precision level, such as 8-bit or 4-bit integers. This uniform approach simplifies computation, especially for hardware implementations, by avoiding the complexity introduced by handling different precisions. While single-precision quantization allows for efficient processing, it may sacrifice accuracy, particularly in low-bit scenarios, due to the loss of granularity in representing critical model parameters.

In contrast, mixed-precision quantization selectively applies different bit-widths to different weights based on their significance within the model. In this approach, certain weights are retained at higher precision, often FP16, to preserve critical information, while other weights are quantized to lower bit-widths, such as 4-bit or 8-bit integers. This technique, employed by methods such as AWQ, OWQ, and SpQR, maintains higher accuracy by protecting the precision of weights that contribute substantially to model performance, while still reducing the memory footprint by quantizing less crucial weights.

However, mixed-precision quantization introduces complexity during inference, especially for GPU-based implementations. Efficient dequantization of mixed-precision weights is challenging, as it disrupts parallel processing due to the varying bit-widths across weights. Customized GPU kernels are often required to manage the heterogeneous precision levels effectively, which can add significant computational overhead. Consequently, although mixed-precision quantization can yield better accuracy, it may slow down inference due to the intricate dequantization and computation processes required for handling diverse weight precisions.

2.2.5 Quantization Training Paradigms: QAT and PTQ

In deep learning, two primary paradigms are used for training quantized models: Quantization-Aware Training (QAT) [28, 43] and Post-Training Quantization (PTQ) [5, 26, 42, 53]. Both methods aim to reduce the bit-width of model parameters, but they differ significantly in computational demands and implementation, as shown in Figure 2.4.

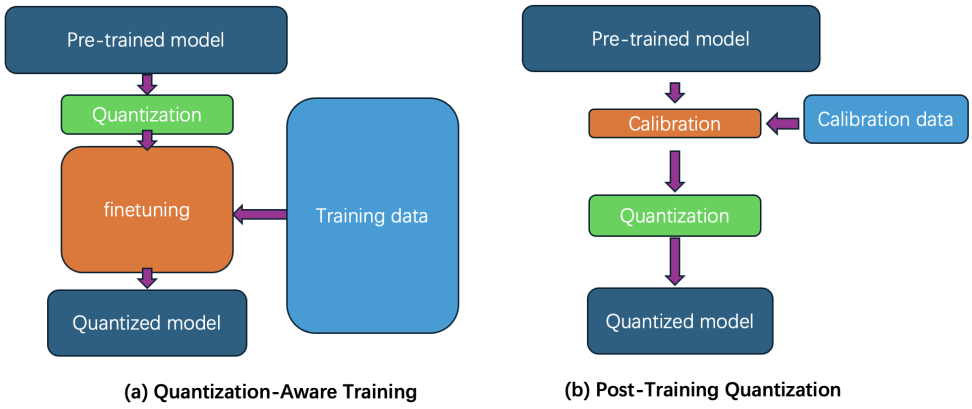


Figure 2.4: Comparison between Quantization-Aware Training (QAT, a) and Post-Training Quantization (PTQ, b). In QAT (a), a pre-trained model undergoes quantization followed by fine-tuning with training data to adjust parameters and mitigate accuracy degradation. In PTQ (b), a pre-trained model is calibrated using a small subset of data (calibration data) to compute the clipping ranges and scaling factors without additional fine-tuning.

- **Quantization-Aware Training (QAT)**: QAT integrates quantization into the training process, allowing the model to adapt to quantization constraints as it learns. This

approach, often referred to as end-to-end quantization, involves incorporating simulated quantization effects into the forward and backward passes. By embedding quantization within the training loops, QAT enables the model to minimize quantization-induced errors, leading to improved accuracy. However, QAT is computationally intensive and typically requires substantial training data and resources, particularly for large models like LLMs. Recent research has explored ways to reduce the data and compute requirements for QAT, but these efforts remain challenging in practical applications.

- **Post-Training Quantization (PTQ)**: PTQ applies quantization after the model has been fully trained, making it more practical for large-scale models due to its reduced computational requirements. Unlike QAT, PTQ does not adapt the model to quantization during training, which can lead to accuracy degradation, especially at low bit-widths. To mitigate this, PTQ methods often rely on minimal calibration data, as seen in techniques like GPTQ and OmniQuant [11, 42]. Some advanced PTQ approaches, such as Adaptive Quantization (AdpQ), operate without any calibration data, aligning layer-wise outputs using a limited number of sampled data points. This calibration-free or minimal-calibration design allows PTQ to be implemented on a single GPU, further enhancing its accessibility for large model deployment.

In summary, while QAT enhances model accuracy by directly addressing quantization effects during training, it is resource-intensive. PTQ, by contrast, excludes itself from the training procedure and instead applies quantization post-training, making it more computationally feasible, though potentially less accurate, especially in very low-bit scenarios. The choice between QAT and PTQ depends on the computational resources available and the required trade-off between accuracy and efficiency.

2.2.6 Weight-Activation Quantization vs. Weight-Only Quantization

Post-Training Quantization (PTQ) methods can be further categorized into *weight-activation quantization* and *weight-only quantization*, each with distinct trade-offs in memory footprint, computational efficiency, and model accuracy.

In neural networks, *activations* refer to the outputs produced by each layer after applying a transformation to the input. They represent the intermediate values that carry information through the network layers from input to output. Mathematically, for a given layer l, the activation $\mathbf{H}^{(l)}$ is computed as:

$$\mathbf{H}^{(l)} = \mathcal{F}^{(l)}(\mathbf{W}^{(l)}, \mathbf{H}^{(l-1)}), \tag{2.20}$$

which was previously introduced in Equation (2.7) and detailed in Section 2.1.1.

- **Weight-Activation Quantization**: In this approach, both the weights and activations are quantized to a lower bit-width (e.g., INT8). This results in a reduced memory footprint and can accelerate inference due to the lower bit precision used throughout the computations. However, quantizing activations alongside weights may lead to some accuracy loss, especially in large language models (LLMs) where maintaining high precision in activations can be critical.

- **Weight-Only Quantization**: In weight-only quantization, only the weights are quantized, while the activations remain in higher precision (e.g., FP16). This approach strikes a balance between memory savings and computational efficiency, preserving model accuracy by keeping the activations in a high-precision format. Weight-only quantization is particularly advantageous for scenarios where activation quantization may significantly degrade performance.

The inference processes for these two approaches differ as follows:

1. **Inference Flow for Weight-Only Quantization (Figure 2.5(a))**:

    (1). Quantized Weight (INT8) → De-quantize (FP16)

    (2). Perform GEMM/GEMV with FP16 Activations

    (3). Use FP16 Accumulator

    (4). Produce FP16 Output Activation

    In weight-only quantization, the quantized weights are first de-quantized back to FP16 before being used in matrix multiplications (General Matrix Multiply, GEMM) or matrix-vector multiplications (General Matrix-Vector Multiply, GEMV). The high-precision FP16 activations are then multiplied by the de-quantized weights using an FP16 accumulator, resulting in a final FP16 activation output.

2. **Inference Flow for Weight-Activation Quantization (Figure 2.5(b))**:

    (1). Quantized Weight (INT8)

    (2). Quantize FP16 Activations to INT8

    (3). Perform GEMM with INT8 Accumulator

    (4). De-quantize Output to FP16 Activation

    In weight-activation quantization, both weights and activations are quantized to INT8. The matrix multiplication and accumulation are performed with quantized values, resulting in an INT8 accumulator. Afterward, the accumulated value is de-quantized back to FP16 for the final output activation.

In these processes, GEMM (General Matrix Multiply) is used for matrix-matrix multiplications, while GEMV (General Matrix-Vector Multiply) is used for matrix-vector operations, both of which are central to the computations in LLMs. The choice between weight-only and

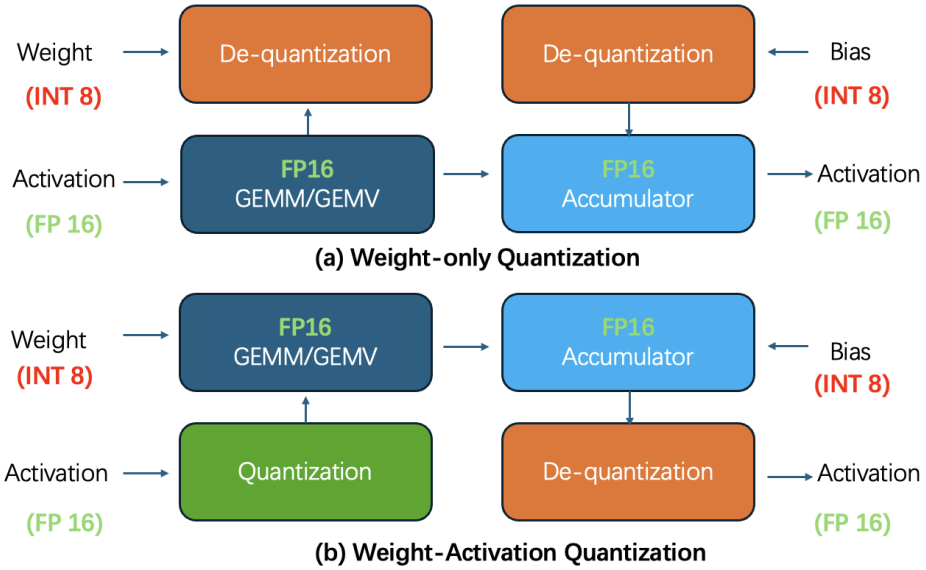


Figure 2.5: Inference flow for (a) Weight-Only Quantization and (b) Weight-Activation Quantization.

weight-activation quantization impacts both memory efficiency and model performance, with weight-only quantization often preferred for maintaining accuracy in LLMs.

2.3 State-of-the-Art PTQ Weight-Only Quantization Methods

This section provides an overview of recent PTQ weight-only quantization methods designed for LLMs, each focusing on reducing memory usage while preserving model accuracy.

2.3.1 OBQ and GPTQ

In weight-only quantization, the Optimal Brain Quantization (OBQ) [10] approach aims to minimize quantization error by selecting weights in a specific, greedy order. This section provides the key ideas of OBQ and show how GPTQ [11] improves its efficiency for large models.

OBQ

OBQ aims to minimize the distortion in neural network behavior during quantization. Consider a linear layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{d \times n}$ and an input matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, where d is the number of features. The goal of OBQ is to iteratively compute dequantized weight values and update the remaining unquantized values in a manner that minimizes the error in the output of the linear transformation.

To perform this, OBQ applies a column-wise iterative procedure to compute the dequantized values for each column $\widetilde{\mathbf{w}}_i$ of $\widetilde{\mathbf{W}}$. The process is constrained by the bit-width N and the derived parameters S_i and Z_i, which depend on the original weight values. For non-symmetric quantization, these parameters are calculated as:

$$S_i = \frac{w_{i,\max} - w_{i,\min}}{2^N - 1}, \tag{2.21}$$

$$Z_i = \text{round}\left(\frac{w_{i,\min}}{S_i}\right), \tag{2.22}$$

where $w_{i,\max}$ and $w_{i,\min}$ are the maximum and minimum values of the weights in column $\mathbf{w}_i$, respectively. Using these parameters, the dequantized grid $\mathcal{G}_i$ for column i is defined as:

$$\mathcal{G}_i = \left\{ g \mid g = (z + Z_i) \cdot S_i,\ z \in \{0, 1, \ldots, 2^N - 1\} \right\},$$

At each iteration, OBQ selects an element from column $\mathbf{w}_i$ to be quantized. The selection process follows a greedy strategy, choosing the element that, when quantized, minimizes its contribution to the overall error, i.e., find the row index q^* such that

$$q^* = \underset{q}{\text{argmin}} \frac{\left(w_{q,i} - \widetilde{w}_{q,i}\right)^2}{\left(\mathbf{H}^{-1}\right)_{q,q}}, \tag{2.23}$$

where $\mathbf{H} = 2\mathbf{X}^\mathsf{T}\mathbf{X}$, $\widetilde{w}_{q,i} \in \mathcal{G}_i$ is the dequantized value of $w_{q,i}$, produced by (see Equation (2.16))

$$\widetilde{w}_{q,i} = \left(\widetilde{w}_{q,i_Q} + Z_i\right) \cdot S_i,$$

with the quantized integer value $\widetilde{w}_{q,i_Q}$ computed as (see Equation (2.13))

$$\widetilde{w}_{q,i_Q} = \text{round}\left(\frac{w_{q,i}}{S_i} - Z_i\right).$$

Once $w_{q,i}$ is quantized, the quantization error is compensated by updating the remaining unquantized elements in $\mathbf{w}_i$. The update is computed as:

$$\boldsymbol{\delta} = -\frac{w_{q,i} - \tilde{w}_{q,i}}{(\mathbf{H}^{-1})_{q,q}} \cdot \left(\mathbf{H}^{-1}\right)_{:,q}, \tag{2.24}$$

where $\boldsymbol{\delta}$ represents the vector of updates to all unquantized elements in $\mathbf{w}_i$. The updated value for the remaining unquantized elements w_r is given by:

$$w_{r,i} := w_{r,i} + \delta_r, \tag{2.25}$$

where r is any row index of the unquantized elements in $\mathbf{w}_i$.

This iterative process continues until all elements in $\mathbf{w}_i$ are quantized. While not globally optimal, the greedy approach enables a computationally efficient solution for quantizing the weights.

The iterative nature of OBQ involves frequent updates to the inverse Hessian and sensitivity evaluations for each weight element. For a single column, this complexity is $O(d^3)$, and for the entire weight matrix $\mathbf{W}$, it is $O(n \cdot d^3)$, where d is the number of rows and n is the number of columns. This limits its scalability for large-scale models.

GPTQ

GPTQ (Generalized Post-Training Quantization) enhances OBQ by addressing computational inefficiencies and scaling challenges in large models. It introduces three key innovations: arbitrary order quantization, lazy batch updates.

1. Arbitrary Order Quantization In OBQ, weights are quantized in a greedy order, selecting the weight with the smallest quantization error at each step (Equation (**??**)). However, this strategy offers minimal advantages for large models with millions of parameters. GPTQ simplifies this process by quantizing weights in a predefined, fixed order, removing the computational overhead of greedy selection while maintaining similar accuracy.

2. Lazy Batch Updates OBQ's element-by-element updates involve frequent error compensations (Equation (2.24)) that require memory-intensive matrix updates, causing inefficiencies on modern GPUs. GPTQ addresses this issue with a **lazy batch update strategy**, which groups updates for B columns (a "lazy batch") at a time. The process works as follows:

1. **Local Updates (Per Column):** Within the batch, each column is quantized element by element. For each quantized element, local updates are applied to the column using only the corresponding $B \times B$ block of the inverse Hessian $\mathbf{H}^{-1}$. This ensures efficient parallel computation across columns within the batch.

2. **Global Updates (After the Batch):** Once all columns in the batch are quantized, a global update is performed to propagate the cumulative quantization error to the entire weight matrix $\mathbf{W}$ and the full inverse Hessian $\mathbf{H}^{-1}$.

This two-step approach avoids frequent global updates and significantly improves GPU utilization by concentrating computations on smaller $B \times B$ blocks during local updates, followed by periodic global updates for accuracy. In practice, using a batch size $B = 128$ provides a good balance between computational efficiency and accuracy.

Algorithm The GPTQ algorithm for quantizing the weight matrix $\mathbf{W}$ using $\mathbf{H}^{-1}$ and a block size B is as follows:

Algorithm 2.1 GPTQ: Quantize Weight Matrix $\mathbf{W}$ Using Cholesky-Based Inverse $\mathbf{H}^{-1}$ and Block Size B

$\widetilde{\mathbf{W}} \leftarrow 0_{d_{\text{row}} \times d_{\text{col}}}$ ▷ Initialize quantized weight matrix

$\mathbf{E} \leftarrow 0_{d_{\text{row}} \times B}$ ▷ Initialize block quantization errors

Compute $\mathbf{H}^{-1}$ by Cholesky factorization: $\mathbf{H} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$, $\mathbf{H}^{-1} = \mathbf{L}^{-\mathsf{T}}\mathbf{L}^{-1}$

for $i = 0, B, 2B, \ldots, d_{\text{col}}$ **do**

  for $j = i, \ldots, i + B - 1$ **do**

    $\widetilde{\mathbf{W}}_{:,j} \leftarrow \text{quant}(\mathbf{W}_{:,j})$ ▷ Quantize column j to the nearest value

    $\mathbf{E}_{:,j-i} \leftarrow \frac{\mathbf{W}_{:,j} - \widetilde{\mathbf{W}}_{:,j}}{(\mathbf{H}^{-1})_{jj}}$ ▷ Compute quantization error

    $\mathbf{W}_{:,j:(i+B)} \leftarrow \mathbf{W}_{:,j:(i+B)} - \mathbf{E}_{:,j-i} \cdot (\mathbf{H}^{-1})_{j:(i+B),j}$ ▷ Update weights in block

  end for

  $\mathbf{W}_{:,(i+B):} \leftarrow \mathbf{W}_{:,(i+B):} - \mathbf{E} \cdot (\mathbf{H}^{-1})_{i:(i+B),(i+B):}$ ▷ Update remaining weights

end for

Summary GPTQ simplifies and accelerates OBQ by leveraging fixed-order quantization, batching updates, and stabilizing computations with Cholesky decomposition. These enhancements make GPTQ a scalable and efficient solution for post-training quantization in large-scale neural networks.

2.3.2 AWQ: Activation-Aware Weight Quantization

AWQ [26] is a weight-only quantization technique designed to address the varying importance of weight channels in large language models (LLMs). It particularly focuses on channels aligned with input features exhibiting activation outliers. By reparameterizing weights to preserve critical channels, AWQ improves quantization accuracy through selective scaling of important weights.

$$\widetilde{\mathbf{w}}_Q = \text{round}\left(\frac{\mathbf{w}}{S}\right), \tag{2.26}$$

where $S = \frac{\max(|\mathbf{w}|)}{2^{N-1}}$ is the scale factor, and N is the number of quantization bits. The corresponding dequantized weights $\widetilde{\mathbf{w}}$ are calculated as

$$\widetilde{\mathbf{w}} = S \cdot \widetilde{\mathbf{w}}_Q. \tag{2.27}$$

Here, we do not introduce a zero point Z, as it represents a fixed offset that does not affect the error analysis.

Consider a linear operation $y = \mathbf{x}^T\mathbf{w}$, where $\mathbf{x} \in \mathbb{R}^d$ is a column vector representing the input, and $\mathbf{w} \in \mathbb{R}^d$ is a column vector representing the weights.

Now consider a scalar weight w, which is an element of the weight vector $\mathbf{w}$. If we multiply w by $p > 1$, and inversely scale a scalar activation x, the new quantization scale S' is defined as

$$S' = \frac{\max\left(\max(|\mathbf{w} \setminus w|), |w \cdot p|\right)}{2^{N-1}}, \tag{2.28}$$

where $\mathbf{w} \setminus w$ denotes the rest of the elements in the weight vector $\mathbf{w}$ excluding the current w, and with scaling we have

$$\frac{x}{p} \cdot \widetilde{(w \cdot p)} = S' \cdot \frac{x}{p} \cdot \text{round}\left(\frac{w \cdot p}{S'}\right); \tag{2.29}$$

and without scaling we have

$$x \cdot \widetilde{w} = S \cdot x \cdot \text{round}\left(\frac{w}{S}\right). \tag{2.30}$$

Error Analysis The expected error from the rounding operation, denoted as $\text{roundErr}(\cdot)$, remains consistent across the scaling and non-scaling cases in Equations (2.29) and (2.30), as rounding simply maps a floating-point number to the nearest integer. This introduces a uniformly distributed error over the range $[0, 0.5]$ with an average value of 0.25:

$$\text{roundErr}(\cdot) \sim 0.25. \tag{2.31}$$

AWQ states that scaling a weight w by a factor S does not significantly alter the maximum value of the weight group $\mathbf{w}$, as the scaling factor is not excessively large. The dynamic range of the group remains dominated by the largest element, implying that $S' \approx S$.

Quantization error arises when w is mapped to a discrete set of representable values. This error propagates through operations involving the input x and the dequantized weight $\widetilde{w}$. The quantization error in the product xw is the difference between the exact value xw and the quantized value $x\widetilde{w}$:

$$\mathrm{Err}(x\widetilde{w}) = S \cdot x \cdot \mathrm{roundErr}\left(\frac{w}{S}\right),$$

where $\mathrm{roundErr}(\cdot)$ is given by Equation (2.31). Since S and x are not quantized and do not participate in the quantization process, they do not introduce additional rounding errors. Thus, the total quantization error is simply the rounding error scaled by S and x.

When scaling by a factor p, the quantization error becomes:

$$\mathrm{Err}\left(\frac{x}{p} \cdot \widetilde{(w \cdot p)}\right) = S' \cdot \frac{x}{p} \cdot \mathrm{roundErr}\left(\frac{w \cdot p}{S'}\right).$$

The ratio of the new error to the original error is given by:

$$\frac{\mathrm{Err}\left(\frac{x}{p} \cdot \widetilde{(w \cdot p)}\right)}{\mathrm{Err}(x\widetilde{w})} = \frac{S'}{S} \cdot \frac{1}{p}.$$

Since $S' \approx S$ and $p > 1$, the relative error decreases with scaling, indicating that scaling improves quantization accuracy for important weights.

AWQ leverages the observation that a small subset of weights, referred to as **salient weights**, significantly influences model performance. Salient weights are those that correspond to channels with high activation magnitudes, as errors in these weights are more likely to degrade model accuracy. A channel, in this context, refers to a row in the weight matrix $\mathbf{W}$, which corresponds to a column in the input activation matrix $\mathbf{X}$. These channels are identified by analyzing the input activations from a calibration dataset.

For example, if the activation values in a particular column of $\mathbf{X}$ are large, the corresponding row in $\mathbf{W}$ is considered salient. These salient weights play a critical role in the model's computations and require special attention during quantization to preserve model accuracy. AWQ addresses this challenge by scaling the salient weights, which has been shown to significantly improve model performance by reducing quantization errors for critical channels. To balance the errors between salient and non-salient weights, AWQ employs a data-driven search to optimize the scaling factor p for each input channel (row).

The optimization is performed using the following objective function:

$$L(\mathbf{p}) = \left\| \text{diag}(\mathbf{p})^{-1} \cdot \mathbf{X} \cdot \mathbf{W} \widetilde{\cdot \text{diag}}(\mathbf{p}) - \mathbf{XW} \right\|_F, \tag{2.32}$$

where $\mathbf{X}$ is the input matrix from a calibration dataset, $\mathbf{W}$ is the original weight matrix, $\text{diag}(\mathbf{p})$ represents a diagonal matrix formed by the per-channel scaling factors in the vector $\mathbf{p}$, and $\mathbf{W} \widetilde{\cdot \text{diag}}(\mathbf{p})$ is the dequantized weight matrix after scaling and quantization-dequantization. The function $L(\mathbf{p})$ quantifies the error introduced by the quantization and scaling process.

To find the optimal scaling factor vector $\mathbf{p}$, AWQ employs a grid search, systematically evaluating $L(\mathbf{p})$ over a predefined set of candidate scaling factors and selecting the configuration that minimizes the error. This approach ensures that salient weights are appropriately scaled, reducing quantization errors while maintaining computational efficiency.

The scaling factor for each channel is defined as:

$$p_i = p_{X,i}^{\alpha}, \tag{2.33}$$

where $p_{X,i}$ is the average magnitude of activations for channel i, derived from the calibration dataset, and the parameter α is a scalar hyperparameter shared across all channels, controlling the degree of scaling globally.

The optimal α is determined by minimizing the objective function:

$$\alpha^* = \arg\min_{\alpha} L(\mathbf{p}_X^{\alpha}), \tag{2.34}$$

which implies that $\mathbf{p}_X$ has been fixed and only α is the variable. Here, $\mathbf{p}_X^\alpha = [p_{X,1}^\alpha, p_{X,2}^\alpha, \ldots, p_{X,k}^\alpha]^T$, where k is the number of rows of $\mathbf{X}$, representing the number of channels. The parameter α is typically searched over the range $[0, 1]$.

To further reduce the mean squared error (MSE) during quantization, AWQ applies weight clipping after determining the scaling factors p, ensuring that quantized weights remain robust to activation outliers. By adaptively scaling critical weights, AWQ effectively reduces quantization errors, particularly in salient weight channels aligned with activation outliers.

This strategy enables AWQ to achieve robust quantization performance at low-bit precision levels, such as 3-bit and 4-bit quantization, while maintaining a balance between compression efficiency and model accuracy.

2.3.3 OmniQuant: Block-Wise Quantization with Learnable Weight Clipping

OmniQuant [42] introduces a block-wise quantization approach for large language models (LLMs) to minimize quantization errors efficiently. Previous post-training quantization (PTQ) methods with gradient optimization, such as AdaRound [32] and BRECQ [25], become impractical for models with billions of parameters due to the vast solution space. Instead of optimizing the entire model, OmniQuant performs block-wise quantization, which reduces computational complexity and enables optimization on a single GPU.

The original OmniQuant method quantizes both activation values $\mathbf{X}$ (using Learnable Equivalent Transformation, LET) and weights $\mathbf{W}$ (using Learnable Weight Clipping, LWC). However, since this work focuses on weight-only quantization, we introduce LWC in detail and omit LET from further discussion.

Block-Wise Quantization Error Minimization

OmniQuant optimizes quantization error by dividing weights into manageable block-wise components. For each layer, the optimization objective is:

$$\min_{\{\gamma_i,\beta_i\}} \left\| \mathcal{F}^{(l)}\left(\widetilde{\mathbf{W}}, \mathbf{X}\right) - \mathcal{F}^{(l)}\left(\mathbf{W}, \mathbf{X}\right) \right\|_F^2. \tag{2.35}$$

Explanations about the loss function are given below. Here $\mathcal{F}^{(l)}$ represents the function of the transformer block l, which transforms the input $\mathbf{X}$ using weights $\mathbf{W}$, and it includes both the attention mechanism and the MLP block within a transformer. The attention block computes the query, key, and value projections and performs self-attention, while the MLP block applies a feed-forward transformation to the attention output. For detailed definitions of these components, refer to Section 2.1.1. In (2.35), $\widetilde{\mathbf{W}}$ consists of group-specific dequantized weights $\widetilde{\mathbf{W}}_i(\gamma_i, \beta_i)$, γ_i and β_i are learnable clipping parameters for group i, each group $\widetilde{\mathbf{W}_i}$ contains non-overlapping sets of dequantized weights within the layer.

The use of group-wise learnable parameters γ_i and β_i aligns the scaling factor s_i and zero-point z_i with the group structure. Instead of training γ and β for each individual weight element, OmniQuant controls quantization at the group level. This strategy improves feasibility and reduces optimization complexity while maintaining quantization accuracy.

Learnable Weight Clipping (LWC)

The Learnable Weight Clipping (LWC) mechanism optimizes the group-specific quantization process. For each group i, the quantized weights $\widetilde{\mathbf{W}}_i^Q(\gamma_i, \beta_i)$ are defined as

$$\widetilde{\mathbf{W}}_i^Q(\gamma_i, \beta_i) = \text{clamp}\left(\text{round}\left(\frac{\mathbf{W}_i}{s_i}\right) + z_i, 0, 2^N - 1\right), \tag{2.36}$$

where

$$s_i = \frac{\gamma_i \cdot \max(\mathbf{W}_i) - \beta_i \cdot \min(\mathbf{W}_i)}{2^N - 1}, \quad z_i = -\text{round}\left(\frac{\beta_i \cdot \min(\mathbf{W}_i)}{s_i}\right). \tag{2.37}$$

The corresponding dequantized weights for group i, $\widetilde{\mathbf{W}}_i(\gamma_i, \beta_i)$, are computed as

$$\widetilde{\mathbf{W}}_i(\gamma_i, \beta_i) = s_i \cdot \left(\widetilde{\mathbf{W}}_i^Q(\gamma_i, \beta_i) - z_i \right). \tag{2.38}$$

This group-wise clipping mechanism optimizes the quantization process by learning the parameters γ_i and β_i, which control the scaling and zero-point adjustments for each weight group. By doing so, LWC minimizes the quantization error within each group while aligning the group-level optimization with the overall block-wise minimization objective described in the previous subsection.

Discussion, Feasibility, and Benefits of Learnable Weight Clipping

OmniQuant achieves a balance between accuracy and computational feasibility by grouping weights and aligning the clipping parameters γ_i and β_i within each group. This group-wise optimization ensures that the scaling factor s_i and zero-point z_i remain consistent within a group, reducing the complexity of training individual parameters for every weight element. Instead of fine-tuning weights directly, OmniQuant optimizes only the clipping strengths γ_i and β_i, making the quantization process computationally efficient while preserving accuracy.

Learnable Weight Clipping (LWC) dynamically adjusts the quantization range to minimize errors. When $\gamma = 1$ and $\beta = 1$, LWC reduces to traditional Min-Max quantization, as used in other quantization approaches [11]. By learning γ_i and β_i, LWC selectively preserves the most important weight information, ensuring effective quantization even at extremely low-bit settings.

This approach eliminates the computational burden of backpropagating gradients for large weight matrices, making Quantization-Aware Training (QAT) feasible for large-scale LLMs with billions of parameters. While OmniQuant originally included both LWC for weight-only quantization and LET for weight-activation quantization, this discussion focuses solely on LWC. This aligns with the primary goal of optimizing weight-only quantization to achieve low-bit representations while maintaining model accuracy and efficiency.

Chapter 3

Two Step Power-of-Two Quantization Method

In this chapter we present a novel approach for quantizing large language models (LLMs) using power-of-two (PoT) formats, addressing the challenges posed by extremely low-bit scenarios and the need for efficient dequantization strategies. The chapter begins by analyzing the performance limitations of current state-of-the-art (SOTA) quantization methods for LLMs in Section 3.1.1 and highlighting the necessity of developing quantization methods that are not only suitable for low-bit settings but also facilitate faster inference through efficient dequantization techniques in Section 3.1.2. The motivation for adopting PoT quantization is then presented in Section 3.2, establishing the advantages of aligning weight distributions with PoT formats.

The chapter further explores the inability of existing SOTA quantization methods to directly quantize weights into PoT formats in Section 3.3.1 and defines the optimization objectives that underpin the proposed method in Section 3.3.1. Subsequently, a detailed description of the proposed two-step PoT quantization method is presented in Section 3.3.2, including both data-agnostic scale initialization in Section 3.3.2 and data-dependent fine-tuning in Section 3.3.2 for layer-wise reconstruction. Finally, the chapter describes an efficient dequantization strategy tailored for PoT quantized weights in Section 3.3.3.

3.1 Assessment of Low-Bit Quantization Methods for LLaMa Models

This section aims to highlight the weaknesses of state-of-the-art quantization methods, such as GPTQ, AWQ, and OmniQuant, when applied to LLaMa models, and intuitively motivate the need for a novel quantization method that addresses these issues.

3.1.1 Quantization Performance Summary for LLaMa Models

Avg Bits	Model	Size	RTN	GPTQ	AWQ	OMNI
4	LLaMa1	7B	5.96	5.85	5.81	5.77
		13B	5.20	5.20	5.25	5.18
		30B	4.23	4.21	4.23	4.19
	LLaMa2	7B	5.72	5.61	5.62	5.59
		13B	4.98	4.98	4.97	4.96
3	LLaMa1	7B	7.01	6.55	6.46	6.16
		13B	5.88	5.62	5.51	5.46
		30B	4.87	4.80	4.63	4.58
	LLaMa2	7B	6.66	6.29	6.24	6.21
		13B	5.51	5.42	5.32	5.28
2	LLaMa1	7B	1.90e3	44.01	2.60e5	9.77
		13B	781.2	15.6	2.80e5	7.93
		30B	68.04	10.92	2.40e5	7.13
	LLaMa2	7B	4.20e3	36.77	2.20e5	11.23
		13B	122.08	28.14	1.20e5	8.33

Table 3.1: WikiText-2 perplexity of 2-, 3-, and 4-bit quantized LLaMa family models (LLaMa-1 and LLaMa-2) using the studied quantization methods. Results are based on group-wise quantization with a group size of 128. Lower perplexity values indicate better model performance, and the table highlights the performance of Round-to-Nearest (RTN), GPTQ, AWQ, and OmniQuant across different bit levels.

Table 3.1 summarizes the perplexity scores for 2-bit, 3-bit, and 4-bit quantized versions of the LLaMa models, including LLaMa-1 and LLaMa-2, developed by Meta [46, 47], using various state-of-the-art quantization methods. Perplexity is a metric used to measure the predictive accuracy of language models, with lower perplexity indicating better model performance.

It is defined as the exponentiated average negative log-likelihood of the predicted probabilities of the model [41].

The results are derived using WikiText-2, a widely used benchmark dataset for evaluating language model performance on natural language processing tasks. WikiText-2 is a curated dataset of Wikipedia articles, specifically designed for language modeling tasks, and contains approximately 2 million tokens of clean and well-structured text. Its focus on high-quality English content makes it ideal for assessing the generalization ability of language models. The “Size" in the table gives the number of model parameters in billions.

The table summarizes key findings about the methods that previously reviewed in Section 1.2.2. We briefly recall their main characteristics here for context:

- **GPTQ** provides effective low-bit quantization down to 3-bit, outperforming standard round-to-nearest (RTN) quantization by minimizing reconstruction errors using second-order Hessian adjustments. However, the second-order Hessian inverse update in GPTQ is limited in precision and risks overfitting to the calibration data. This may compromise the robustness of the model when the quantized weights ($\widetilde{\mathbf{W}}$) deviate significantly from the original weights ($\mathbf{W}$).

- **AWQ** achieves more robust quantization by employing a mixed-precision approach that preserves the most important (salient) weights in higher precision, thereby enhancing model quality at 3-bit and 4-bit levels. However, at extreme low-bit (2-bit) levels, the effectiveness of AWQ decreases substantially, leading to significant degradation in perplexity.

- **OmniQuant** introduces a learnable weight clipping strategy to refine the clipping bounds of weights. This enables 2-bit quantization with minimal perplexity impact, demonstrating OmniQuant’s adaptability for very low-bit quantization while maintaining higher accuracy. However, there remains potential for improvement in both the 2-bit perplexity and the dequantization speed, suggesting further exploration for enhanced efficiency.

3.1.2 Need for Efficient Low-Bit Quantization with Fast Dequantization

Table 3.1 demonstrates the importance of developing quantization methods that can achieve extreme low-bit (2-bit and 3-bit) quantization while preserving model accuracy. Additionally, the need for faster dequantization is essential for practical deployment, as conventional mixed-precision multiplication operations can significantly slow down inference.

To meet these requirements, an effective quantization method should:

1. Provide robust 2-bit and 3-bit quantization without compromising model quality.
2. Enable rapid dequantization for efficient inference, reducing reliance on costly mixed-precision multiplications.

Developing a method that fulfills both requirements would significantly benefit the deployment of large language models by balancing model compression with computational efficiency.

3.2 Motivation for Power-of-Two Quantization in LLMs

Power-of-two (PoT) quantization [9,21,22,24,56] is a technique that maps weights to discrete values that are powers of two, enabling efficient computation through bit-shifting operations. In its traditional form, the quantized values are selected from the set:

$$\{\pm 0, \pm 2^0, \pm 2^1, \ldots, \pm 2^{q_{\max}}\},$$

where $q_{\max} = 2^{N-1} - 1$ for the quantization of N bits. This set provides a symmetric range of values around zero and is widely used for hardware efficiency. By mapping weights to this discrete set, PoT quantization replaces floating-point multiplications with simple shift operations, significantly reducing computational overhead [31].

However, recent advances, such as DenseShift [21], have introduced a modified version of PoT quantization that excludes zero from the set, leaving only the non-zero values:

$$\{\pm 2^0, \pm 2^1, \ldots, \pm 2^{q_{\max}}\}.$$

This adjustment is particularly beneficial in low-bit scenarios by maximizing usable quantization levels and reducing the need for handling zero, which can streamline dequantization. In our method, we adopt this zero-exclusion approach as it aligns well with the motivations for PoT quantization in large language models (LLMs).

3.2.1 Alignment with Weight Distribution in LLMs

Deep Neural Network weights generally exhibit a bell-shaped, long-tailed distribution [15], as opposed to a uniform distribution. Power-of-two quantization naturally aligns with this distribution, especially in lower-bit settings, as it provides denser quantization levels near zero and sparser levels as values increase. This exponential spacing allows for greater precision in representing smaller values, which are more frequent in LLM weights.

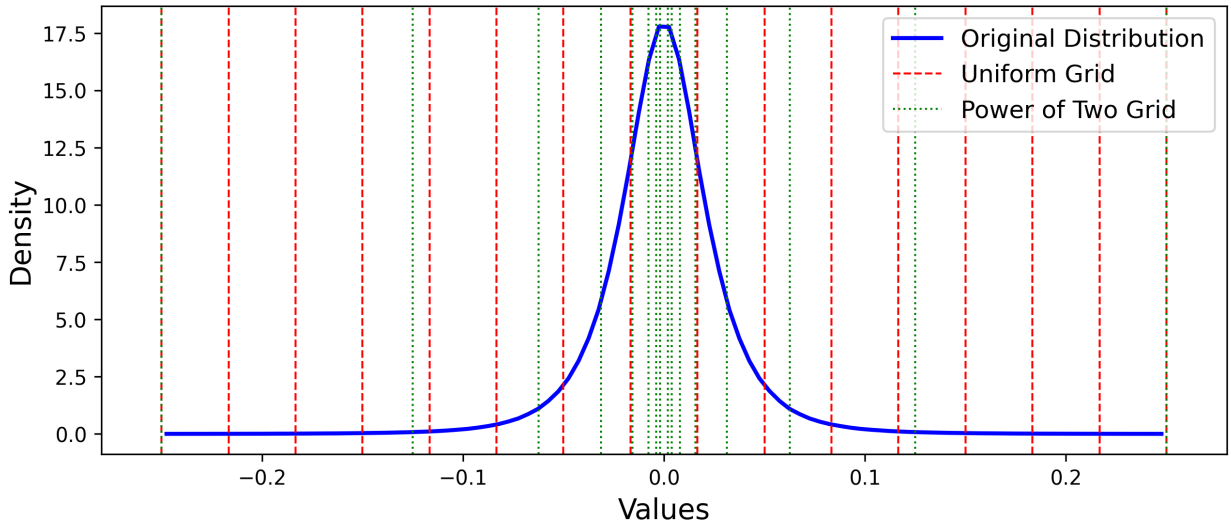
By choosing a zero-exclusion approach, we maximize the quantization levels allocated to non-zero values, enhancing the quantization resolution in low-bit scenarios. This setup enables our method to more accurately represent weights while minimizing quantization errors, as illustrated in Figures 3.1a and 3.1b.

3.2.2 Efficient Dequantization for Inference Acceleration

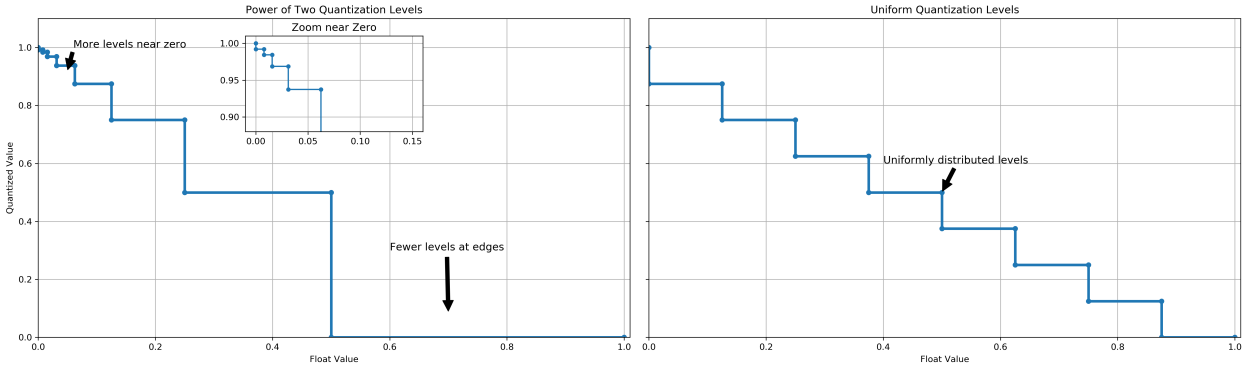
In weight-only quantization, quantized weights must be converted back to FP16 format to meet GEMM (General Matrix Multiply) requirements during inference. While uniform quantization typically relies on floating-point multiplication for dequantization (see Equation (2.16)), which can be computationally expensive, power-of-two quantization enables faster dequantization by replacing the multiplication of power-of-two values in Equation (2.19) with efficient bit-shifting operations. This method significantly reduces computational overhead.

Since the quantization scheme used here is symmetric, the absence of a zero-point Z further reduces memory usage and simplifies the dequantization process by eliminating the need for addition.

This optimized technique is particularly advantageous in low-bit quantization settings and is described in greater detail in Section 3.3.3.



(a) Density distribution of the first matrix from LLaMa 7B, showcasing min-max uniform and power-of-two grids. The power-of-two grid is closely aligned with the parameter distribution near zero.



(b) Quantization levels for uniform and non-uniform (PoT) quantization.

Figure 3.1: Illustrations of LLM weight distribution and power-of-two quantization alignment. Power-of-two quantization aligns closely with LLM weight characteristics, especially in the low-bit setting, enhancing accuracy.

3.2.3 Conclusion

Power-of-two quantization, particularly in its zero-exclusion form, is well-suited for low-bit quantization in LLMs, offering two main advantages:

1. **Distribution Alignment:** Power-of-two quantization closely aligns with the natural distribution of LLM weights, accurately representing frequently occurring small values and optimizing bit usage.

2. **Efficient Dequantization:** By using shift-based dequantization, power-of-two quantization accelerates inference, addressing the computational inefficiencies inherent in low-bit quantization.

These advantages make power-of-two quantization a strong choice for low-bit quantization in LLMs. In the following section, we propose a specific algorithm to implement this quantization strategy effectively.

3.3 Proposed method

3.3.1 Explore and Preliminaries

Exploration of Existing PoT Quantization Methods

We explore and implement state-of-the-art (SOTA) quantization methods that utilize power-of-two (PoT) formats for quantizing large language models (LLMs). Our experiments reveal that the value-based approaches, such as GPTQ and AWQ, perform poorly in extremely low-bit settings, particularly at 2-bit precision. In contrast, OmniQuant, a scale-based method, shows relatively better results, though its performance remains suboptimal for practical deployment.

Value-based methods like GPTQ and AWQ directly adjust the quantized values to minimize the quantization error. However, this strategy struggles in extremely low-bit scenarios due to the lack of sufficient precision to faithfully represent weights. OmniQuant,

on the other hand, adopts a scale-based approach, optimizing scaling factors to preserve model accuracy across shared weights. This shared scaling simplifies representation and enables gradient-based optimization techniques, allowing for partially quantization-aware training on a small calibration dataset. By fine-tuning these scaling factors, OmniQuant achieves better robustness compared to value-based methods but still faces limitations in extremely low-bit settings.

Table 3.2 presents the performance (perplexity score) of state-of-the-art methods (AWQ, GPTQ, and OmniQuant) under power-of-two (PoT) quantization, where the weights are constrained to power-of-two formats. While these methods were originally designed for general low-bit quantization, adapting them to PoT constraints highlights significant limitations.

The results reveal that none of the methods achieve successful 2-bit quantization in the PoT format. Specifically, the lowest perplexity exceeds 100, indicating that the quantized models fail to generate coherent or readable outputs—an outcome deemed unacceptable for practical applications.

Given these limitations, we adopt a scale-optimization approach for our PoT quantization. By leveraging shared scaling factors and efficient gradient-based fine-tuning, this approach aims to enhance robustness and efficiency, particularly in extremely low-bit quantization settings for LLMs. Detailed analysis and evidence for this method are provided in Section 3.3.1.

Avg Bits	Model	Size	AWQ_POT	GPTQ_POT	OMNI_POT
3.125	LLaMa1	7B	6.52	8.27×10^4	**6.37**
		13B	5.61	5.85×10^4	**5.60**
		30B	4.72	2.61×10^4	**4.75**
	LLaMa2	7B	6.49	NAN	**6.46**
		13B	5.43	6.41×10^4	**5.45**
2.125	LLaMa1	7B	2.69×10^5	2.92×10^5	**888**
		13B	2.80×10^5	1.83×10^5	**487**
		30B	2.39×10^5	1.44×10^5	**297**
	LLaMa2	7B	2.24×10^5	2.78×10^5	**3730**
		13B	1.27×10^5	1.03×10^5	**812**

Table 3.2: Performance in perplexity comparison of various quantization methods on different LLaMa models and sizes, with a group size of 128.

Optimization Objectives

Our approach aims to balance two objectives: aligning the dequantized and original weights while minimizing output discrepancies in the quantized model.

To achieve this, we define two optimization objectives. At this stage, we provide a high-level overview, with specific details on Step 1 and Step 2 presented in Section 3.3.2.

1. **Weight Reconstruction Objective:** The primary goal is to minimize the discrepancy between the dequantized weights and the original weights. This objective ensures that the quantization process is more robust and not overly reliant on calibration data to avoid the overfitting issue. Formally, we define this objective as:

$$\min_{\mathbf{S}^{(l)}} \left\| \mathbf{W}^{(l)} - \widetilde{\mathbf{W}}^{(l)} \right\|_F^2, \tag{3.1}$$

where $\widetilde{\mathbf{W}}_{ij}^{(l)}$, the dequantized weight at position (i, j) at layer l, is defined as

$$\widetilde{\mathbf{W}}_{ij}^{(l)} = \mathbf{S}_{ij}^{(l)} \cdot \mathbf{P}_{ij} \cdot 2^{\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}}. \tag{3.2}$$

Here the factors on the right hand side of (3.2) are explained as follows:

- The scaling factor matrix $\mathbf{S}^{(l)}$ enforces shared scaling within groups of size G (e.g., $G = 128$) in each column. Specifically,

$$\mathbf{S}_{ij}^{(l)} = \mathbf{S}_{kj}^{(l)}, \quad \text{if } \lfloor i/G \rfloor = \lfloor k/G \rfloor, \tag{3.3}$$

  Here i and k are row indices of the elements in the j-th column in the same group sharing the same scaling factor.

- The sign matrix $\mathbf{P}$ represents the sign of each element in the original weight matrix $\mathbf{W}^{(l)}$. It is defined element-wise as:

$$\mathbf{P}_{ij} = \begin{cases} 1 & \text{if } \mathbf{W}_{ij}^{(l)} \geq 0, \\ -1 & \text{if } \mathbf{W}_{ij}^{(l)} < 0. \end{cases}$$

- The quantized weight $\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}$ is obtained by applying the scaling factor $\mathbf{S}_{ij}^{(l)}$ to the original weight $\mathbf{W}_{ij}^{(l)}$ and quantizing it within the range $[0, 2^{N-1} - 1]$. Using the 'clamp' function, the process is expressed as:

$$\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)} = \mathrm{clamp}\left(\mathrm{round}\left(\log_2\left(\frac{\left|\mathbf{W}_{ij}^{(l)}\right|}{\mathbf{S}_{ij}^{(l)}}\right)\right), 0, 2^{N-1} - 1\right), \tag{3.4}$$

where

$$\mathrm{clamp}(a, I_{\min}, I_{\max}) = \begin{cases} I_{\min} & \text{if } a < I_{\min}, \\ a & \text{if } I_{\min} \leq a \leq I_{\max}, \\ I_{\max} & \text{if } a > I_{\max}. \end{cases} \tag{3.5}$$

This ensures that the quantized value $\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}$ is within the valid range $[0, 2^{N-1} - 1]$.

2. **Output Alignment Objective:** The second objective is to minimize the difference in output between the quantized and original models by fine-tuning a deviation parameter matrix $\mathbf{\Gamma}$, which adjusts the scaling factors obtained from the first step. Unlike the weight alignment optimization (step 1, Equation (3.1)), which focuses on optimizing the scales ($\mathbf{S}^{(l)}$), this step further refines the scales by introducing $\mathbf{\Gamma}$. The optimization problem is defined as:

$$\min_{\mathbf{\Gamma}} \left\| \mathcal{F}^{(l)}(\mathbf{W}^{(l)}, \mathbf{X}) - \mathcal{F}^{(l)}(\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma}), \mathbf{X}) \right\|_F^2 + \frac{\lambda}{2} \|\mathbf{\Gamma}\|_F^2, \tag{3.6}$$

Here, $\mathcal{F}^{(l)}$ represents the layer function for layer l, which transforms the input $\mathbf{X}$ using the layer weights. This serves as a general framework for the transformation applied by the layer. $\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma})$ represents the refined quantized weights, which incorporate $\mathbf{\Gamma}$.

To ensure consistency, $\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma}) = \widetilde{\mathbf{W}}^{(l)}$ when $\mathbf{\Gamma} = 0$. Here, $\mathbf{\Gamma} = 0$ means that every entry of $\mathbf{\Gamma}$ is zero. This indicates that $\mathbf{\Gamma}$ quantifies the deviation of $\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma})$ from $\widetilde{\mathbf{W}}^{(l)}$. The precise definition and formulation of $\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma})$ will be provided in Section 3.3.2. Minimizing this objective balances two goals: aligning the outputs of the original and quantized models, while keeping $\mathbf{\Gamma}$ small to limit deviations.

Constraint on $\mathbf{\Gamma}$: The adjustment parameter $\mathbf{\Gamma}$ must satisfy a group-wise sharing constraint, similar to the scaling factors $\mathbf{S}^{(l)}$. Specifically:

$$\Gamma_{ij} = \Gamma_{kj}, \quad \text{if } \lfloor i/G \rfloor = \lfloor k/G \rfloor, \tag{3.7}$$

where G is the group size. This constraint ensures that all elements within the same group share the same adjustment parameter, preserving the group-wise structure critical for efficiency.

Together, these objectives form a robust framework for PoT quantization, ensuring that both the weights and outputs of the quantized model remain closely aligned with their original counterparts. In our proposed two-step approach, we first optimize the weight reconstruction objective to initialize scaling factors in a data-agnostic manner. Subsequently, we refine these factors using a small calibration set to minimize the output alignment error, thereby improving quantization accuracy and stability.

3.3.2 Proposed Two-Step Quantization Method

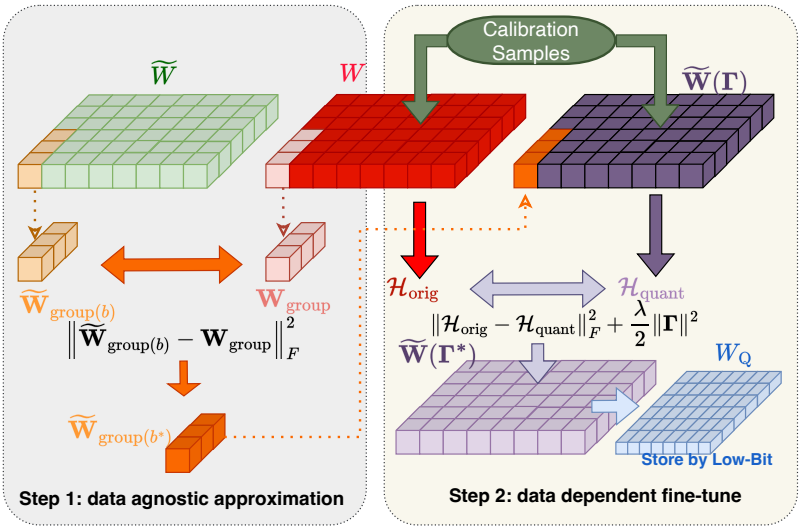


Figure 3.2: Overview of the proposed two-step quantization method. The first step involves data-agnostic scale initialization, while the second step refines the scaling using activation data from a calibration set.

Step 1: Data-Agnostic Scale Initialization

To initialize our quantization process, we focus on optimizing the weight reconstruction objective for each quantization group, aiming to minimize the weight alignment error as defined in Equation (3.1).

As explained in Equation (3.3), each column of the weight matrix $\mathbf{W}^{(l)}$ is partitioned into groups of G elements, where G denotes the group size. In our numerical experiments, we take $G = 128$ and $G = 64$ to evaluate performance under different group configurations. where each group shares the same scale factor $\mathbf{S}_{ij}^{(l)}$. Let $\mathbf{W}_{\text{group},ij}$ represents the weight at position (i, j) within group.

The scale factor s for each group is initialized based on the maximum absolute value of the weights within the group. Specifically, the initial scale s_0 is calculated as:

$$s_0 = \frac{\max |\mathbf{W}_{\text{group}}|}{2^{q_{\max}} - 1}, \tag{3.8}$$

where $\mathbf{W}_{\text{group}}$ refers to a subvector of a column of $\mathbf{W}$, formed by elements belonging to the same group and $q_{\max} = 2^{n-1} - 1$ represents the maximum exponent level for n-bit quantization.

To further optimize the scaling factor, we employ a grid search over possible scaling adjustments. We define a set of scale adjustments $B = \{0.01 \times i : i = 1, \ldots, 200\}$ and evaluate each scaling factor by computing the mean squared error (MSE) between the original and dequantized weights for each group:

$$b^* = \underset{b \in B}{\operatorname{argmin}} Q_1(b), \quad Q_1(b) = \left\| \mathbf{W}_{\text{group}} - \widetilde{\mathbf{W}}_{\text{group}}(b) \right\|_F^2, \tag{3.9}$$

where $\widetilde{\mathbf{W}}_{\text{group}}(b)$ represents the dequantized weights computed using the adjusted scale $s_0 \cdot b$:

$$\widetilde{\mathbf{W}}_{\text{group},ij}(b) = s_0 \cdot b \cdot \mathbf{P}_{ij} \cdot 2^{\mathbf{W}_{\text{Q,group},ij}(b)}, \tag{3.10}$$

with the quantized exponents defined as

$$\mathbf{W}_{\text{Q,group},ij}(b) = \text{clamp}\left(\text{round}\left(\log_2 \left(\frac{|\mathbf{W}_{\text{group},ij}|}{s_0 \cdot b} \right) \right), 0, 2^{N-1} - 1 \right). \tag{3.11}$$

The optimal scaling factor for that group is $s^* = s_0 \cdot b^*$.

These group-specific optimal scaling factors collectively form the matrix $\mathbf{S}^{(l)}$, which serves as the optimal scale for the weight reconstruction objective defined in Equation (3.1). Figure 3.2 (left side, Step 1) illustrates the abstract flow of the method.

Insights on Grid Search for PoT Quantization

From Equation (3.8), the round-to-nearest (RTN) scaling factor provides a naive initialization for s_0. However, experiments show that the optimal scaling factor for many groups often deviates significantly from the RTN value, sometimes requiring much larger or smaller adjustments. This observation motivates the inclusion of a grid search over a wide range of potential adjustments to refine the scale beyond the initial s_0.

Furthermore, the error landscape for power-of-two quantization across the grid is not as smooth as that for uniform quantization. As shown in Figure 3.3, the loss surface often contains sharp transitions due to the discrete nature of power-of-two scaling. This characteristic can cause value-based methods to converge to suboptimal local minima. Our grid search strategy explicitly addresses this challenge by exhaustively exploring the search space, ensuring that the globally optimal scaling factor is selected for each group.

By incorporating this grid search into the quantization process, we achieve significantly improved alignment between the original and quantized weights, providing a strong foundation for subsequent optimization steps such as output alignment.

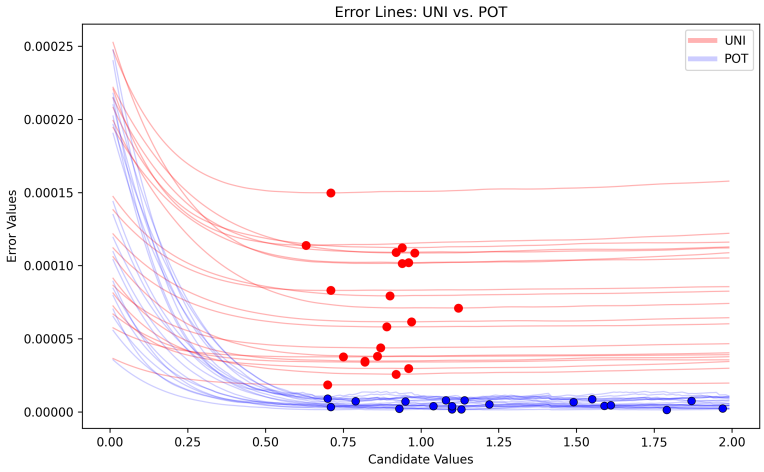


Figure 3.3: Error curves over 200 grid points in the range [0,2] for both uniform and power-of-two scaling. Power-of-two scale results demonstrate less smooth loss curves compared to uniform scaling.

Therefore, we selected a search range of $[0, 2]$ with a precision of 0.01 as a balanced trade-off between search speed and performance.

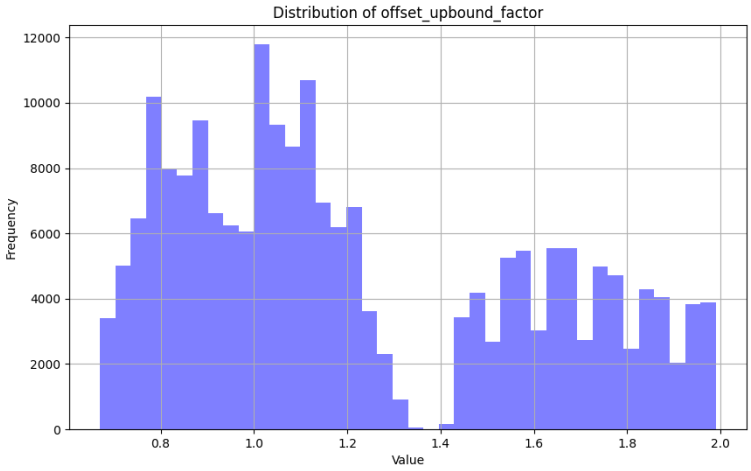


Figure 3.4: Distribution of optimal scaling values across 200 grid points in the range [0,2] for both uniform and power-of-two scaling. The results show that optimal power-of-two scaling values are widely distributed and not centered around 1, deviating from the naive RTN scale.

Figure 3.4 illustrates the distribution of optimal scaling values for both uniform and power-of-two scaling across 200 grid points within the range [0,2]. For power-of-two scaling, the distribution is spread out and lacks a concentration around 1, which is the naive RTN scale. This indicates that power-of-two scaling requires a broader search range to accurately capture the best scaling values, as these values frequently lie outside the predictions of traditional scaling approaches.

Another key advantage of this approach is that each group can be searched independently. This independence allows us to parallelize the grid search across all groups within a layer, rather than looping over each group sequentially. By leveraging parallelism, we significantly reduce the search time, achieving an efficient initial scale estimation for each group. This parallel search provides a robust base scaling factor s^* that sets up our second step, where we further fine-tune the scaling factors, with minimal computational overhead.

The grid search for the optimal scaling factor s^* is conducted in parallel across weight groups. The following algorithm illustrates this process:

Algorithm 3.1 Parallel Initial Scale Search

$s_0 \leftarrow \frac{\max|\mathbf{W}_{\text{group}}|}{2^{2^{n_{\text{bits}}-1}-1}}$ ▷ Initialize scale using the max weight in each group

$B \leftarrow \{0.01 \times i \mid i = 1, \ldots, 200\}$ ▷ Define a range of scaling adjustments

Initialize b^* for each group group ▷ Prepare to store optimal scaling adjustments

for each group **in parallel do**

  $Q_1^{\min} \leftarrow \infty$ ▷ Set the initial minimum MSE to infinity

  for each $b \in B$ **do**

    $s_b \leftarrow s_0 \cdot b$ ▷ Adjust the scale by the current scaling adjustment b

    $\widetilde{\mathbf{W}}_{\text{group}}(b) \leftarrow s_b \cdot \mathbf{P} \cdot 2^{\mathbf{W}_{\text{Q,group}}(b)}$ ▷ Dequantize using scale s_b

    $\mathbf{W}_{\text{Q,group}}(b) \leftarrow \text{clamp}\left(\text{round}\left(\log_2\left(\frac{|\mathbf{W}_{\text{group}}|}{s_b}\right)\right), 0, 2^{N-1}-1\right)$ ▷ Compute the quantized exponents

    $Q_1(b) \leftarrow \text{mean}\left(\left\|\mathbf{W}_{\text{group}_k} - \widetilde{\mathbf{W}}_{\text{group}}(b)\right\|_F^2\right)$ ▷ Calculate the MSE $Q_1(b)$

    if $Q_1(b) < Q_1^{\min}$ **then** ▷ Check if this adjustment yields a lower MSE

      $Q_1^{\min} \leftarrow Q_1(b)$ ▷ Update minimum MSE

      $b^* \leftarrow b$ ▷ Store the optimal scaling adjustment b for the group

    end if

  end for

  $s^* \leftarrow s_0 \cdot b^*$ ▷ Compute the optimal scale for the group

end for

Assemble all s^* into $\mathbf{S}^{(l)}$ ▷ Combine group-specific scales into the matrix $\mathbf{S}^{(l)}$

Step 2: Data-Dependent Fine-Tuning for Layer-Wise Quantization Reconstruction

In this step, we refine the scaling factor $\mathbf{S}_{ij}^{(l)}$ from its initial value obtained in Step 1 (Section 3.3.2). This fine-tuning process introduces a learnable parameter matrix $\mathbf{\Gamma}$ and uses calibration data to minimize the output alignment error, as defined in Equation (3.6). By optimizing this objective, we aim to reduce the discrepancy between the original and quantized model activations for each layer, further enhancing the robustness of the quantization process.

For clarity, we restate the optimization objective, previously introduced in Equation (3.6), as it is central to the current discussion. The objective, incorporating weight decay regularization, is defined as:

$$\min_{\mathbf{\Gamma}} Q_2(\mathbf{\Gamma}) = \left\| \mathcal{F}^{(l)}(\mathbf{W}^{(l)}, \mathbf{X}) - \mathcal{F}^{(l)}(\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma}), \mathbf{X}) \right\|_F^2 + \frac{\lambda}{2} \|\mathbf{\Gamma}\|^2, \tag{3.12}$$

where λ is a regularization term. $\mathcal{F}^{(l)}$ represents the function of the transformer block l, which transforms the input $\mathbf{X}$ using weights $\mathbf{W}$, it includes both the attention mechanism and the MLP block within a transformer. The attention block computes the query, key, and value projections and performs self-attention, while the MLP block applies a feed-forward transformation to the attention output. For detailed definitions of these components, refer to Section 2.1.1. The first term in $Q_2(\mathbf{\Gamma})$ captures the alignment between the output of the original model and the quantized model, while the second term penalizes large deviations in $\mathbf{\Gamma}$ to mitigate overfitting during fine-tuning.

The refined scaling factor $\hat{\mathbf{S}}_{ij}^{(l)}(\mathbf{\Gamma})$ is computed as:

$$\hat{\mathbf{S}}_{ij}^{(l)}(\mathbf{\Gamma}) = \mathbf{S}_{ij}^{(l)} \cdot (1 + \mathbf{\Gamma_{ij}}), \tag{3.13}$$

where $\mathbf{\Gamma_{ij}}$ represents the perturbation coefficient applied to the initial scaling factor $\mathbf{S}_{ij}^{(l)}$. Using this updated scaling factor, the dequantized weights $\widetilde{\mathbf{W}}_{ij}^{(l)}(\mathbf{\Gamma})$ are expressed as:

$$\widetilde{\mathbf{W}}_{ij}^{(l)}(\mathbf{\Gamma}) = \hat{\mathbf{S}}_{ij}^{(l)}(\mathbf{\Gamma}) \cdot \mathbf{P}_{ij} \cdot 2^{\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}(\mathbf{\Gamma})}, \tag{3.14}$$

where $\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}(\mathbf{\Gamma})$ is the quantized exponent value, computed as:

$$\widetilde{\mathbf{W}}_{\mathrm{Q},ij}^{(l)}(\mathbf{\Gamma}) = \mathrm{clamp}\left(\mathrm{round}\left(\log_2 \left(\frac{\left| \mathbf{W}_{ij}^{(l)} \right|}{\hat{\mathbf{S}}_{ij}^{(l)}(\mathbf{\Gamma})} \right) \right), 0, 2^{N-1} - 1 \right). \tag{3.15}$$

To enable gradient-based optimization despite the non-differentiable rounding operation in Equation (3.15), we apply the **Straight-Through Estimator (STE)**. In the backward

pass, the gradient of the rounding function is approximated as:

$$\frac{\partial \widetilde{\mathbf{W}}^{(l)}_{\mathrm{Q},ij}(\mathbf{\Gamma})}{\partial \hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma})} \approx \frac{\partial \left(\log_2 \left(\frac{\left|\mathbf{W}^{(l)}_{ij}\right|}{\hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma})} \right) \right)}{\partial \hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma})}. \tag{3.16}$$

The rounding operation is ignored during backpropagation, and the gradient of the continuous logarithmic term is used as a surrogate. Using this STE-based method, we can perform quantization-aware training on the calibration dataset while only storing and updating the minimal gradient associated with the parameter $\mathbf{\Gamma}$. This approach efficiently refines the scaling factor without requiring extensive modifications, leveraging only the minimal parameters necessary for calibration. Figure 3.2 (right side, Step 2) illustrates the abstract flow of the method. The detailed procedure for this fine-tuning is provided in Algorithm 3.2.

Algorithm 3.2 Fine-Tuning the Learnable Parameter $\mathbf{\Gamma}$ for Scaling Factors

Initialize: $\Gamma_{ij} \leftarrow 0 \quad \forall(ij)$

Set Parameters: Learning rate η, weight decay λ, epochs N

for epoch $= 1$ to N **do**

  for each input activation $\mathbf{X}$ in the calibration set **do**

    $\mathcal{H}_{\text{orig}} \leftarrow \mathcal{F}^{(l)}(\mathbf{W}^{(l)}, \mathbf{X})$ ▷ Compute original output

    $\hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma}) \leftarrow \mathbf{S}^{(l)}_{ij} \cdot (1 + \Gamma_{ij})$ ▷ Update scale

    $\widetilde{\mathbf{W}}^{(l)}_{\mathrm{Q},ij}(\mathbf{\Gamma}) \leftarrow \text{clamp}\left(\text{round}\left(\log_2 \left(\frac{\left|\mathbf{W}^{(l)}_{ij}\right|}{\hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma})} \right) \right), 0, 2^{N-1} - 1 \right)$

    $\widetilde{\mathbf{W}}^{(l)}_{ij}(\mathbf{\Gamma}) \leftarrow \hat{\mathbf{S}}^{(l)}_{ij}(\mathbf{\Gamma}) \cdot \mathbf{P}_{ij} \cdot 2^{\widetilde{\mathbf{W}}^{(l)}_{\mathrm{Q},ij}(\mathbf{\Gamma})}$ ▷ Dequantize weights

    $\mathcal{H}_{\text{quant}} \leftarrow \mathcal{F}^{(l)}(\widetilde{\mathbf{W}}^{(l)}(\mathbf{\Gamma}), \mathbf{X})$ ▷ Compute quantized output

    $Q_2(\mathbf{\Gamma}) \leftarrow \|\mathcal{H}_{\text{orig}} - \mathcal{H}_{\text{quant}}\|_F^2 + \frac{\lambda}{2}\|\mathbf{\Gamma}\|^2$ ▷ Compute loss

    $\Gamma_{ij} \leftarrow \Gamma_{ij} - \eta \cdot \frac{\partial Q_2(\mathbf{\Gamma})}{\partial \Gamma_{ij}}$ ▷ Update $\mathbf{\Gamma}$ via gradient descent

  end for

end for

3.3.3 Efficient Dequantization for Power-of-Two Quantized Weights

In our proposed quantization approach, we utilize a **Power-of-Two (PoT)** dequantization method, which offers computational efficiency by replacing floating-point multiplications with bitwise operations and integer additions. This section details the implementation of dequantization for PoT quantized weights and provides a comparison with traditional uniform quantization.

Power-of-Two (PoT) vs. Uniform Dequantization

For PoT quantization, each quantized weight $\widetilde{\mathbf{W}}^{(l)}$ is reconstructed directly from the quantized weight tensor $\widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)}$ and the scaling factor $\mathbf{S}^{(l)}$ as:

$$\widetilde{\mathbf{W}}^{(l)} = \mathbf{S}^{(l)} \circ \widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)}, \tag{3.17}$$

$$\widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)} = (-1)^{\mathsf{S}} \circ 2^{\mathsf{E}}, \tag{3.18}$$

where $\mathbf{S}^{(l)}$ represents the scaling factor for each quantization group. Here, $\widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)}$ encodes the quantized representation using a sign bit (S) and an exponent (E).

In contrast, traditional uniform quantization employs a non-symmetric dequantization approach:

$$\widetilde{\mathbf{W}}^{(l)} = (\widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)} + Z) \circ S, \tag{3.19}$$

where $\widetilde{\mathbf{W}}_{\mathrm{Q}}^{(l)}$ is the quantized value, Z is the zero-point offset, and S is the scaling factor.

The key distinction between the two approaches lies in computational efficiency. While uniform quantization relies on floating-point operations to apply both S and Z, PoT quantization eliminates the need for floating-point arithmetic. Instead, the exponent 2^{E} in Equation (3.18) can be efficiently implemented using bit-shifting operations, significantly speeding up the dequantization process in low-bit scenarios.

FP16 Format and Efficient Dequantization

To efficiently multiply PoT quantized weights by FP16 values, we leverage the structure of FP16 representation. Each FP16 value x consists of three parts: the sign bit S, the exponent bits E, and the mantissa bits M:

$$x = \overbrace{\mathsf{S}}^{\text{Sign}} \overbrace{\mathsf{E}_1, \ldots, \mathsf{E}_5}^{\text{Exponent}} \overbrace{\mathsf{M}_1, \ldots, \mathsf{M}_{10}}^{\text{Mantissa}}.$$

The full value of x is computed as:

$$x = (-1)^{\mathsf{S}} \times 2^{\mathsf{E}-15} \times 1.\mathsf{M}. \tag{3.20}$$

For a PoT quantized weight w in the format $\widetilde{w}_{\mathrm{Q}} = \mathsf{S}'\mathsf{E}'$ (where S' is the sign and E' is the exponent), the dequantized multiplication of x and w is computed as:

$$xw = (-1)^{\mathsf{S}+\mathsf{S}'} \times 2^{\mathsf{E}+\mathsf{E}'-15} \times 1.\mathsf{M}. \tag{3.21}$$

Dequantization Process: Computation Flow for PoT

The computation for PoT dequantization can be broken down into the following steps:

1. **Extract the Exponent Bits**: We isolate the exponent bits from the quantized weight $w_{\mathrm{Q}} = \mathsf{S}_0\mathsf{E}_1\mathsf{E}_2$ using an AND operation with a mask 000011 to retrieve the exponent bits $0000\mathsf{E}_1\mathsf{E}_2$.

2. **Combine Sign and Exponent**: Right-shift $\mathsf{S}\mathsf{E}_1\mathsf{E}_2$ by three positions to align the sign bit S_0 with the remaining exponent bits. We then apply an AND operation with 100000 to extract $\mathsf{S}00000$, followed by an OR operation to merge the sign and exponent bits into $\mathsf{S}_0 0000\mathsf{E}_1\mathsf{E}_2$.

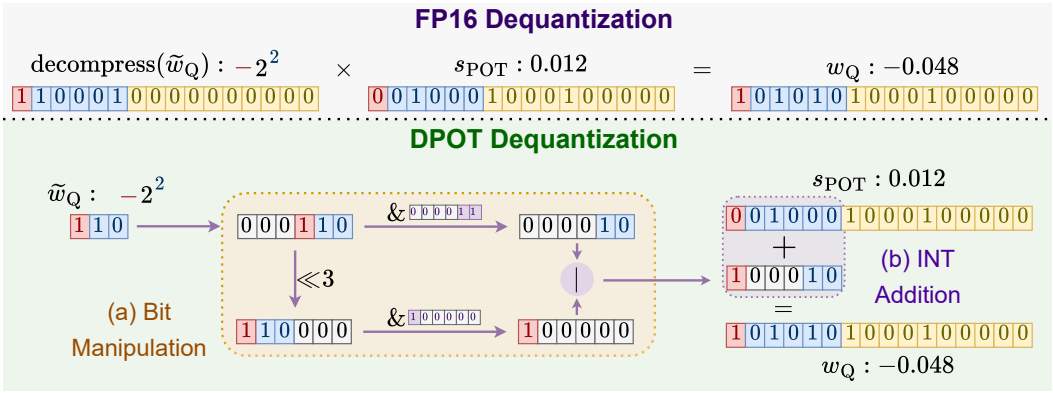


Figure 3.5: Multiplication between FP16 scale and power-of-two weights for a 3-bit example, implemented through efficient bit manipulation and fixed-point integer addition for accelerated PoT dequantization.

3. **Apply Scale**: The resulting bits $\mathsf{S}_0 0000 \mathsf{E}_1 \mathsf{E}_2$ are right-shifted by 10 bits and added to the scale s to produce the FP16 dequantized weight, completing the computation for each element in parallel.

Figure 3.5 illustrates the bitwise operations involved in the PoT dequantization process, demonstrating how the exponent and sign bits are manipulated for efficient dequantization.

Performance Evaluation

We tested our dequantization kernel on various GPU architectures to assess the efficiency of our PoT approach. Our implementation achieved a 3.6× faster warp cycle on the Tesla V100 and a 1.5× speedup on the NVIDIA RTX 4090 compared to uniform FP16 dequantization. These results underscore the practical benefits of PoT dequantization over traditional weight-only methods, particularly for large language model inference on GPU-accelerated systems.

In summary, our PoT dequantization process, by leveraging bitwise operations, is optimized for GPU parallelism, allowing efficient scaling and dequantization in large-scale models.

Chapter 4

Experiments and Results

In this chapter, we evaluate the effectiveness of our proposed **Power-of-Two (PoT)** quantization method for weight-only quantization in Large Language Models (LLMs). Our goal is to optimize the scale factor to enhance model accuracy at extremely low-precision quantization levels (2 or 3 bits). We integrate PoT quantization within the PyTorch framework using HuggingFace's LLaMA model family, specifically testing on LLaMA1 and LLaMA2 models at 7B, 13B, and 30B parameters [46,47]. All experiments are conducted on a single NVIDIA V100 GPU with 32GB of memory, which accommodates the entire quantization process. Additionally, we benchmark CUDA kernel warp cycles on both the NVIDIA RTX 4090 and NVIDIA Tesla V100 GPUs to compare the efficiency of PoT dequantization with traditional mixed-precision multiplication-based uniform dequantization on varying GPU architectures.

4.1 Experimental Settings

Our two-step quantization method includes the following configurations:

- In **Step 1** (Data-Agnostic Scale Initialization), a grid search is conducted over the range [0, 2] with a step size of 0.01 to find the optimal initial scale for each quantization group.

- In **Step 2** (Data-Driven Fine-Tuning), we use a calibration dataset of 128 randomly selected 2048-token segments from WikiText-2. The learning rate is set to 1×10^{-3} with a weight decay of 1×10^{-1}. For 3-bit quantization, we train for 10 epochs, while for 2-bit quantization, we use 40 epochs.

4.2 Perplexity Calculation

Perplexity is a standard metric for evaluating language models, measuring how well a model predicts a sample of text. It is commonly used in generation tasks and reflects the model's confidence in its predictions. For a language model with probability distribution P, the perplexity $\mathcal{P}$ over a dataset with N tokens is defined as:

$$\mathcal{P} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(x_i)\right), \tag{4.1}$$

where $P(x_i)$ is the predicted probability of the i-th token in the sequence. Lower perplexity indicates higher confidence and accuracy, as it implies the model assigns higher probabilities to the correct tokens. We evaluate perplexity on WikiText-2 [29] and C4 [38] to assess the language generation performance of our proposed PoT quantization compared to baselines.

4.3 Baselines

As the first approach to apply power-of-two quantization on LLMs, we benchmark PoT against several leading methods in the broader domain of uniform weight-only quantization. The baselines include:

- **Vanilla Rounding to Nearest (RTN)** [7]
- **GPTQ** [11]
- **AWQ** [26]

- **OmniQuant** [42]

Since methods focusing on weight activation quantization or mixed-precision quantization are not aligned with our objectives, they are excluded from our comparison. To ensure fair evaluation, all baselines are adapted to a power-of-two format where applicable. For uniform quantization baselines, we maintain a group size of 128 and set both scale and zero-point offsets to 16 bits, matching the average bit usage in our PoT approach.

4.4 Quantization Results on PoT and Uniform Quantization Baselines

In our experiments, we evaluate the performance of standard PTQ algorithms (GPTQ, AWQ, and OmniQuant) adapted to power-of-two (PoT) quantization with a group size of 128. When using PoT quantization across all methods, we maintain a group size of 128, while uniform quantization baselines use group sizes resulting in an average of 4.25 bits. For uniform methods with a 128 group size, 16-bit zero-point offsets allow flexibility that PoT quantization does not offer, as our method only employs symmetric scaling.

The following tables provide a detailed comparison of perplexity results across the WikiText-2, C4, and PTB datasets, demonstrating the effectiveness of our PoT quantization method in 2- and 3-bit scenarios. These results consistently highlight that PoT quantization, specifically tailored through our approach, surpasses traditional methods adapted to PoT and yields lower perplexity scores on challenging language modeling benchmarks.

4.4.1 WikiText-2 Quantization Comparison

Table 4.1 and Table 4.2 present the perplexity comparison across various LLaMa model sizes for both PoT and uniform quantization.

Avg Bits	Model	Size	AWQ_POT	GPTQ_POT	OMNI_POT	PoT
3.125	LLaMa1	7B	6.52	8.27×10^4	6.37	**6.25**
		13B	5.61	5.85×10^4	5.60	**5.50**
		30B	4.72	2.61×10^4	4.75	**4.58**
	LLaMa2	7B	6.49	NAN	6.46	**6.22**
		13B	5.43	6.41×10^4	5.45	**5.34**
2.125	LLaMa1	7B	2.69×10^5	2.92×10^5	888	**10.86**
		13B	2.80×10^5	1.83×10^5	487	**8.54**
		30B	2.39×10^5	1.44×10^5	297	**7.47**
	LLaMa2	7B	2.24×10^5	2.78×10^5	3730	**12.80**
		13B	1.27×10^5	1.03×10^5	812	**9.18**

Table 4.1: Performance comparison of various quantization methods adapted to power-of-two (PoT) format on different LLaMa models and sizes, with a group size of 128. Our PoT method shows superior results, overcoming the limitations of applying PoT to existing uniform quantization methods.

Avg Bits	Model	Size	RTN	GPTQ	AWQ	OMNI	PoT
3.25	LLaMa1	7B	7.01	6.55	6.46	6.16	**6.12**
		13B	5.88	5.62	5.51	5.46	**5.42**
		30B	4.87	4.80	4.63	4.58	**4.50**
	LLaMa2	7B	6.66	6.29	6.24	6.21	**6.03**
		13B	5.51	5.42	5.32	5.28	**5.24**
2.25	LLaMa1	7B	1.90e3	44.01	2.6×10^5	**9.77**	9.79
		13B	781.2	15.6	2.8×10^5	**7.93**	7.96
		30B	68.04	10.92	2.4×10^5	7.13	**7.01**
	LLaMa2	7B	4.20e3	36.77	2.2×10^5	11.23	**11.03**
		13B	122.08	28.14	1.2×10^5	8.33	**8.29**

Table 4.2: WikiText-2 perplexity of 2- and 3-bit quantized LLaMa family using uniform quantization for baseline methods and PoT quantization for our method.

4.4.2 C4 and PTB Quantization Comparison

We also provide the perplexity results on the C4 and PTB datasets to further validate the effectiveness of our PoT method. Tables 4.3 and 4.4 show that our PoT method maintains a lower perplexity compared to the baselines, even under power-of-two quantization constraints.

The results shown in Tables 4.3 and 4.4 confirm the robustness of our PoT quantization method, even when adapted to different datasets like C4 and PTB. Our method achieves superior perplexity performance on these datasets, especially in the lower-bit configurations, further demonstrating its applicability across varied model sizes and data distributions.

Model	3-bit					2-bit				
	GPTQ	AWQ	OMNI	RTN	PoT	GPTQ	AWQ	OMNI	RTN	PoT
1-7B	7.85	7.92	7.76	8.62	7.79	27.71	1.90e5	13.04	1.00e3	14.82
1-13B	7.10	7.07	7.06	7.49	7.05	15.29	2.30e5	10.40	447.64	11.62
1-30B	6.47	6.37	6.38	6.58	6.35	11.93	2.40e5	9.41	99.45	11.56
2-7B	7.89	7.84	7.87	8.40	7.87	33.70	1.70e5	15.45	4.90e3	18.87
2-13B	7.00	6.94	6.99	7.18	6.98	20.97	9.40e4	11.15	139.65	12.84

Table 4.3: Perplexity comparison across LLaMa models at various bit levels on C4.

Model	OMNI		PoT	
	3-bit	2-bit	3-bit	2-bit
LLaMA1 7B	33.78	97.78	30.46	48.53
LLaMA1 13B	21.08	40.66	22.10	32.19
LLaMA1 30B	17.31	29.68	17.04	29.50
LLaMA2 7B	26.81	165.48	27.24	58.79
LLaMA2 13B	31.53	275.30	29.73	89.06

Table 4.4: PTB Perplexity results for different LLaMa models using OMNI (g128) and PoT (g64) configurations.

By utilizing group size adjustments and PoT constraints, our method avoids the need for zero-point offsets required in uniform quantization and maintains competitive results. The use of symmetric scaling without additional offsets helps streamline the quantization pipeline, ensuring both efficiency and model fidelity.

4.5 Ablation Studies

To further understand the effectiveness of each step in our two-step PoT quantization process, we perform an ablation study on various configurations. Table 4.5 shows the impact of each step on perplexity for LLaMa models of different sizes. Results indicate that both steps are crucial for achieving optimal performance. Specifically, Step 1 (Data-Agnostic Scale Initialization) alone provides a significant improvement, while the additional data-driven fine-tuning in Step 2 further reduces perplexity substantially.

Model	Step 1	Step 2	Perplexity
1-7B	✗	✗	408838.25
	✓	✗	20135.70
	✗	✓	51.87
	✓	✓	9.79
1-13B	✗	✗	40328.34
	✓	✗	8267.57
	✗	✓	103.45
	✓	✓	7.96

Table 4.5: Ablation study on perplexity for different configurations of the PoT method for LLaMA models.

The ablation study highlights that Step 1 (Section 3.3.2) alone, while effective, cannot achieve the high accuracy observed with the complete two-step process. Step 2 (Section 3.3.2) fine-tunes the scale using calibration data, enabling the model to handle nuanced patterns in the data better.

4.5.1 Efficiency of Step 2 (Section 3.3.2) : Epoch-Wise Perplexity and Loss Reduction

To further illustrate the efficiency of Step 2 (Section 3.3.2), we record the perplexity and loss values for each epoch during fine-tuning on the LLaMA 13B model. Table 4.6 demonstrates that Step 2 (Section 3.3.2) achieves consistent reductions in perplexity and loss with each epoch, reflecting the effectiveness of the data-driven fine-tuning process in optimizing the scale for PoT quantization.

The ablation study and epoch-wise analysis highlight that Step 1 (Section 3.3.2) alone, while effective, cannot achieve the high accuracy observed with the complete two-step process. Step 2 (Section 3.3.2) fine-tunes the scale using calibration data, enabling the model to handle

Epoch	3-bit Perplexity	3-bit Loss	2-bit Perplexity	2-bit Loss
1	6.8546	6.2062	1.01e6	25.7530
2	5.9315	3.8642	3209.49	20.4510
3	5.7009	3.0159	239.271	17.1299
4	5.5911	2.5444	94.2473	15.3905
5	5.5361	2.2745	49.9549	13.4980
6	5.5133	2.1291	31.4580	11.9883
7	5.4922	2.0373	23.2966	10.5491
8	5.4833	1.9688	18.2681	9.5961
9	5.4815	1.9264	15.3137	8.6508
10	5.4787	1.8962	12.8997	7.8868

Table 4.6: WikiText-2 Perplexity and Loss of LLaMA 13B model during Step 2(Section 3.3.2) fine-tuning.

nuanced patterns in the data better. This iterative learning in Step 2 (Section 3.3.2) allows for precise adjustments, progressively reducing perplexity and loss with each epoch.

4.6 Quantization Efficiency

In addition to the accuracy results, we also evaluate the time required for quantizing models of different sizes. Table 4.7 shows the quantization time for LLaMA models at 4-bit precision across various model sizes, providing insight into the efficiency of our quantization method. These quantization is on a single Titan V100 GPU.

Model	7B	13B	30B
Quantization Time (hours)	0.71	1.26	3.07

Table 4.7: LLaMA 4-bit Quantization Time

4.7 Dequantization Efficiency

To evaluate the efficiency of our PoT quantization approach, we measure warp cycles during dequantization on different GPU architectures. Table 4.8 summarizes the results, showing that our method achieves significant performance improvements over uniform FP16 dequantization.

GPU Model	Uniform FP16	PoT Quantization	Speedup
Tesla V100	110 cycles	30 cycles	3.66×
RTX 4090	98 cycles	60 cycles	1.48×

Table 4.8: Dequantization efficiency measured in warp cycles on NVIDIA Tesla V100 and RTX 4090 GPUs.

The results indicate that our PoT dequantization kernel is highly optimized for parallel execution, achieving a 3.66× speedup on the Tesla V100 and a 1.48× speedup on the RTX 4090 compared to the traditional FP16 approach. These efficiency gains demonstrate that PoT quantization is well-suited for high-performance inference tasks, making it a valuable method for large-scale LLM deployments on both current and older GPU architectures.

4.8 Conclusion of Experiments

Our PoT quantization method presents a robust, low-bit quantization solution tailored for large language models. The experiments demonstrate that our two-step approach achieves state-of-the-art perplexity across diverse benchmarks while significantly enhancing inference efficiency. The ablation studies and comparative evaluations with existing methods adapted to PoT reveal the unique challenges of PoT quantization and underscore the effectiveness of our method in addressing them. These results indicate that with targeted optimizations, PoT quantization can be a powerful, accuracy-preserving approach for efficient deployment of LLMs.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

We have presented a novel non-uniform post-training quantization technique that leverages the hardware acceleration benefits of power-of-two quantization. By employing a two-step scale factor initialization and tuning process, our method effectively overcomes the typical accuracy limitations associated with power-of-two quantization. The empirical results demonstrate that the proposed approach significantly outperforms existing methods at stringent quantization levels of 3 bits and 2 bits. Furthermore, the design enhances computational efficiency by enabling parallel search capabilities and reducing training durations even with minimal batch sizes. This efficiency ensures that the quantization process can be executed on standard hardware configurations without specialized requirements.

Notably, the proposed method achieves a 3.6× speedup in dequantization on the NVIDIA V100 and a 1.5× speedup on the RTX 4090 compared to traditional weight-only uniform quantization. These performance gains translate directly into improved real-time inference capabilities, making the approach highly practical for deployment in resource-constrained scenarios.

5.2 Limitation

Despite its advantages, our method primarily accelerates dequantization rather than the entire inference process. While power-of-two quantization simplifies computations by enabling shift and addition operations, real inference speedup remains constrained by the lack of optimized computational kernels tailored for power-of-two arithmetic. Existing deep learning frameworks and hardware accelerators are not fully optimized to exploit the efficiency potential of power-of-two quantization at the inference stage.

Addressing this limitation requires further advancements in hardware-aware kernel design, ensuring that power-of-two arithmetic operations are efficiently implemented within existing deep learning inference engines. This remains an open challenge and an important direction for future work.

5.3 Future Work

While this work primarily focuses on weight-only quantization to power-of-two values, future research could explore quantizing the scale factors themselves into power-of-two values. Such an advancement would enable the entire network to rely solely on shift and addition operations, further optimizing computational efficiency. Another promising avenue is the quantization of activation values to power-of-two formats. This extension could align low-bit tensor core computations with high-bit tensor core precision, thereby reducing data transfer overheads and potentially enabling all operations to occur within the same precision core. These advancements would further streamline the computation pipeline and maximize the resource efficiency of low-bit quantized large language models, opening new possibilities for deploying these models in real-world applications.

Additionally, I plan to produce more results on various language models and evaluate them on diverse downstream tasks, including LM Harness benchmarks, to provide deeper insights into the practical impact and performance of these quantization techniques.

Bibliography

[1] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019.

[2] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] 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.

[4] F. Cardinaux, S. Uhlich, K. Yoshiyama, J. A. García, L. Mauch, S. Tiedemann, T. Kemp, and A. Nakamura. Iteratively training look-up tables for network quantization. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):860–870, 2020.

[5] J. Chee, Y. Cai, V. Kuleshov, and C. D. Sa. QuIP: 2-bit quantization of large language models with guarantees. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[6] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[7] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022.

[8] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

[9] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li. Deepshift: Towards multiplication-less neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2359–2368, 2021.

[10] E. Frantar and D. Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.

[11] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. OPTQ: accurate quantization for generative pre-trained transformers. In *ICLR*. OpenReview.net, 2023.

[12] A. Ghaffari, M. S. Tahaei, M. Tayaranian, M. Asgharian, and V. Partovi Nia. Is integer arithmetic enough for deep learning training? *Advances in Neural Information Processing Systems*, 35:27402–27413, 2022.

[13] A. Ghaffari, S. Younesian, V. P. Nia, B. Chen, and M. Asgharian. Adpq: A zero-shot calibration free adaptive post training quantization method for llms, 2024.

[14] 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.

[15] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[16] 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.

[17] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pages 2704–2713. Computer Vision Foundation / IEEE Computer Society, 2018.

[18] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

[19] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park. Owq: Outlier-aware weight quantization for efficient fine-tuning and inference of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13355–13364, 2024.

[20] S. Li, X. Ning, K. Hong, T. Liu, L. Wang, X. Li, K. Zhong, G. Dai, H. Yang, and Y. Wang. Llm-mq: Mixed-precision quantization for efficient llm deployment. In *The Efficient Natural Language and Speech Processing Workshop with NeurIPS*, volume 9, 2023.

[21] X. Li, B. Liu, R. H. Yang, V. Courville, C. Xing, and V. P. Nia. Denseshift: Towards accurate and efficient low-bit power-of-two quantization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17010–17020, 2023.

[22] X. Li, B. Liu, Y. Yu, W. Liu, C. Xu, and V. Partovi Nia. S3: Sign-sparse-shift reparametrization for effective training of low-bit shift networks. *Advances in Neural Information Processing Systems*, 34:14555–14566, 2021.

[23] Y. Li, X. Dong, and W. Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. *arXiv preprint arXiv:1909.13144*, 2019.

[24] Y. Li, X. Dong, and W. Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *ICLR*. OpenReview.net, 2020.

[25] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. {BRECQ}: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021.

[26] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *MLSys*, 2024.

[27] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2024.

[28] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

[29] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models, 2016.

[30] D. Miyashita, E. H. Lee, and B. Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[31] D. Miyashita, E. H. Lee, and B. Murmann. Convolutional neural networks using logarithmic data representation. *ArXiv*, abs/1603.01025, 2016.

[32] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

[33] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[34] M. L. Overton. *Numerical Computing with IEEE Floating Point Arithmetic*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2004.

[35] J. Pan, C. Wang, K. Zheng, Y. Li, Z. Wang, and B. Feng. Smoothquant+: Accurate and efficient 4-bit post-training weightquantization for LLM. *CoRR*, abs/2312.03788, 2023.

[36] D. Przewlocka-Rus, S. S. Sarwar, H. E. Sumbul, Y. Li, and B. De Salvo. Power-of-two quantization for low bitwidth and hardware compliant neural networks. *arXiv preprint arXiv:2203.05025*, 2022.

[37] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[38] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[39] B. Riemann. *Ueber die Darstellbarkeit einer Function durch eine trigonometrische Reihe*, volume 13. Dieterichschen Buchhandlung, 1867.

[40] R. Saha, M. Pilanci, and A. J. Goldsmith. Minimax optimal quantization of linear models: Information-theoretic limits and efficient algorithms. *CoRR*, abs/2202.11277, 2022.

[41] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[42] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[43] 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.

[44] S. M. Stigler. *The history of statistics: The measurement of uncertainty before 1900*. Harvard University Press, 1990.

[45] X. Tang, Y. Wang, T. Cao, L. L. Zhang, Q. Chen, D. Cai, Y. Liu, and M. Yang. Lut-nn: Empower efficient neural network inference with centroid learning and table lookup. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023.

[46] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[47] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[49] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: hardware-aware automated quantization with mixed precision. In *CVPR*, pages 8612–8620. Computer Vision Foundation / IEEE, 2019.

[50] L. Wang, X. Dong, Y. Wang, L. Liu, W. An, and Y. Guo. Learnable lookup table for neural network quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12423–12433, 2022.

[51] X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, and X. Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and optimal shifting and scaling. *ArXiv*, abs/2304.09145, 2023.

[52] L. Xia, M. Anthonissen, M. Hochstenbach, and B. Koren. A simple and efficient stochastic rounding method for training neural networks in low precision. *arXiv preprint arXiv:2103.13445*, 2021.

[53] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[54] H. Yang, S. Gui, Y. Zhu, and J. Liu. Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2178–2188, 2020.

[55] H. Yao, P. Li, J. Cao, X. Liu, C. Xie, and B. Wang. Rapq: Rescuing accuracy for power-of-two low-bit post-training quantization. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)*, 2022.

[56] H. You, X. Chen, Y. Zhang, C. Li, S. Li, Z. Liu, Z. Wang, and Y. Lin. Shiftaddnet: A hardware-inspired deep network. *Advances in Neural Information Processing Systems*, 33:2771–2783, 2020.

[57] H. You, Y. Guo, Y. Fu, W. Zhou, H. Shi, X. Zhang, S. Kundu, A. Yazdanbakhsh, and Y. Lin. Shiftaddllm: Accelerating pretrained llms via post-training multiplication-less reparameterization. *CoRR*, abs/2406.05981, 2024.

[58] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.