MONASH University

# Towards Efficient Transformers via Network Quantization

Jing Liu

Doctor of Philosophy

A Thesis Submitted for the Degree of Doctor of Philosophy at Monash University in 2024

Department of Data Science and AI
Faculty of Information Technology

## Copyright notice

**Abstract**

In recent years, deep learning has been a cornerstone of major advancements in artificial intelligence (AI). The Transformer architecture, in particular, has been pivotal due to its unparalleled ability to capture complex representations. However, the widespread deployment of these models is often limited by their intensive computational and memory requirements, which are particularly problematic in resource-constrained environments like smartphones and drones. This thesis proposes a series of methodologies geared towards Green AI, focusing on the development of technologies designed to reduce computational costs while preserving performance levels.

To achieve this, this thesis explores network quantization techniques that reduce the precision of numbers used in computations, which are pivotal for democratizing AI by enhancing accessibility to advanced technologies across various platforms and communities. These techniques are categorized into two main types: quantization-aware training (QAT) and post-training quantization (PTQ). QAT involves incorporating quantization during training, which allows models to adapt to reduced precision and typically results in superior performance but higher training costs, making it well-suited for small Transformer models. Conversely, PTQ is applied with minimal post-training adjustments, presenting a cost-effective alternative with moderate performance, particularly advantageous for large-scale foundation models.

This thesis introduces novel methods for both QAT and PTQ to mitigate performance degradation after quantization. For QAT, we first introduce a Sharpness-Aware Quantization (SAQ) method that utilizes loss landscape smoothing to improve the generalization performance of quantized models. We then propose EcoFormer, an energy-efficient attention mechanism that employs kernelized hashing with an RBF kernel to transform queries and keys into compact binary codes, facilitating the approximation of attention computations with linear complexity, significantly reducing the demand for intensive computational resources. For PTQ, we present QLLM, an accurate and efficient low-bitwidth method tailored for large language models (LLMs). This method tackles the challenge of activation outliers by redistributing their magnitudes, making activations more quantization-friendly. Lastly, we propose ME-Switch, a memory-efficient expert switching framework optimized for serving multiple LLMs, which addresses the challenge of balancing model performance with storage efficiency.

**Declaration**

This thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Print Name: Jing Liu

Date: October 25, 2024

**Publications during enrolment**

(* denotes equal contribution)

First author works:

- ME-Switch: A Memory-Efficient Expert Switching Framework for Large Language Models
  **Jing Liu**, Ruihao Gong, Mingyang Zhang, Yefei He, Jianfei Cai, Bohan Zhuang
  In arXiv, 2024.

- QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models
  **Jing Liu**, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, Bohan Zhuang
  In International Conference on Learning Representations (ICLR), 2024.

- Single-path Bit Sharing for Automatic Loss-aware Model Compression
  **Jing Liu**, Bohan Zhuang, Peng Chen, Chunhua Shen, Jianfei Cai, Mingkui Tan
  In IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2023.

- EcoFormer: Energy-Saving Attention with Linear Complexity
  **Jing Liu***, Zizheng Pan*, Haoyu He, Jianfei Cai, Bohan Zhuang
  In Conference on Neural Information Processing Systems (NeurIPS), 2022. (**Spotlight, Top 5%**)

- Sharpness-aware Quantization for Deep Neural Networks
  **Jing Liu**, Jianfei Cai, Bohan Zhuang
  In arXiv, 2021.

Co-author works:

- MiniCache: KV Cache Compression in Depth Dimension for Large Language Models
  Akide Liu, **Jing Liu**, Zizheng Pan, Yefei He, Gholamreza Haffari, Bohan Zhuang
  In Conference on Neural Information Processing Systems (NeurIPS), 2024.

- ZipCache: Accurate and Efficient KV Cache Quantization with Salient Token Identification
  Yefei He, Luoming Zhang, Weijia Wu, **Jing Liu**, Hong Zhou, Bohan Zhuang
  In Conference on Neural Information Processing Systems (NeurIPS), 2024.

- EfficientDM: Efficient Quantization-Aware Fine-Tuning of Low-Bit Diffusion Models
  Yefei He, **Jing Liu**, Weijia Wu, Hong Zhou, Bohan Zhuang
  In International Conference on Learning Representations (ICLR), 2024. (**Spotlight, Top 5%**)

- Stitched ViTs are Flexible Vision Backbones
  Zizheng Pan, **Jing Liu**, Haoyu He, Jianfei Cai, Bohan Zhuang
  In European Conference on Computer Vision (ECCV), 2024.

- Efficient Stitchable Task Adaptation
  Haoyu He, Zizheng Pan, **Jing Liu**, Jianfei Cai, and Bohan Zhuang
  In Conference on Computer Vision and Pattern Recognition (CVPR), 2024.

- TFMQ-DM: Temporal Feature Maintenance Quantization for Diffusion Models
  Yushi Huang*, Ruihao Gong*, **Jing Liu**, Tianlong Chen, Xianglong Liu
  In Conference on Computer Vision and Pattern Recognition (CVPR), 2024. (**Spotlight, Top 11%**)

- Pruning self-attentions into convolutional layers in single path
  Haoyu He, Jianfei Cai, **Jing Liu**, Zizheng Pan, Jing Zhang, Dacheng Tao, Bohan Zhuang
  In IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2024.

- PTQD: Accurate Post-Training Quantization for Diffusion Models
  Yefei He, Luping Liu, **Jing Liu**, Weijia Wu, Hong Zhou, Bohan Zhuang
  In Conference on Neural Information Processing Systems (NeurIPS), 2023.

- BiViT: Extremely Compressed Binary Vision Transformers
  Yefei He, Lou Zhenyu, Luoming Zhang, **Jing Liu**, Weijia Wu, Bohan Zhuang, Hong Zhou
  In International Conference on Computer Vision (ICCV), 2023.

- A Survey on Efficient Training of Transformers
  Bohan Zhuang, **Jing Liu**, Zizheng Pan, Haoyu He, Yuetian Weng, Chunhua Shen
  In International Joint Conference on Artificial Intelligence (IJCAI), 2023.

- Dynamic Focus-aware Positional Queries for Semantic Segmentation
  Haoyu He, Jianfei Cai, Zizheng Pan, **Jing Liu**, Jing Zhang, Dacheng Tao, Bohan Zhuang
  In Conference on Computer Vision and Pattern Recognition (CVPR), 2023.

- Less is More: Pay Less Attention in Vision Transformers
  Zizheng Pan, Bohan Zhuang, Haoyu He, **Jing Liu**, Jianfei Cai
  In AAAI Conference on Artificial Intelligence (AAAI), 2022.

- Scalable visual transformers with hierarchical pooling
  Zizheng Pan, Bohan Zhuang, **Jing Liu**, Haoyu He, Jianfei Cai
  In International Conference on Computer Vision (ICCV), 2021.

- Mesa: A Memory-saving Training Framework for Transformers
  Zizheng Pan, Peng Chen, Haoyu He, **Jing Liu**, Jianfei Cai, Bohan Zhuang
  In arXiv, 2021.

**Thesis including published works declaration**

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes 2 original papers published in peer reviewed conferences and 2 submitted publications. The core theme of the thesis is **Towards Efficient Transformers via Network Quantization**. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the **ZIP Lab, DSAI** under the supervision of **Prof. Bohan Zhuang and Prof. Jianfei Cai**.

My contributions in Chapters 3 and 6 are part of work currently under submission. For Chapter 3, titled "Sharpness-aware Quantization for Deep Neural Networks", I contributed 60% of the work, including developing the core concept, conducting experiments, and drafting the manuscript. Professor Jianfei Cai contributed 15% and Professor Bohan Zhuang contributed 25%, primarily through idea discussions and manuscript refinement. For Chapter 6, titled "ME-Switch: A Memory-Efficient Expert Switching Framework for Large Language Models", I also contributed 60%, focusing on the core concept, experimentation, and initial manuscript drafting. Additional contributions were made by Ruihao Gong (5%), Mingyang Zhang (5%), and Yefei He (5%), with Professors Jianfei Cai (10%) and Bohan Zhuang (15%) further contributing through idea discussions and manuscript revisions. In the case of Chapters 4 and 5, my contribution to the work involved the following:

| Thesis Chapter | Publication Title | Status | Nature and % of student contribution | Co-author name(s) Nature and % of Co-author's contribution* | Co-author(s), Monash student Y/N* |
|---|---|---|---|---|---|
| 4 | EcoFormer: Energy-Saving Attention with Linear Complexity | Published | 55%. Concept, experimenting and writing first draft | 1) Zizheng Pan, 15%, discussing idea, experimenting, revising draft<br>2) Haoyu He, 5%, discussing idea, revising draft<br>3) Jianfei Cai, 10%, discussing idea, revising draft<br>4) Bohan Zhuang, 15%, discussing idea, revising draft | Y<br>Y<br>Supervisor<br>Supervisor |
| 5 | QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models | Published | 60%. Concept, experimenting and writing first draft | 1) Ruihao Gong, 5%, discussing idea, revising draft<br>2) Xiuying Wei, 5%, discussing idea, revising draft<br>3) Zhiwei Dong, 5%, discussing idea, revising draft<br>4) Jianfei Cai, 10%, discussing idea, revising draft<br>5) Bohan Zhuang, 15%, discussing idea, revising draft | N<br>N<br>N<br>Supervisor<br>Supervisor |

I have renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

**Student name:** Jing Liu

**Date:** 25 October, 2024

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

**Main Supervisor name:** Bohan Zhuang

**Date:** 27 October, 2024

**Acknowledgements**

First and foremost, I would like to express my deepest appreciation to my supervisors, Prof. Bohan Zhuang and Prof. Jianfei Cai, for their invaluable guidance, profound expertise, and unwavering patience. Their dedicated mentorship and kind support have greatly enriched my PhD journey, providing me with exceptional insight and guidance to navigate the challenges of the research process. Their significant and pivotal contributions have been instrumental in my development toward becoming a junior researcher. Prof. Bohan Zhuang, who has been my main supervisor since my PhD and a collaborative partner since my master's studies in 2017, has been a constant source of support, significantly enhancing my research capabilities and professional growth. His personal care and assistance, especially during the challenges of settling in Melbourne amidst the COVID-19 pandemic, have been crucial to my well-being. I am also deeply appreciative of Prof. Jianfei Cai, whose broad vision and strategic guidance have profoundly influenced my approach to research and decision-making. I am incredibly fortunate to have been supervised by such distinguished mentors.

Apart from my supervisors, I would like to extend my thanks to all my co-authors, with whom I have collaborated closely to advance our research projects. Brainstorming with them has been invaluable in helping me overcome local minima, preventing common cognitive pitfalls and broadening my perspectives. In fact, the work presented here would not have seen the light of day without the invaluable assistance provided.

I would also be delighted to say thank you to all my colleagues and friends. Although the PhD journey is arduous, their companionship has greatly enriched my life by sharing the joys and providing constant support. Beyond the rigors of academic study, I have had a great time in Melbourne – moments outside the lab have been as rewarding and memorable as those within.

I would like to thank my family for all the love—my mom, dad, and brother. It is said that life is like a box of chocolates—one never really knows what comes next. This was even more appropriately grasped with the COVID pandemic. The unconditional love helps in setting my mindset positive, giving strength to tackle all those uncertainties confidently.

Last, I acknowledge the use of generative models like ChatGPT and Grammarly to refine the language and grammar of this thesis.

# Contents

# List of Figures

# List of Tables

# Abbreviations

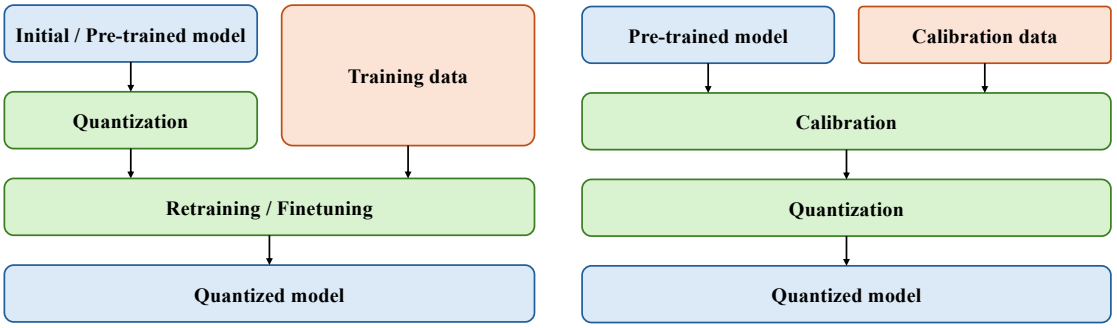
| | |
|---|---|
| **AI** | **A**rtificial **I**ntelligence |
| **ASP** | **A**udio **S**ignal **P**rocessing |
| **BERT** | **B**idirectional **E**ncoder **R**epresentations from **T**ransformer |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **CV** | **C**omputer **V**ision |
| **DL** | **D**eep **L**earning |
| **DNN** | **D**eep **N**eural **N**etwork |
| **FFN** | **F**eed-**f**orward **N**etwork |
| **FLOPS** | **F**loating **P**oint **O**perations per **S**econd |
| **FLOPs** | **F**loating **P**oint **O**peration**s** |
| **GPT** | **G**enerative **P**re-trained **T**ransformer |
| **LLM** | **L**arge **L**anguage **M**odel |
| **LN** | **L**ayer **N**ormalization |
| **MLP** | **M**ulti-**l**ayer **P**erceptron |
| **MSA** | **M**ulti-head **S**elf-**a**ttention |
| **NLP** | **N**atural **L**anguage **P**rocessing |
| **PTQ** | **P**ost-**t**raining **Q**uantization |
| **QAT** | **Q**uantization-**a**ware **T**raining |
| **RNN** | **R**ecurrent **N**eural **N**etwork |
| **SAM** | **S**harpness-**A**ware **M**inimization |
| **SAQ** | **S**harpness-**A**ware **Q**uantization |
| **STE** | **S**traight-**t**hrough **E**stimator |
| **ViT** | **V**ision **T**ransformer |

# Chapter 1

# Introduction

## 1.1 Background

Artificial intelligence (AI) is now seamlessly integrated into various aspects of our daily lives. For example, AI powers personalized recommendations on streaming services [14, 15], optimizes routes in real-time navigation apps [16, 17], and improves diagnostic accuracy in healthcare [18, 19]. At the heart of these advancements is deep learning (DL) [20], which has seen significant development over recent decades. Deep neural networks (DNNs), the backbone of deep learning, have evolved from basic multi-layer perceptrons (MLP) [21] and convolutional neural networks (CNNs) [22] to today's advanced Transformers [23]. Originally designed for natural language processing (NLP), Transformers have now become a versatile architecture broadly applied across various fields of AI, such as computer vision (CV) [24] and audio signal processing (ASP) [25]. With the availability of extensive training data and powerful computing resources such as GPUs, the scale of Transformer models has grown significantly, as exemplified by the development of generative AI models like Generative Pre-trained Transformer 4 (GPT-4) [26]. For example, GPT-3 [27], the precursor to GPT-4, already contains a remarkable 175 billion parameters, requiring a minimum of 325GB of memory for storage in half-precision (FP16) format. This immense scale, while contributing to groundbreaking capabilities, also brings significant inference costs.




(A) QAT (B) PTQ

FIGURE 1.1: Quantization-aware training (QAT) *vs.* post-training quantization (PTQ). QAT integrates quantization into the training process and can be applied to either randomly initialized models or pre-trained models, making it suitable for CNNs and small transformers due to its higher computational cost and ability to maintain accuracy. In contrast, PTQ is applied after training and requires only a small set of calibration data, making it more resource-efficient and ideal for large models, though it may come with a slight performance trade-off.

However, existing devices are limited by their computational resources and memory capacities. For instance, edge devices such as the Raspberry Pi 4 provide only 13.5 giga-floating point operations per second (GFLOPS) of computational power and 4 GB of memory[1]. Similarly, in the cloud computing domain, the NVIDIA A100 GPU offers 156 teraflops (TFLOPS) of computation and 80 GB of memory[2], illustrating the persistent constraints even within high-end GPUs. These constraints significantly challenge the deployment of large models that require extensive memory and computational resources, often resulting in slower inference times or an inability to run the models. Nonetheless, many applications such as video conferencing [28], autonomous driving [29], and chatbots [26], necessitate real-time processing on resource-constrained devices, requiring rapid data handling and decision-making capabilities. Consequently, a significant gap exists between the real-time requirements of these applications on constrained devices and the high computational costs associated with large models.

## 1.2 Network Quantization

To mitigate the gap, model compression is an effective approach, which aims to reduce the model redundancy without significant degeneration in performance. By doing so, it not only facilitates substantial financial savings, but also promotes green AI [30]

[1] https://web.eece.maine.edu/~vweaver/group/green_machines.html
[2] https://www.nvidia.com/en-us/data-center/a100/

practices by reducing operational costs and energy consumption. One key technique in model compression is network quantization [31, 32], which seeks to represent values with lower precision. By adopting lower-precision formats (*e.g.*, 8-bit integers instead of 32-bit floating-point numbers), the memory required for storing weights and activations is considerably reduced. For example, converting a model's parameters from 32-bit to 8-bit precision results in a 4× reduction in memory usage. This reduction not only decreases the overall model size, making it more memory-efficient and faster to load, but also allows for the replacement of intensive matrix multiplications with more efficient bitwise operations. Consequently, this leads to quicker inference times and reduced resource consumption, enhancing the model's suitability for real-time applications.

Existing quantization methods can be classified into two types according to the training resources: quantization-aware training (QAT) [33, 34] and post-training quantization (PTQ) [35, 36]. As illustrated in Figure 1.1, QAT incorporates quantization directly into the training process, allowing the model to adapt to lower precision throughout training. QAT can be applied to either a randomly initialized model (training from scratch) or a pre-trained model (fine-tuning), making it highly flexible. Due to the high computational cost, QAT is suitable for smaller models such as CNNs [37] and small transformers [38], where higher accuracy is prioritized. In contrast, PTQ can only be applied to pre-trained models and involves a calibration step using a small set of calibration data to adjust the model's parameters before quantization. PTQ is more resource-efficient and better suited for larger models, such as large language models (LLMs) [39], although it may come with a slight performance drop compared to QAT.

## 1.3 Research Challenges

With the reduced precision, the model's ability to capture intricate patterns and subtle relationships in the data is diminished, leading to performance degradation. Therefore, the primary challenge for both QAT and PTQ lies in mitigating the performance loss caused by quantization. In the following, we present an overview of four research challenges. The first two challenges pertain to QAT, while the last two focus on PTQ.

**RC1: Sharp loss landscape of the quantized models.** One of the major challenges in QAT arises from the inherently discrete and non-differentiable nature of network quantization. Unlike full-precision models, low-precision models represent weights and activations using a constrained set of values, which significantly limits their representation capacity. This can result in drastic loss fluctuations, as even small changes in full-precision weights due to gradient updates or quantization noise can cause significant changes in the quantized weights. These fluctuations lead to a much sharper loss landscape [40], making gradients less reliable during optimization and ultimately causing performance degradation.

**RC2: Loss of token similarity and efficiency bottlenecks in attention.** Binarization methods [41, 42] often focus on minimizing quantization errors for individual tokens but fail to preserve the pairwise similarity between tokens, which is critical for attention mechanisms. This results in degraded model performance, especially in tasks requiring detailed token relationships. Despite the efficiency gains from binarizing weights, the softmax attention mechanism remains computationally expensive due to its quadratic time complexity $\mathcal{O}\left(N^2\right)$ regarding the number of tokens $N$. This challenge becomes more pronounced with long input sequences, such as long text or high-resolution images, limiting the feasibility of deploying Transformers on resource-constrained devices.

**RC3: Activation outliers and high calibration cost in LLMs.** Activation outliers [2, 36, 43] with significantly large magnitudes in certain channels pose a major challenge for existing quantization methods for LLMs. These outliers expand the quantization range of activations, leading to imprecise quantization of typical activation values and causing substantial performance degradation. The problem is exacerbated in layer-wise or token-wise quantization, commonly employed for hardware efficiency, as the presence of outliers further distorts the quantization process, resulting in a significant drop in performance. Moreover, even with PTQ, the training cost is substantial due to the sheer volume of parameters, making the process both time-consuming and resource-intensive.

**RC4: Salient delta weights disruption.** Since no single model can excel at all tasks, deploying multiple LLMs, each fine-tuned for specific tasks, is crucial for optimal performance. To address the memory bottleneck in serving multiple LLMs, model

weights can be decomposed into pre-trained weights and delta weights introduced during fine-tuning. Quantizing these delta weights allows for efficient storage sharing of the base model across different fine-tuned models. However, variations in delta weight values across input and output channels result in information loss during quantization, degrading model performance.

## 1.4 Research Contributions

In this thesis, we present novel and effective network quantization methods for both QAT and PTQ, specifically designed to tackle the key challenges discussed earlier. The main contributions of this thesis are summarized as follows:

**Contributions for RC1**

- We propose a new method to seek flatter minima for quantized models by jointly performing quantization and loss landscape smoothing, significantly improving the generalization performance of the quantized models.
- We present a framework for analyzing loss smoothing in quantized models, viewing quantization as quantization noise and weight perturbations as adversarial adjustments. We explore three variations and provide a detailed comparison. Additionally, we propose an efficient training strategy that substantially reduces the training cost while preserving performance.

**Contributions for RC2**

- We propose a new binarization paradigm to better preserve the pairwise similarity in softmax attention. In particular, we present an energy-efficient attention with linear complexity powered by kernelized hashing to map the queries and keys into compact binary codes.
- We learn the kernelized hash functions based on the ground-truth Hamming affinity extracted from the attention scores in a self-supervised way.

**Contributions for RC3**

- We introduce a simple and effective method to redistribute activation magnitudes from outlier channels across multiple channels, making activations quantization-friendly. The overall process is gradient-free and highly efficient.
- We propose an efficient error correction mechanism that uses low-rank parameters to counteract quantization errors, reducing training time and GPU memory usage without adding any inference overhead.

**Contributions for RC4**

- We introduce a memory-efficient framework that stores a single full-precision pre-trained model and dynamically loads compressed delta weights based on user queries.
- We develop a mixed-precision quantization method that reduces storage demands for serving multiple LLMs by selectively quantizing non-salient delta weight channels while keeping salient ones intact. Additionally, we introduce a routing method that dynamically selects the appropriate LLM for a query.

## 1.5 Thesis Outline

The rest of the thesis is organized as follows: in Chapter 2, we provide a comprehensive literature review on Transformers and network quantization methods. In Chapter 3, we present a novel approach that jointly performs network quantization and loss landscape smoothing to improve the generalization performance of quantized models. In Chapter 4, we propose a new binarization technique for softmax attention that effectively preserves pairwise token similarity while significantly reducing computational complexity. In Chapter 5, we develop a method to mitigate activation outliers across different channels in LLMs and introduce an efficient error correction method to improve training efficiency. In Chapter 6, we introduce a memory-efficient framework for deploying multiple LLMs, focusing on reducing storage requirements through selective quantization of non-salient delta weights. Finally, in Chapter 7, we provide concluding remarks and discuss potential directions for future work.

# Chapter 2

# Literature Review

In this chapter, we begin with an overview of Transformers and their applications in both vision and language domains. We then delve into the development of efficient Transformer architectures aimed at overcoming the computational challenges of Transformers. Lastly, we explore the evolution of network quantization methods for improving the inference efficiency.

## 2.1 Transformers

The Transformer architecture [23] was initially introduced for machine translation tasks in NLP. A vanilla Transformer consists of an encoder and a decoder, each composed of multiple blocks, as shown in Figure 2.1. Each block within the encoder contains a multi-head self-attention (MSA) and a fully connected feed-forward network (FFN). Layer normalization (LN) [44] is applied after each MSA and FFN, with residual connections [37] linking their inputs and outputs. In each block of the decoder, apart from MSA and FFN, an additional multi-head cross-attention mechanism is introduced between the MSA and FFN to attend to the encoder's output. The input to Transformers first passes through a tokenizer, converting images or texts into a sequence of token embeddings. These embeddings, optionally combined with positional encodings [23], are then processed by the encoder and/or decoder blocks.

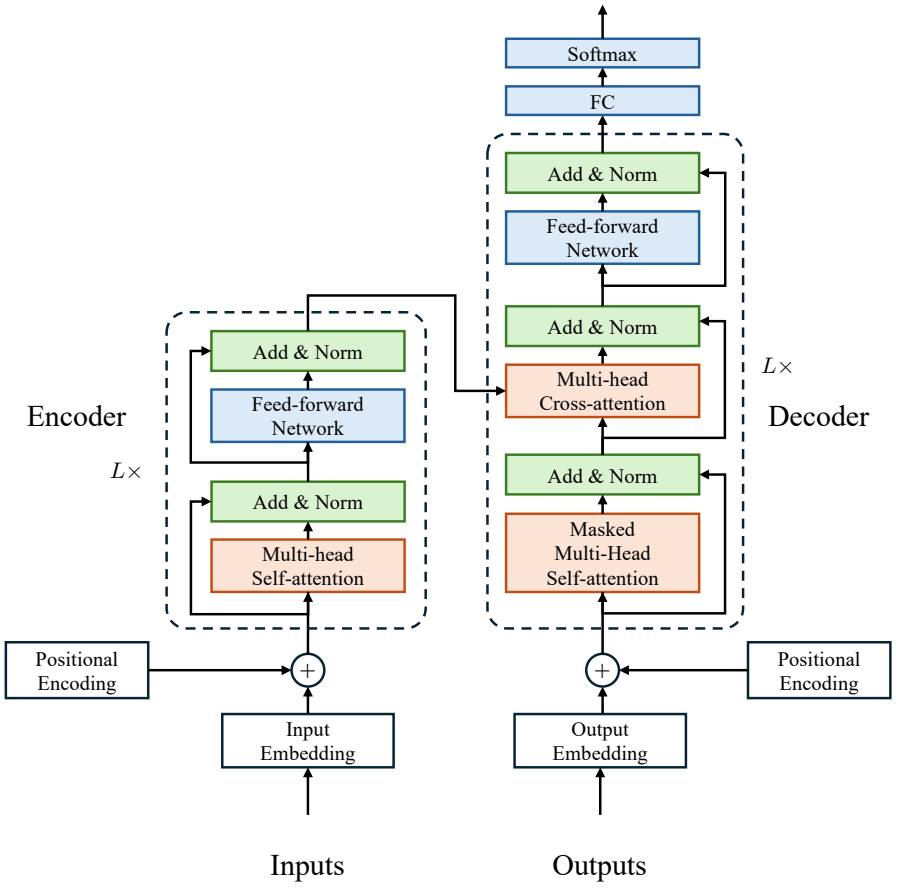



FIGURE 2.1: The Transformers model architecture.

### 2.1.1 Transformers in Vision

Vision Transformers (ViT) [45], introduced by Dosovitskiy *et al.*, demonstrated that applying a pure Transformer encoder architecture to sequences of image patches can achieve impressive performance in image classification tasks. Unlike Transformers in NLP, which process word tokens, ViT uses a patch embedding layer (*e.g.*, a linear projection) to convert an input image into a sequence of visual tokens. Additionally, ViT positions LN before each MSA and FFN. In contrast to CNNs [37], Transformers excel at capturing long-range dependencies between input tokens, making them particularly effective for modeling global context in images. However, full-length sequences often contain significant redundancy. To address this, several hierarchical vision Transformers [46–50] have been proposed, which divide Transformer blocks into multiple stages and progressively reduce the size of feature maps as the network deepens. To further improve performance, various methods have been introduced, including incorporating inductive bias through convolutional layers [51–59] and enhancing positional encodings [60–63], among others. More recently, the plain ViT has become widely adopted for large-scale

pre-training [64, 65], thanks to its simple architecture, offering general and versatile features for various downstream tasks.

### 2.1.2 Transformers in Language

Unlike recurrent neural networks (RNNs) [66], Transformers are capable of modeling long-range dependencies within input sequences while also allowing for parallel processing of those sequences. Built on the Transformer architecture, Bidirectional Encoder Representations from Transformers (BERT) [67] introduces a groundbreaking approach to pre-training language models on unlabeled text by capturing contextual information in both directions (left-to-right and right-to-left) simultaneously. The pre-trained BERT model can then be fine-tuned with an additional output layer, achieving remarkable performance on various tasks, including question answering [68, 69] and entity recognition [70, 71]. Subsequent works explore alternative architectures [72–74] and improve pre-training methods [75–77]. Additionally, research by Kaplan *et al.* [27, 78] shows that increasing the size of language models consistently enhances performance. These large models, known as large language models (LLMs) [39, 79–84], exhibit impressive capabilities, including emergent abilities [85], which are absent in smaller models. The application of large language models reached a new pinnacle with the development of ChatGPT [26], demonstrating exceptional performance across a wide range of tasks, including advanced reasoning in code [86] and mathematics [87], as well as question answering [88].

## 2.2 Efficient Transformers

The computational cost of Transformers arises from several factors. First, the memory and computational complexity of the attention mechanism scales quadratically, $\mathcal{O}\left(N^2\right)$, with the sequence length $N$, leading to extremely high computational demands when processing long sequences. Second, aside from the attention operation, a significant portion of the computation comes from the feed-forward network (FFN), which contains a large number of parameters, further contributing to the overall computational burden.

### 2.2.1 Efficient Attention

To alleviate the quadratic computational cost of vanilla attention with respect to the number of tokens, numerous efforts have focused on developing more efficient attention mechanisms. One line of research performs attention only on part of the tokens [47, 89–97]. For instance, Reformer [92] restricts the attention to the most similar token pairs via hashing and reduces the computational complexity to $\mathcal{O}(N \log N)$. Linformer [93] approximates the attention with low-rank factorization that reduces the length of the key and value. Another line of research speeds up the vanilla attention with kernel-based methods [98–105]. For example, Performer [98] approximates the softmax operation with orthogonal random features. Nyströmformer [99] and SOFT [100] approximate the full self-attention matrix via matrix decomposition. Cosformer [102] decomposes the attention into a linear form with a non-negative property and a re-weighting scheme. Although impressive achievements have been achieved, how to develop attention that is highly energy-efficient remains under-explored, as multiplications dominate the energy consumption. AdderNet variants [106–109] replace the energy-hungry multiplication-based similarity measurement with the energy-efficient addition-based L1 distance and argue that additions can also provide powerful feature representations.

### 2.2.2 Model compression

In addition to efficient attention mechanisms, model compression offers a powerful approach to reducing the size and complexity of Transformer models. A variety of methods focus on pruning redundant modules [59, 110–117] or tokens [118–121], improving inference efficiency by eliminating unnecessary components while maintaining model performance. A key advantage of pruning is that, when applied with structured patterns such as channel pruning, it can be effectively supported by existing deep learning libraries with minimal additional effort. However, when the pruning ratio exceeds certain thresholds, the performance of the model degrades significantly [122, 123]. Furthermore, some approaches employ knowledge distillation [124–129], where smaller models are trained to mimic the behavior and performance of larger models, thereby significantly enhancing their performance. However, it incurs high training costs as it requires forwarding data through the teacher model during training. Additionally, quantization

| Method | Quantization | Pruning | Knowledge Distillation |
|---|---|---|---|
| Model Size Reduction | High (*e.g.*, 4× with 8-bit quantization) | High (depends on sparsity, *e.g.*, 90%) | Medium–High (depends on student model design) |
| Inference Speed | High (on low-precision hardware) | Medium (for unstructured sparsity), High (for structured sparsity) | Medium–High (depends on student model size) |
| Accuracy Retention | High (minimal loss at 8/16 bits) | Medium (can degrade significantly under high sparsity) | Medium–High (depends on student size; may struggle if too small) |
| Training Requirement | No (for post-training quantization); Yes (for quantization-aware training) | Yes (typically requires fine-tuning to recover performance) | Yes (requires training with teacher model) |
| Hardware Dependency | Low (widely supported) | Medium (unstructured), Low (structured) | Low (hardware-agnostic) |

TABLE 2.1: A comparative analysis of quantization, pruning, and knowledge distillation.

techniques [2, 36, 130–135] have emerged as a highly effective compression strategy by reducing the numerical precision of weights and activations. This substantially lowers computational and memory costs, enabling faster inference and lower energy consumption while preserving competitive accuracy. These advantages make quantization especially attractive for deployment on resource-constrained hardware. A comparison between quantization and other compression methods is summarized in Table 2.1. Together, these techniques combined form a comprehensive strategy for optimizing Transformer models to meet the constraints of various deployment environments.

## 2.3 Network Quantization

### 2.3.1 Quantization-aware Training

QAT incorporates quantization directly into the training process, allowing the model to adapt to lower precision as it learns. Depending on the quantization bitwidth, existing methods can be broadly categorized into two main types: fixed-point quantization [31, 33, 34, 130, 132, 136–140] and binary quantization [40, 41, 133, 134, 141–145]. While binary quantization offers substantial efficiency gains in storage and computation [41, 141], it suffers from notable drawbacks, including severe accuracy degradation due to limited representational capacity [143]. To enhance the performance of quantized models, various techniques have been proposed to learn more accurate quantizers [3, 33, 136, 137, 146, 147] for better quantization mapping. Furthermore, in

order to address the optimization challenges posed by the non-differentiability of rounding operations, several approaches have been developed to approximate the gradients of the rounding function [148–150], thereby enabling more effective training to reduce quantization loss. Moreover, most previous works assign the same bitwidth for all layers [31, 33, 137, 142, 151–153]. Though attractive for simplicity, setting a uniform precision places no guarantee on optimizing network performance, since different layers have different redundancy and arithmetic intensity. Therefore, several studies proposed mixed-precision quantization [4, 154–157] that assigns different bitwidths according to the redundancy of each layer, achieving better performance and compression trade-off.

### 2.3.2 Post-training Quantization

Compared to QAT, which incurs prohibitively high training costs, PTQ offers a more resource-efficient alternative. PTQ enables models to be quantized after they have been fully trained, using only a small amount of data for calibration. To minimize the performance degradation often associated with PTQ, various methods have been proposed to perform layer-wise [35, 158–162] or block-wise calibration [135, 162–164], ensuring better alignment of quantized activations and weights with the original model. Further innovations focus on mitigating the impact of outliers, which can skew the quantization process. Strategies such as clipping [165–167] or value splitting [168] have been proposed to handle extreme values in weights and activations, allowing for more precise quantization by allocating more bits to intermediate ranges. However, in the context of large language models (LLMs), a recent study [34] has shown that MinMax quantization, which preserves the full value range of activations, outperforms clipping-based methods. This is because the outliers, while rare, are critical to the performance of LLMs, making it essential to maintain their influence in the quantization process.

### 2.3.3 Quantization on LLMs

Given constraints such as limited training data and intensive computational demands, prevailing quantization techniques for LLMs are primarily based on PTQ. Existing LLM quantization approaches can be classified into two categories: weight-only quantization [169–179] and weight-activation quantization [2, 34, 36, 43, 135, 180–185]. The former focuses on compressing the vast number of weights in LLMs to reduce the memory

footprint, which can potentially enhance training efficiency by enabling faster memory access and reducing communication overhead through the use of compressed weights, as demonstrated in QLoRA [175] and LoftQ [176]. Meanwhile, the latter compresses both weights and activations into low-bit values, aiming to accelerate computation-intensive matrix multiplication. To handle the different value ranges of weight matrices, recent studies have delved into more fine-grained quantization, such as channel-wise quantization [169] or group-wise quantization [169, 171]. To further compensate for the performance drop for extremely low-bitwidth quantization, QLoRA [175], and INT2.1 [173] introduce additional full-precision weights [182]. Recent research [43] has shown that activation outliers exist in some feature dimensions across different tokens. Several works [2, 36, 164, 180] have been proposed to migrate the quantization difficulty from activations to weights within the same channel, based on gradient-free methods [2, 36, 180] or gradient-based methods [164]. However, when handling pronounced activation outliers, existing methods often show limited improvement or result in unstable gradients.

# Chapter 3

# Sharpness-Aware Quantization for Deep Neural Networks

## 3.1 Overview

Network quantization is a prevalent paradigm in model compression. However, the abrupt changes in quantized weights during training often cause severe loss fluctuations, leading to a sharper loss landscape and unstable gradients, ultimately degrading model performance. Recently, Sharpness-Aware Minimization (SAM) has been introduced to smooth the loss landscape and improve the generalization of models. Directly applying SAM to quantized models can result in perturbation mismatches or diminishment, leading to suboptimal performance.

In this chapter, we propose a novel approach, Sharpness-Aware Quantization (SAQ), to investigate the application of SAM in model compression, specifically for quantization. We provide a unified framework, treating quantization and SAM as the introduction of quantization noise and adversarial perturbations to model weights, respectively. Based on the interaction between noise and perturbations, we formulate SAQ into three cases, each analyzed and compared comprehensively. Additionally, we introduce an efficient training strategy that incurs minimal overhead compared to standard optimizers such as SGD or AdamW. Extensive experiments on both CNNs and Transformers across diverse datasets (ImageNet, CIFAR-10/100, Oxford Flowers-102, Oxford-IIIT

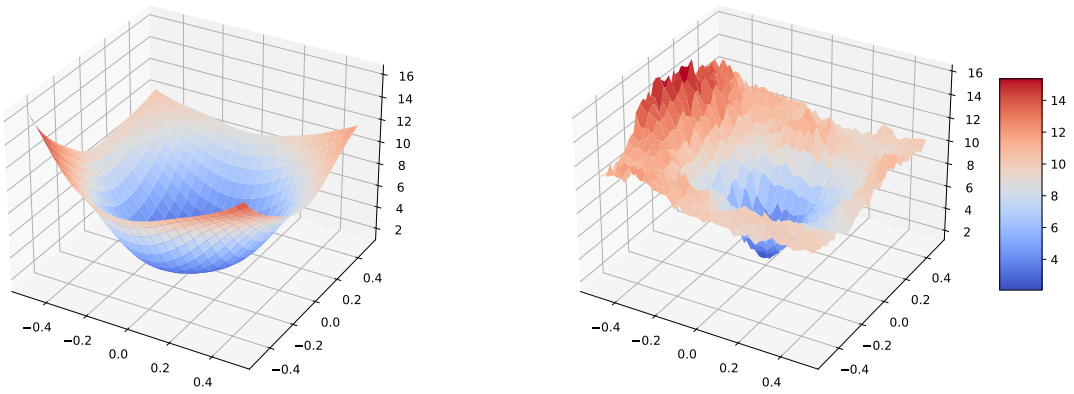
Pets) demonstrate that SAQ significantly improves the generalization performance of quantized models, achieving state-of-the-art results in uniform quantization.

## 3.2 Introduction

With powerful high-performance computing and massive labeled data, CNNs and Transformers have dramatically improved the accuracy of many CV and NLP tasks, such as image classification [37, 45], dense prediction [186, 187], sentence classification [67, 188], and machine translation [23, 189], to the level of being ready for real-world applications. Despite the remarkable breakthroughs that deep learning has achieved, the considerable computational overhead and model size greatly hampers the development and deployment of deep learning techniques at scale, especially on resource-constrained devices such as mobile phones. To obtain compact models, many network quantization methods [31, 141] have been proposed to tackle the efficiency bottlenecks.

Despite the high compression ratio, training low-precision models poses significant challenges due to the discrete and non-differentiable nature of network quantization. Unlike full-precision models, low-precision models represent weights, activations, and even gradients with only a limited set of values, which significantly constrains the representation power of the quantized models. As illustrated in Figure 3.1, even a slight change in full-precision weights, caused by either gradient updates or quantization noise, can lead to disproportionately large changes in the quantized weights due to the discretization process. This, in turn, results in drastic loss fluctuations and a much sharper loss landscape [40]. Consequently, the sharp fluctuations in the loss make the gradients highly unreliable during optimization, misguiding the weight updates and ultimately leading to a significant performance drop.

There have been some studies showing that flat minima of the loss function found by stochastic gradient-based methods tend to result in good generalization [190–193]. One prominent approach to achieving this is Sharpness-Aware Minimization (SAM) [194], along with its various extensions [195–197], which aim to smooth the loss landscape and significantly enhance model generalization. SAM works by introducing perturbations to the model weights and then minimizing a perturbed loss function, seeking parameters that reside in regions of the loss landscape characterized by uniformly low training loss.



(A) Full-precision ResNet-18 (B) 2-bit ResNet-18

FIGURE 3.1: The loss landscapes of the full-precision and 2-bit ResNet-18 models on ImageNet. We plot the loss landscapes using the visualization method in [1].

However, all the existing methods are based on full-precision over-parameterized models. *How to perform SAM on the compressed models, especially on the quantized ones, has rarely been explored, which is a new and important problem.* A simple solution is directly applying SAM to train the quantized models. Nevertheless, as we will discuss in Section 3.4.2, the introduced perturbations can be either mismatched with the quantized weights or diminished by clipping and discretization operations, which may lead to suboptimal performance.

In this chapter, we propose a novel method, called Sharpness-aware Quantization (SAQ), to find minima with both low loss value and low loss curvature and improve the performance of the quantized models. *To our knowledge, this is a pioneering work to study the effect of SAM in model compression, especially in quantization.* To this end, we first provide a unified view for quantization and SAM, where we treat them as introducing quantization noises $\boldsymbol{\epsilon}_q$ and adversarial perturbations $\hat{\boldsymbol{\epsilon}}_s$ to the model weights, respectively. According to whether $\boldsymbol{\epsilon}_q$ and $\hat{\boldsymbol{\epsilon}}_s$ are dependent on each other, we can formulate SAQ into three cases. We then study and compare these cases comprehensively. Considering that SAQ requires additional training overhead to compute $\hat{\boldsymbol{\epsilon}}_s$, we further introduce an efficient training strategy, enabling SAQ to achieve comparable training efficiency as the default optimization counterpart such as AdamW or SGD, which makes it scalable to large models. Extensive experiments on both CNNs and Transformers across various datasets show the promising performance of SAQ.

Our main contributions are summarized as follows:

- We propose SAQ to seek flatter minima for the quantized models in order to materially improve the generalization performance. To our knowledge, this is a pioneering work that jointly performs the model compression (*i.e.*, quantization) and the loss landscape smoothing.

- We provide a unified view for the landscape smoothing of the quantized models, where we consider quantization and SAM as introducing quantization noises and adversarial perturbations to the model weights, respectively. Relying on this, we present three cases of SAQ and make comprehensive comparisons among them. We further introduce an efficient training strategy to largely reduce the computational overhead brought by SAQ while keeping its performance gain.

- Experiments on both CNNs and Transformers across a variety of datasets show that SAQ improves quantized models' generalization performance and performs favorably against state-of-the-art uniform quantization methods. For example, on ImageNet, our 4-bit ViT-B/16 surpasses AdamW by 1.2% on the Top-1 accuracy. Moreover, our 4-bit ResNet-50 exceeds the state-of-the-art method by 0.9% on the Top-1 accuracy.

## 3.3 Preliminary

### 3.3.1 Network Quantization

In this chapter, we use uniform quantization which is hardware-friendly [31]. Given an $L$-layer deep model, let $w^l$ and $x^l$ be the weight and input activation w.r.t. the $l$-th layer. For simplicity, we omit the layer index $l$ afterward. Before performing quantization, we first normalize $w$ and $x$ into the scale of $[0, 1]$ by applying clipping as

$$\hat{w} = \begin{cases} \frac{1}{2}\left(\frac{w}{\alpha_w} + 1\right), & \text{if } -1 < \frac{w}{\alpha_w} < 1 \\ 0, & \text{if } \frac{w}{\alpha_w} \leq -1 \\ 1, & \text{if } \frac{w}{\alpha_w} \geq 1 \end{cases}, \tag{3.1}$$

$$\hat{x} = \begin{cases} \frac{x}{\alpha_x}, & \text{if } 0 < \frac{x}{\alpha_x} < 1 \\ 0, & \text{if } \frac{x}{\alpha_x} \leq 0 \\ 1, & \text{if } \frac{x}{\alpha_x} \geq 1 \end{cases}, \tag{3.2}$$

where $\alpha_w$ and $\alpha_x$ are layer-wise trainable clipping levels that limit the range of weight and activation, respectively. We then quantize $\hat{z} \in \{\hat{w}, \hat{x}\}$ to the discrete one $\bar{z} \in \{\bar{w}, \bar{x}\}$ by $\bar{z} = D(\hat{z}, s) = s \cdot \lfloor \hat{z}/s \rceil$, where $\lfloor \cdot \rceil$ is a rounding operator that returns the nearest integer of a given value and $s = 1/(2^b - 1)$ is the normalized step size for $b$-bit quantization. Lastly, we obtain the quantized $w$ and $x$ by

$$Q_w(w) = \alpha_w(2\bar{w} - 1), \; Q_x(x) = \alpha_x \bar{x}. \tag{3.3}$$

During training, the rounding operation $\lfloor \cdot \rceil$ is non-differentiable. To overcome this issue, following [31, 141], we apply the STE operation [198] to approximate the gradient of the rounding operator by identity mapping for backpropagation, namely, $\partial \bar{z}/\partial \hat{z} \approx 1$.

### 3.3.2 Sharpness-Aware Minimization

Without loss of generality, let $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the training data. The goal of model training is to minimize the empirical risk $\mathcal{L}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^n \ell(\mathbf{w}, \mathbf{x}_i, y_i)$, where $\ell(\mathbf{w}, \mathbf{x}_i, y_i)$ is a loss function for the sample $(\mathbf{x}_i, y_i)$ with model weights $\mathbf{w}$. Instead of seeking a single place with a local minimal loss, Sharpness-Aware Minimization [194] (SAM) seeks a region that has uniformly low training loss (both low loss and low curvature), which can be formulated as a min-max optimization problem

$$\min_{\mathbf{w}} \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}), \tag{3.4}$$

where the inner optimization problem attempts to find perturbations $\boldsymbol{\epsilon}$ in an $\ell_2$ Euclidean ball with a pre-defined radius $\rho$ that maximizes the perturbed loss $\mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon})$. To solve the inner problem, SAM approximates the optimal $\boldsymbol{\epsilon}$ to maximize $\mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon})$ using a first-order Taylor expansion as

$$\begin{aligned} \hat{\boldsymbol{\epsilon}} &= \arg\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}) \\ &\approx \arg\max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} \mathcal{L}(\mathbf{w}) + \boldsymbol{\epsilon}^\top \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \\ &\approx \rho \frac{\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})}{\|\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})\|_2}. \end{aligned} \tag{3.5}$$

By substituting Eq. (3.5) back into Eq. (3.4), we reformulate the objective and arrive at the following optimization problem:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w} + \hat{\boldsymbol{\epsilon}}). \tag{3.6}$$

Lastly, SAM updates the model weights based on the gradient $\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})|_{\mathbf{w}+\hat{\boldsymbol{\epsilon}}}$.

## 3.4 Sharpness-Aware Quantization

As shown in Figure 3.1, the low-precision model shows a much sharper loss landscape compared with the full-precision one. Therefore, small perturbations on the full-precision weights may incur large changes in the quantized weights, leading to severe loss oscillation. As a result, the gradients are unstable during training, which might mislead weight update and the resulting quantized model might converge to poor local minima. To overcome this, one may directly apply SAM to train the quantized models, which, however, *can suffer from perturbation mismatch or diminishment problems due to the clipping and discretization operations in quantization (See Section 3.4.2)*, resulting in suboptimal performance.

In the following, we describe our proposed Sharpness-Aware Quantization (SAQ) to smooth the loss landscape and improve the generalization performance of the quantized models. We begin with a unified view on the loss landscape smoothing of quantized models in Section 3.4.1, and then provide an analysis of three different cases of SAQ in Section 3.4.2. Last, we introduce a fast optimization method for SAQ in Section 3.4.3.

### 3.4.1 Unifying Quantization and SAM

We consider quantization and SAM as introducing quantization noises $\boldsymbol{\epsilon}_q$ and adversarial perturbations $\boldsymbol{\epsilon}_s$ to the model weights $\mathbf{w}$, respectively, which provides a unified view for the loss landscape smoothing of the quantized models. In this way, the optimization problem is

$$\min_{\mathbf{w},\boldsymbol{\alpha}_w,\boldsymbol{\alpha}_x} \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s) \text{ where } \hat{\boldsymbol{\epsilon}}_s = \arg\max_{\|\boldsymbol{\epsilon}_s\|_2 \leq \rho} \mathcal{L}_p, \tag{3.7}$$

where $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$ is a perturbed quantization loss and $\mathcal{L}_p$ is a perturbed loss depending on the full-precision weights $\mathbf{w}$ or the quantized weights $Q_w(\mathbf{w})$.

### 3.4.2 Case Analysis for SAQ

To solve the optimization problem in Eq. (3.7), we need to obtain $\boldsymbol{\epsilon}_q$ as well as $\hat{\boldsymbol{\epsilon}}_s$. According to whether $\boldsymbol{\epsilon}_q$ and $\hat{\boldsymbol{\epsilon}}_s$ are dependent on each other, we can transform the loss function in Eq. (3.7) to different objectives, as shown in Table 3.1. To simplify the notation, we define the quantization error function $\boldsymbol{\epsilon}_q(\mathbf{w})$ and the perturbation function $\hat{\boldsymbol{\epsilon}}_s(\mathbf{w})$ as

$$\boldsymbol{\epsilon}_q(\mathbf{w}) = Q_w(\mathbf{w}) - \mathbf{w}, \ \hat{\boldsymbol{\epsilon}}_s(\mathbf{w}) = \rho \frac{\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})}{\|\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})\|_2}. \tag{3.8}$$

TABLE 3.1: Objectives for different cases of SAQ.

| Name | Objective function |
|---|---|
| Unified | $\mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s)$ |
| Case 1 | $\mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_q(\mathbf{w}) + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w}))$ |
| Case 2 | $\mathcal{L}((\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})) + \boldsymbol{\epsilon}_q(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})))$ |
| Case 3 | $\mathcal{L}((\mathbf{w} + \boldsymbol{\epsilon}_q(\mathbf{w})) + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w} + \boldsymbol{\epsilon}_q(\mathbf{w})))$ |

**Case 1:** We calculate the quantization noises $\boldsymbol{\epsilon}_q$ and optimal perturbations $\hat{\boldsymbol{\epsilon}}_s$ independently. In this case, the perturbed loss is defined as $\mathcal{L}_p = \mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_s)$. By maximizing the perturbed loss with $\ell_2$-norm constraint, the optimal perturbations can be approximated by $\hat{\boldsymbol{\epsilon}}_s(\mathbf{w})$. In this way, the optimization problem can be transformed to

$$\min_{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x} \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_q(\mathbf{w}) + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})). \tag{3.9}$$

With Eq. (3.8), we have $\mathbf{w} + \boldsymbol{\epsilon}_q(\mathbf{w}) = Q_w(\mathbf{w})$. Consequently, the optimization problem above can now be reformulated as follows

$$\min_{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x} \mathcal{L}(Q_w(\mathbf{w}) + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})). \tag{3.10}$$

In this case, the optimal perturbations introduced to the quantized weights $Q_w(\mathbf{w})$

depend on the gradient of the full-precision weights $\mathbf{w}$. Using the chain rule, the full-precision weights' gradient can be computed by

$$\begin{aligned}\frac{\partial \mathcal{L}_p(\mathbf{w})}{\partial w_i} &= \frac{\partial \mathcal{L}_p(\mathbf{w})}{\partial Q_w(w_i)} \frac{\partial Q_w(w_i)}{\partial w_i} \\ &= \begin{cases} \frac{\partial \mathcal{L}_p(\mathbf{w})}{\partial Q_w(w_i)} & \text{if } -1 \leq \frac{w_i}{\alpha_w^l} \leq 1 \\ 0 & \text{otherwise} \end{cases},\end{aligned} \tag{3.11}$$

where $w_i$ is the $i$-th element of $\mathbf{w}$ for layer $l$ and $\alpha_w^l$ is the corresponding clipping level. Due to the clipping operation, the difference between the full-precision weights' gradient $\partial \mathcal{L}_p(\mathbf{w})/\partial w_i$ and the quantized weights' gradient $\partial \mathcal{L}_p(\mathbf{w})/\partial Q_w(w_i)$ results in a perturbation mismatch problem, which makes the training process noisy and might degrade the quantization performance.

Besides, Case 1 assumes that $\boldsymbol{\epsilon}_q$ and $\hat{\boldsymbol{\epsilon}}_s$ are computed independently, which ignores the dependency between them. To address this issue, we introduce another two cases of SAQ in the following.

**Case 2:** We first combine model weights with the optimal perturbations $\hat{\boldsymbol{\epsilon}}_s$ and then introduce the quantization noises $\boldsymbol{\epsilon}_q$ to the perturbed model weights. In this way, the optimization problem is transformed to

$$\min_{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x} \mathcal{L}((\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})) + \boldsymbol{\epsilon}_q(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w}))). \tag{3.12}$$

Same as Case 1, the perturbed loss is $\mathcal{L}_p = \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_s)$ and the optimal perturbations can be obtained by $\hat{\boldsymbol{\epsilon}}_s(\mathbf{w})$. In this case, the quantization noises $\boldsymbol{\epsilon}_q(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w}))$ is represented as a function of the optimal perturbations $\hat{\boldsymbol{\epsilon}}_s(\mathbf{w})$. Using Eq. (3.8), we reformulate the problem as

$$\min_{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x} \mathcal{L}(Q_w(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w}))). \tag{3.13}$$

Nevertheless, the introduced small perturbations may not change the resulting quantized weights due to the discretization process and results in perturbation diminishment issue, *i.e.*, $Q_w(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})) = Q_w(\mathbf{w})$. As a result, $\mathcal{L}(Q_w(\mathbf{w} + \hat{\boldsymbol{\epsilon}}_s(\mathbf{w})))$ might be reduced to $\mathcal{L}(Q_w(\mathbf{w}))$, which degenerates to the regular quantization.

**Case 3:** We first combine model weights with the quantization noises $\boldsymbol{\epsilon}_q$ and then introduce the optimal perturbations $\hat{\boldsymbol{\epsilon}}_s$. In this way, the optimization problem becomes

$$\min_{\mathbf{w},\boldsymbol{\alpha}_w,\boldsymbol{\alpha}_x} \mathcal{L}((\mathbf{w}+\boldsymbol{\epsilon}_q(\mathbf{w}))+\hat{\boldsymbol{\epsilon}}_s(\mathbf{w}+\boldsymbol{\epsilon}_q(\mathbf{w}))). \tag{3.14}$$

In this case, we define the perturbed loss as $\mathcal{L}_p = \mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q(\mathbf{w})+\boldsymbol{\epsilon}_s)$ and obtain the optimal perturbations by $\hat{\boldsymbol{\epsilon}}_s(\mathbf{w}+\boldsymbol{\epsilon}_q(\mathbf{w}))$ which is expressed as a function of the quantization noises $\boldsymbol{\epsilon}_q(\mathbf{w})$. With Eq. (3.8), the optimization problem can be rewritten as

$$\min_{\mathbf{w},\boldsymbol{\alpha}_w,\boldsymbol{\alpha}_x} \mathcal{L}(Q_w(\mathbf{w})+\hat{\boldsymbol{\epsilon}}_s(Q_w(\mathbf{w}))), \tag{3.15}$$

where we introduce perturbations to the quantized weights $Q_w(\mathbf{w})$ rather than the full-precision weights $\mathbf{w}$ as in Case 2. In this way, the introduced perturbations will not be diminished by the quantization operation. Moreover, compared with Case 1, Case 3 does not suffer from the perturbation mismatch issue since the optimal perturbations depend on the gradient of the quantized weights instead of the full-precision ones. In summary, Case 3 is the best suited to smooth the loss landscape of the quantized models.

### 3.4.3 Fast Optimization for SAQ

**Final objective.** In SAQ, we seek model parameters $\mathbf{u} \in \{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x\}$ that is located in a neighborhood with uniformly low value of $\mathcal{L}(Q_w(\mathbf{w}))$ by minimizing a surrogate loss $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$. However, the sharp loss landscape of the quantized model leads to a large angle $\theta$ between the gradient of the perturbed quantization loss $\mathbf{g}_s = \nabla_{\mathbf{u}}\mathcal{L}\left(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s\right)$ and the gradient of vanilla quantization loss $\mathbf{g} = \nabla_{\mathbf{u}}\mathcal{L}(Q_w(\mathbf{w}))$ (See Figure 3.2), which forms a gap between $\mathcal{L}(Q_w(\mathbf{w}))$ and $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$, particularly at extreme low-bit cases. Consequently, the optimization becomes challenging and leads to sub-optimal performance.

To overcome this challenge, we introduce an additional vanilla quantization loss $\mathcal{L}(Q_w(\mathbf{w}))$ into the objective and reformulate the optimization problem as

$$\begin{aligned} &\min_{\mathbf{w},\boldsymbol{\alpha}_w,\boldsymbol{\alpha}_x} \mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)+\mathcal{L}(Q_w(\mathbf{w})) \\ &\text{where } \hat{\boldsymbol{\epsilon}}_s = \arg\max_{\|\boldsymbol{\epsilon}\|_2\leq\rho}\mathcal{L}_p, \end{aligned} \tag{3.16}$$

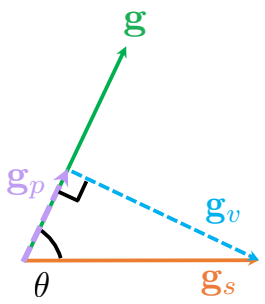



FIGURE 3.2: An illustration of gradient decomposition in SAQ where we decompose $\mathbf{g}_s = \nabla_{\mathbf{u}} \mathcal{L}\left(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s\right)$ into components $\mathbf{g}_p$ and $\mathbf{g}_v$ that are parallel and vertical to $\mathbf{g} = \nabla_{\mathbf{u}} \mathcal{L}(Q_w(\mathbf{w}))$. $\theta$ is the angle between $\mathbf{g}_s$ and $\mathbf{g}$.

where we enforce the quantized models to find minima with both low $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$ and $\mathcal{L}(Q_w(\mathbf{w}))$. As the gradient of $\mathcal{L}(Q_w(\mathbf{w}))$ has been computed during the backpropagation when solving the inner problem as shown in Eq. (3.5), we can reuse them when solving the outer problem.

**Efficient training.** Note that solving the problem in Eq. (3.16) requires additional forward and backward propagation, which cumulatively results in roughly doubled training cost compared with regular optimizers such as SGD or Adam. To address this issue, we customize an efficient training strategy following LookSAM [199]. As shown in Figure 3.2, we decompose the perturbed quantization loss gradient $\mathbf{g}_s = \nabla_{\mathbf{u}} \mathcal{L}\left(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s\right)$ into two components, $\mathbf{g}_p$ and $\mathbf{g}_v$, which are parallel and vertical to the gradient of the vanilla quantization loss, $\mathbf{g} = \nabla_{\mathbf{u}} \mathcal{L}(Q_w(\mathbf{w}))$, respectively. We have a similar observation as in LookSAM that $\mathbf{g}_v$ changes much slower than $\mathbf{g}_p$ and $\mathbf{g}_s$ during training. Therefore, we only calculate $\mathbf{g}_s$ every $\tau$ iterations and obtain its vertical component $\mathbf{g}_v$. For the intermediate iterations, we reuse the direction of $\mathbf{g}_v$ to approximate $\mathbf{g}_s = \mathbf{g} + \beta \|\mathbf{g}\| \frac{\mathbf{g}_v}{\|\mathbf{g}_v\|}$, where $\beta$ is a hyperparameter scaling the gradient. As a result, SAQ is only slightly slower than the regular optimizers (See Table 3.7). We then update the model parameters using a gradient combination of $\mathbf{g}_s + \mathbf{g}$. The training process of SAQ is summarized in Algorithm 1.

## 3.5 Experiments

**Datasets and evaluation metrics.** We evaluate our method on ImageNet [200] which is a large-scale dataset containing 1.28 million training images and 50k validation samples

**Algorithm 1:** Training algorithm for SAQ

**Input:** Training set $\mathcal{D}$, model parameter $\mathbf{u} \in \{\mathbf{w}, \boldsymbol{\alpha}_w, \boldsymbol{\alpha}_x\}$, learning rate $\eta$, update frequency $\tau$, total training iterations $T$, hyperparameter $\beta$.

**for** $t \in \{1, 2, \ldots, T\}$ **do**
  Sample a batch of data $\mathcal{B}_t$ from $\mathcal{D}$
  Compute $\mathbf{g} = \nabla_{\mathbf{u}} \mathcal{L}(Q_w(\mathbf{w}))$ on $\mathcal{B}_t$
  **if** $t = \tau k, \quad k \in \mathbb{N}^+$ **then**
    Compute $\boldsymbol{\epsilon}_q$ and $\hat{\boldsymbol{\epsilon}}_s$
    Compute $\mathbf{g}_s = \nabla_{\mathbf{u}} \mathcal{L}\left(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s\right)$ on $\mathcal{B}_t$
    Project $\mathbf{g}_s$ onto $\mathbf{g}$ to obtain $\mathbf{g}_p = \frac{\mathbf{g}_s{}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{g}} \mathbf{g}$
    Compute $\mathbf{g}_v = \mathbf{g}_s - \mathbf{g}_p$
  **else**
    Compute $\mathbf{g}_s = \mathbf{g} + \beta \|\mathbf{g}\| \frac{\mathbf{g}_v}{\|\mathbf{g}_v\|}$
  Update $\mathbf{u}$ by $\mathbf{u} = \mathbf{u} - \eta(\mathbf{g}_s + \mathbf{g})$

with 1k classes. We measure the performance of different methods using the Top-1 and Top-5 accuracy.

**Implementation details.** Our implementations are based on PyTorch [201]. We apply SAQ to CNNs and vision Transformers, including ResNet-18 [37], ResNet-34, ResNet-50, MobileNetV2 [202] and ViT [45]. We first train the full-precision models and use them to initialize the low-precision ones. Following LSQ [33], we quantize both weights and activations for all matrix multiplication layers, including convolutional layers, fully-connected layers, and self-attention layers. For the first and last layers, we quantize both weights and activations to 8-bit to preserve the performance. We do not apply quantization to the input images, as they are already stored in an 8-bit format.

For CNNs, we use the uniform quantization method mentioned in Section 3.3.1. Relying on SGD with the momentum term of 0.9, we apply SAQ with Case 3 to train the quantized models with a mini-batch size of 512 unless otherwise specified. Following APOT [151], we use weight normalization before quantization. We initialize the clipping levels to 1. We fine-tune 90 epochs for ResNet-18, ResNet-34, and ResNet-50. We set weight decay to $1 \times 10^{-4}$ by default except for 2-bit ResNet-18, for which we set it to $2.5 \times 10^{-5}$ following [33]. For MobileNetV2, we fine-tune 140 epochs following [5]. We set the weight decay to $4 \times 10^{-5}$. The learning rate is initialized to 0.02 and decreased to 0 following the cosine annealing [203]. For ViTs, we use LSQ+ [147] uniform quantization following Q-ViT [204]. We initialize the clipping levels by minimizing the quantization error following [205]. Relying on AdamW [206], we apply SAQ with Case 3 to train

ViTs. The learning rate is initialized to $2 \times 10^{-4}$ and decreased to 0 using the cosine annealing. We train the quantized model for 150 epochs with a mini-batch size of 1,024.

We do not apply the automatic mixed-precision training following [204]. As suggested by SAM [194] and GSAM [196], we apply $m$-sharpness strategy with $m = 128$. For both CNNs and ViTs, we use inception-style pre-processing [207] without strong data augmentation. Specifically, we randomly crop $224 \times 224$ patches from an image or its horizontal flip counterpart for training. At test time, a $224 \times 224$ centered crop is chosen. For the hyper-parameter $\rho$, we conduct grid search over {0.02, 0.05, 0.1, 0.15, 0.2, ..., 1.0} to find appropriate values following the common practice in SAM [194] and GSAM [196]. We put the detailed settings of $\rho$ in Table 3.2. Following LookSAM [199], we set hyperparameter $\beta$ and update frequency $\tau$ to 0.7 and 4, respectively. For MobileNetV2, we fine-tune the quantized model with additional learnable layer-wise offsets for activations and knowledge distillation following [5, 208]. As suggested by [5, 209], we apply BN re-estimation for 10 iterations after training. To compute the largest eigenvalue $\lambda_{max}$ of the Hessian of different quantized models on ImageNet, we use the power iteration algorithm following [210]. To reduce the computational cost, we randomly sample 10k training images for computation.

TABLE 3.2: Hyper-parameter $\rho$ for different quantized models on ImageNet.

| Network | ResNet-18 | | ResNet-34 | | ResNet-50 | | MobileNetV2 | ViT-S/32 | ViT-S/16 | ViT-B/32 | ViT-B/16 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bitwidth | 2 | 4 | 2 | 4 | 2 | 4 | 4 | 4 | 4 | 4 | 4 |
| $\rho$ | 0.30 | 0.65 | 0.65 | 0.90 | 0.65 | 0.95 | 0.40 | 0.01 | 0.01 | 0.01 | 0.01 |

For the transfer learning experiments, we train all models for 100 epochs. We use SGD with a momentum term of 0.9 for optimization. The learning rate is initialized to 0.01 and decreased to 0 using the cosine annealing. The mini-batch size and the weight decay are set to 64 and 0, respectively.

**Compared methods.** We conduct a comprehensive comparison with a wide array of fixed-point quantization methods, including well-known techniques such as DoReFa-Net [31], PACT [3], LQ-Nets [136], DSQ [211], FAQ [166], QIL [137], Auxi [212], and PROFIT [5]. In addition, we compare with more recent advancements like LSQ [33], APOT [151], LSQ+[147], LLSQ[213], DAQ [214], BRECQ [162], EWGS [215], BR [139], OOQ [209], and LLT [216].

TABLE 3.3: Performance comparisons of different methods with ResNet-18, ResNet-34 and ResNet-50 on ImageNet. We obtain DoReFa-Net results from [3]. "W/A" refers to the bitwidth of weights and activations, respectively. "FP" represents the Top-1 accuracy of the full-precision models. "-" denotes that the results are not reported. We do not apply advanced techniques to boost the performance as mentioned in the compared methods in Section 3.5.

| Method | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 |
|---|---|---|---|---|---|---|
| ResNet-18 (FP: 70.7) | | | | | | |
| DoReFa-Net* [31] | 4/4 | 68.1 | 88.1 | 2/2 | 62.6 | 84.4 |
| PACT* [3] | 4/4 | 69.2 | 89.0 | 2/2 | 64.4 | 85.6 |
| LQ-Nets* [136] | 4/4 | 69.3 | 88.8 | 2/2 | 64.9 | 85.9 |
| DSQ [211] | 4/4 | 69.6 | - | 2/2 | 65.2 | - |
| BRECQ [162] | 4/4 | 69.6 | - | 2/2 | - | - |
| FAQ [166] | 4/4 | 69.8 | 89.1 | 2/2 | - | - |
| QIL* [137] | 4/4 | 70.1 | - | 2/2 | 65.7 | - |
| LLT* [216] | 4/4 | 70.4 | 89.6 | 2/2 | 66.0 | 86.2 |
| Auxi [212] | 4/4 | - | - | 2/2 | 66.7 | 87.0 |
| DAQ* [214] | 4/4 | 70.5 | - | 2/2 | 66.9 | - |
| EWGS* [215] | 4/4 | 70.6 | - | 2/2 | 67.0 | - |
| BR [139] | 4/4 | 70.8 | 89.6 | 2/2 | 67.2 | 87.3 |
| APOT [151] | 4/4 | 70.7 | 89.6 | 2/2 | 67.3 | 87.5 |
| LSQ [33] | 4/4 | 71.1 | **90.0** | 2/2 | 67.6 | **87.6** |
| SAQ (Ours) | 4/4 | **71.6** | **90.0** | 2/2 | **67.8** | **87.6** |
| ResNet-34 (FP: 74.1) | | | | | | |
| LQ-Nets* [136] | 4/4 | - | - | 2/2 | 69.8 | 89.1 |
| DSQ [211] | 4/4 | 72.8 | - | 2/2 | 70.0 | - |
| FAQ [166] | 4/4 | 73.3 | 91.3 | 2/2 | - | - |
| QIL* [137] | 4/4 | 73.7 | - | 2/2 | 70.6 | - |
| APOT [151] | 4/4 | 73.8 | 91.6 | 2/2 | 70.9 | 89.7 |
| DAQ* [214] | 4/4 | 73.7 | - | 2/2 | 71.0 | - |
| Auxi [212] | 4/4 | - | - | 2/2 | 71.2 | 89.8 |
| EWGS* [215] | 4/4 | 73.9 | - | 2/2 | 71.4 | - |
| LSQ [33] | 4/4 | 74.1 | 91.7 | 2/2 | 71.6 | 90.3 |
| SAQ (Ours) | 4/4 | **75.0** | **92.3** | 2/2 | **71.8** | **90.4** |
| ResNet-50 (FP: 76.8) | | | | | | |
| DoReFa-Net* [31] | 4/4 | 71.4 | 89.8 | 2/2 | 67.1 | 87.3 |
| LQ-Net* [136] | 4/4 | 75.1 | 92.4 | 2/2 | 71.5 | 90.3 |
| FAQ [166] | 4/4 | 76.3 | 93.0 | 2/2 | - | - |
| PACT* [3] | 4/4 | 76.5 | 93.2 | 2/2 | 72.2 | 90.5 |
| APOT [151] | 4/4 | 76.6 | 93.1 | 2/2 | 73.4 | 91.4 |
| LSQ [33] | 4/4 | 76.7 | 93.2 | 2/2 | 73.7 | 91.5 |
| Auxi [212] | 4/4 | - | - | 2/2 | 73.8 | 91.4 |
| SAQ (Ours) | 4/4 | **77.6** | **93.6** | 2/2 | **74.5** | **91.9** |

* denotes that the first and last layers are not quantized.

Unless otherwise specified, we do not apply advanced techniques such as knowledge distillation [5, 208, 212, 217], non-uniform quantization [151, 208, 218], asymmetric quantization [5, 147, 208], pre-activation [33, 208, 218], weight regularization [139, 208, 209], gradient scaling [215], progressive quantization [5, 217], batch normalization re-estimation [5, 209] and iterative training with weight freezing [5, 209], which can further improve the performance of the quantized models.

### 3.5.1 Main Results

We apply SAQ to quantize ResNet-18, ResNet-34, and ResNet-50 on ImageNet. From Table 3.3, SAQ outperforms existing state-of-the-art uniform quantization methods by a large margin. The improvement is more obvious with the increase of bitwidth. For example, for 2-bit ResNet-34, the Top-1 accuracy improvement of SAQ over LSQ is 0.2% while for the 4-bit one is 0.9%. We speculate that the loss landscape of the quantized models becomes sharper with the decrease of bitwidths due to the discretization operation in quantization as shown in Figures 3.6 and 3.7. As a result, smoothing the loss landscapes of the 2-bit quantized models is harder than the 4-bit counterparts. Moreover, for 2-bit quantization, deeper models show more obvious accuracy improvement over the state-of-the-art methods. For instance, SAQ surpasses Auxi by 0.7% on 2-bit ResNet-50 while only bringing 0.2% Top-1 accuracy improvement over LSQ on 2-bit ResNet-18. Remarkably, our 4-bit ResNet-34 surpasses the full-precision model by 0.9% on the Top-1 accuracy. One possible reason is that performing quantization with SAQ helps to remove redundancy and regularize the networks. Similar phenomena are also observed in LSQ.

To show the effectiveness of our method on lightweight models, we further apply SAQ to quantize MobileNetV2. From Table 3.4, SAQ yields better performance than the state-of-the-art uniform quantization methods. For example, SAQ exceeds PROFIT by 0.4% on the Top-1 accuracy. We also apply SAQ to ViT [45]. We implement LSQ+ following [204] and compare our method with it. From Table 3.4, our SAQ shows consistently superior performance over the baseline LSQ+ (*e.g.*, 1.2% Top-1 accuracy improvement on ViT-B/16).

TABLE 3.4: Performance comparisons in terms of ViT-S/32, ViT-S/16, ViT-B/32, ViT-B/16, and MobileNetV2 on ImageNet. We obtain the results of PACT from [4]. We do not apply iterative training with weight freezing and progressive quantization in PROFIT [5] to improve the performance of the quantized models.

| Network | Method | Bitwidth (W/A) | Top-1 Acc. (%) | Top-5 Acc. (%) |
|---|---|---|---|---|
| ViT-S/32 (FP: 68.5) | LSQ+ [147] | 4/4 | 68.0 | 88.1 |
| | SAQ (Ours) | 4/4 | **68.6** | **88.4** |
| ViT-S/16 (FP: 75.9) | LSQ+ [147] | 4/4 | 76.1 | 93.0 |
| | SAQ (Ours) | 4/4 | **76.9** | **93.5** |
| ViT-B/32 (FP: 70.7) | LSQ+ [147] | 4/4 | 72.1 | 90.4 |
| | SAQ (Ours) | 4/4 | **72.7** | **90.7** |
| ViT-B/16 (FP: 77.2) | LSQ+ [147] | 4/4 | 78.0 | 93.4 |
| | SAQ (Ours) | 4/4 | **79.2** | **94.2** |
| MobileNetV2 (FP: 71.9) | PACT [3] | 4/4 | 61.4 | 83.7 |
| | DSQ* [211] | 4/4 | 64.8 | - |
| | BRECQ [162] | 4/4 | 66.6 | - |
| | LLSQ* [213] | 4/4 | 67.4 | 88.0 |
| | EWGS [215] | 4/4 | 70.3 | - |
| | BR [139] | 4/4 | 70.4 | 89.4 |
| | OOQ [209] | 4/4 | 70.6 | - |
| | PROFIT [5] | 4/4 | 71.6 | **90.4** |
| | SAQ (Ours) | 4/4 | **72.0** | **90.4** |

* denotes that the first and last layers are not quantized.

### 3.5.2 Ablation Studies

**Performance comparisons of different cases.** To investigate the effectiveness of different cases introduced in Section 3.4.2, we apply different methods to quantize ResNet-50 on ImageNet. We use "SGD" to represent training the quantized models with the vanilla SGD. To measure the loss curvature, we report the largest eigenvalue $\lambda_{\max}$ of the Hessian of the converged quantized models following [194, 219]. From Table 3.5, Case 1, Case 2, and Case 3 all yield significantly higher accuracy and lower $\lambda_{\max}$ than the SGD counterpart. This strongly shows that our method is able to smooth the loss landscape and improve the generalization performance of the quantized models. Among the three cases, Case 2 performs the worst with the lowest accuracy and the highest $\lambda_{\max}$, which suggests that the perturbations introduced by SAM might be diminished due to the discretization, leading to suboptimal performance. Moreover, Case 3 consistently performs better than Case 1. For example, on 4-bit ResNet-50, Case 3 exceeds Case 1 by 0.3%

TABLE 3.5: Performance comparisons of different cases. We report the results of ResNet-50 on ImageNet. $\lambda_{max}$ denotes the largest eigenvalue of the Hessian of the converged quantized model. Lower $\lambda_{\max}$ indicates flatter loss landscapes.

| Method | Bitwidth (W/A) | Acc. (%) Top-1 | Acc. (%) Top-5 | $\lambda_{\max}$ | Bitwidth (W/A) | Acc. (%) Top-1 | Acc. (%) Top-5 | $\lambda_{\max}$ |
|---|---|---|---|---|---|---|---|---|
| ResNet-50 | | | | | | | | |
| SGD | 4/4 | 76.5 | 93.1 | 71.8 | 2/2 | 73.9 | 91.6 | 60.1 |
| Case 1 | 4/4 | 77.3 | 93.5 | 6.6 | 2/2 | 74.3 | 91.8 | 12.6 |
| Case 2 | 4/4 | 77.0 | 93.3 | 14.0 | 2/2 | 74.2 | 91.8 | 24.4 |
| Case 3 | 4/4 | **77.6** | **93.6** | **6.3** | 2/2 | **74.5** | **91.9** | **9.5** |

on the Top-1 accuracy as well as achieving lower $\lambda_{\max}$. These results indicate that the perturbation mismatch issue in Case 1 might degrade the quantization performance.

Besides, to investigate the generalization capability of SGD and our SAQ, we show the training and validation losses of 4-bit ViT-B/16 on ImageNet in Figure 3.3. Compared with SGD, SAQ achieves lower training and validation losses at the beginning of training. At the end of training, SAQ shows slightly higher training loss but much lower validation loss and $\lambda_{\max}$ than SGD. These results justify that SAQ converges to a flatter local minimum and thus achieves better generalization performance.

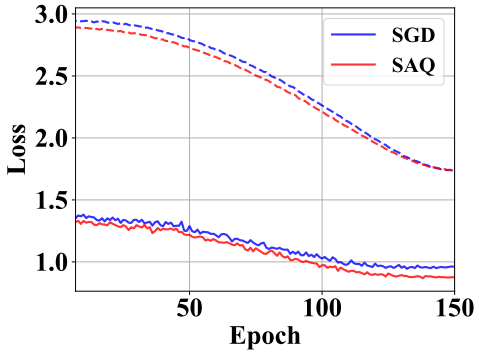



FIGURE 3.3: The training (dashed line) and validation (solid line) losses for 4-bit ViT-B/16 on ImageNet. The $\lambda_{max}$ of SGD and SAQ obtained quantized models are 14,564 and 676, respectively.

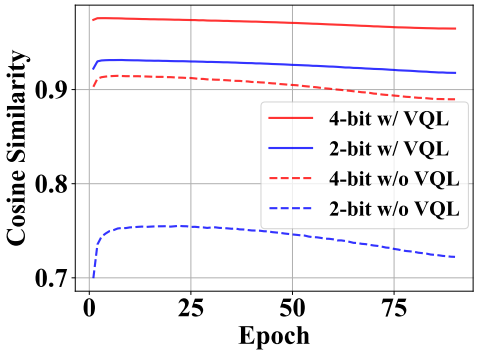



FIGURE 3.4: Effect of introducing the vanilla quantization loss (VQL). We visualize the cosine similarity between $\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon}_q + \hat{\boldsymbol{\epsilon}}_s)$ and $\nabla_{\mathbf{u}}\mathcal{L}(Q_w(\mathbf{w}))$ in terms of 2-bit and 4-bit ResNet-18 on ImageNet.

**Effect of different losses in the objective function.** To investigate the effect of different components in the objective in Eq. (3.16), we apply different methods to quantize ResNet-18. From Table 3.6, using the perturbed quantization loss $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$ surpasses the one equipped with the vanilla quantization loss $\mathcal{L}(Q_w(\mathbf{w}))$ by 0.2% on the Top-1 accuracy for 2-bit and 4-bit quantization, supporting that smoothing the

loss landscape improves the generalization performance of the quantized models. By combining $\mathcal{L}(Q_w(\mathbf{w}))$ and $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$, we observe Top-1 accuracy improvement of 0.3% for both 2-bit and 4-bit quantization.

TABLE 3.6: Effect of different losses in the objective function in Eq. (3.16) on ImageNet. "VQL" represents the vanilla quantization loss $\mathcal{L}(Q_w(\mathbf{w}))$ and "PQL" denotes the perturbed quantization loss $\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$.

| VQL | PQL | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 |
|---|---|---|---|---|---|---|---|
| | | ResNet-18 | | | | | |
| ✓ | | 2/2 | 67.3 | 87.4 | 4/4 | 71.1 | 89.8 |
| | ✓ | 2/2 | 67.5 | 87.5 | 4/4 | 71.3 | 90.0 |
| ✓ | ✓ | 2/2 | **67.8** | **87.6** | 4/4 | **71.6** | **90.1** |

To further show the effect of $\mathcal{L}(Q_w(\mathbf{w}))$, we visualize the cosine similarity between $\nabla_{\mathbf{u}}\mathcal{L}(\mathbf{w}+\boldsymbol{\epsilon}_q+\hat{\boldsymbol{\epsilon}}_s)$ and $\nabla_{\mathbf{u}}\mathcal{L}(Q_w(\mathbf{w}))$ of the quantized ResNet-18 in Figure 3.4. We observe that the cosine similarities of 2-bit models are lower than those of 4-bit ones during training. By introducing $\mathcal{L}(Q_w(\mathbf{w}))$, all quantized models achieve higher similarities. These results justify that introducing $\mathcal{L}(Q_w(\mathbf{w}))$ helps to reduce the gap between two losses and improve the performance.

**Effect of the efficient training strategy.** To investigate the effect of the efficient training strategy mentioned in Section 3.4.3, we apply SAQ to train 4-bit ResNet-34 with and without the efficient training strategy on ImageNet. The training throughput is quantified by the number of processed images per second on 4 NVIDIA V100 GPUs with a mini-batch size of 512. From Table 3.7, we observe that our SAQ with efficient training strategy is only ∼11% slower than SGD while achieving nearly the same performance as SAQ without the efficient training strategy.

TABLE 3.7: Effect of the efficient training (ET) strategy on ImageNet. We measure the training throughput on 4 NVIDIA V100 GPUs with a mini-batch size of 512.

| Network | Method | Accuracy (%) Top-1 | Accuracy (%) Top-5 | Train Throughput (images/s) |
|---|---|---|---|---|
| 4-bit ResNet-34 | SGD | 74.4 | 91.9 | **1632** |
| | SAQ w/o ET | **75.1** | 92.2 | 841 |
| | SAQ w/ ET | 75.0 | **92.3** | 1454 |

**Sensitivity of Hyperparameter $\rho$.** To investigate the sensitivity of hyperparameter $\rho$, we apply SAQ to train 4-bit ResNet-18 with different $\rho$ and show the results in Figure 3.5. From the results, SAQ is relatively insensitive to $\rho$ as it outperforms SGD for a wide range

of values. With the increase of $\rho$, the performance of the quantized models improves initially but then deteriorates. On the one hand, increasing perturbation strength helps to improve the generalization performance of the quantized model. On the other hand, too large $\rho$ may incur optimization difficulty and thus lead to sub-optimal performance.

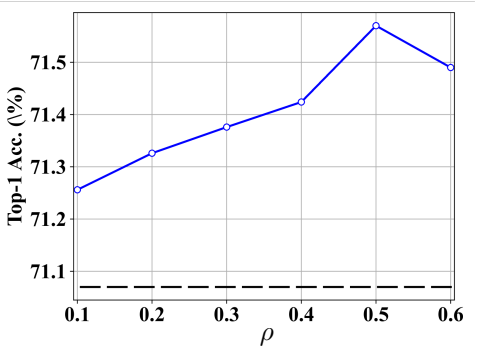



FIGURE 3.5: Performance comparisons with different hyperparameter $\rho$. We apply SAQ to obtain 4-bit ResNet-18 on ImageNet. The black dashed line denotes the results of the quantized model trained by SGD.

TABLE 3.8: Effect of jointly performing quantization and loss landscape smoothing on ImageNet. The Top-1 accuracy of the full-precision ResNet-18 obtained by SAM is 70.9%.

| Method | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 | Bitwidth (W/A) | Accuracy (%) Top-1 | Accuracy (%) Top-5 |
|---|---|---|---|---|---|---|
| ResNet-18 (FP: 70.7) | | | | | | |
| SGD | 2/2 | 67.3 | 87.4 | 4/4 | 71.1 | 89.8 |
| SAM → SGD | 2/2 | 67.4 | 87.5 | 4/4 | 71.1 | 89.9 |
| SAQ (Ours) | 2/2 | **67.8** | **87.6** | 4/4 | **71.6** | **90.1** |

**SAQ vs. train flat and then quantize.** To further investigate the effectiveness of SAQ, we compare our method with "SAM → SGD" that first obtains a full-precision model with SAM and then trains a quantized model with SGD using full-precision model weights as initialization. We also include "SGD" that trains the quantized models with SGD for comparisons. We report the results of different methods with ResNet-18 on ImageNet. As seen from Table 3.8, for the full-precision model, SAM outperforms SGD by 0.2% on the Top-1 accuracy. However, for 2-bit quantization, SAM → SGD only yields 0.1% Top-1 accuracy improvement compared with SGD. We speculate that smoothing the loss landscape of the pre-trained models provides a better weight initialization for the quantized models. Nevertheless, due to the large distribution gap between the quantized weights and full-precision weights, the performance gain over SGD is limited. In contrast, SAQ performs better than SAM → SGD, which shows the superiority of

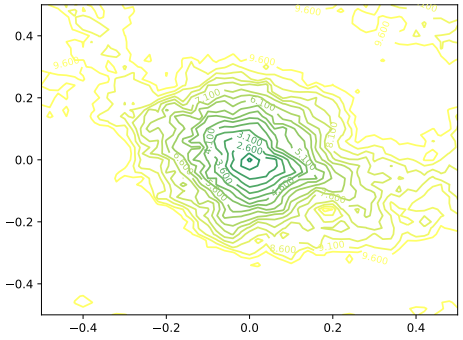
jointly performing quantization and loss landscape smoothing. For example, on ResNet-18, SAQ exceeds SAM → SGD by 0.4% on the Top-1 accuracy. These results suggest that the improvement comes not only from SAM but also from our SAM customization for network quantization.

**More results on transfer learning.** To evaluate the transfer power of different quantized models, we conduct transfer learning experiments on new datasets, including CIFAR-10 [220], CIFAR-100, Oxford-IIIT Pets [221], and Oxford Flowers-102 [222]. We use the quantized models trained on ImageNet to initialize the model weights and then fine-tune all layers using SGD. We repeat the experiments 5 times and report the mean and the standard deviation of the Top-1 accuracy. From Table 3.9, SAQ leads to much better transfer performance. For example, on Oxford-IIIT Pets, SAQ quantized 4-bit ResNet-50 brings 1.0% accuracy improvement over the SGD counterpart. These results justify that SAQ improves the generalization performance by smoothing the loss landscape of the quantized models.

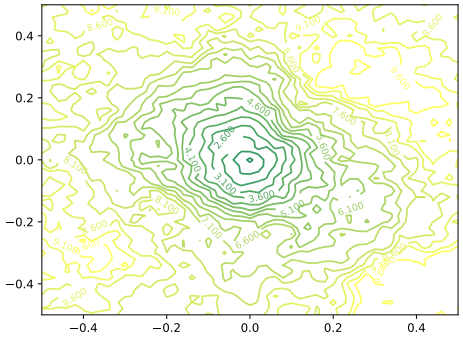
TABLE 3.9: Transfer performance comparisons on downstream tasks. We measure the performance of different methods on 4-bit ResNet-50 using the Top-1 accuracy (%).

| Method | SGD | SAQ (Ours) |
|---|---|---|
| CIFAR-10 [220] | 97.0±0.0 | **97.1±0.1** |
| CIFAR-100 [220] | 82.4±0.2 | **83.1±0.2** |
| Oxford Flowers-102 [221] | 96.1±0.2 | **96.4±0.4** |
| Oxford-IIIT Pets [222] | 94.9±0.2 | **95.9±0.2** |

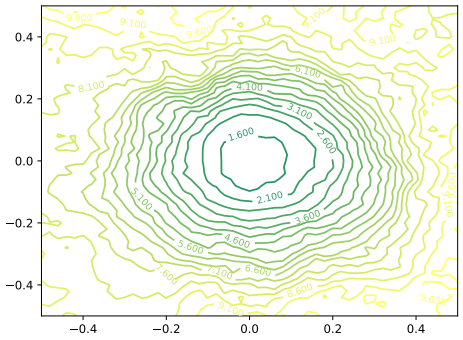
**Visualization of the Loss Landscapes.** In this section, we show the loss landscape of different quantized models on ImageNet using the visualization method in [1]. We show the results in Figures 3.6 and 3.7. The $x$- and $y$-axes of the figures represent two randomly sampled orthogonal directions. From the results, the loss landscapes of the quantized models become smoother and flatter with the increase of bitwidth, suggesting that smoothing the loss landscapes of the 4-bit quantized models is easier than the 2-bit counterparts. Moreover, the loss landscapes of the quantized models obtained by SAQ are less chaotic and show larger contour interval compared with the SGD counterpart, indicating that SAQ is able to find flatter and smoother minima over SGD.



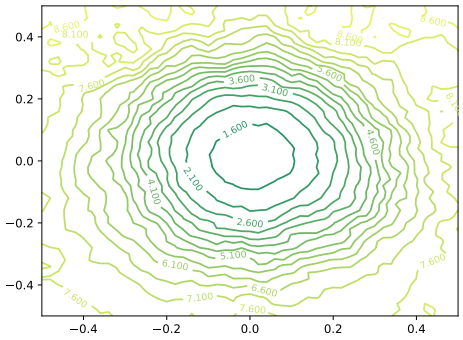
(A) 2-bit ResNet-18 obtained by SGD.



(B) 2-bit ResNet-18 obtained by SAQ

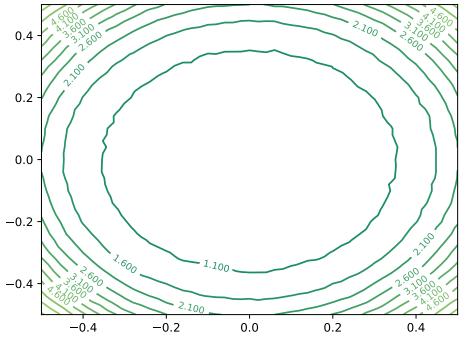


(C) 4-bit ResNet-18 obtained by SGD

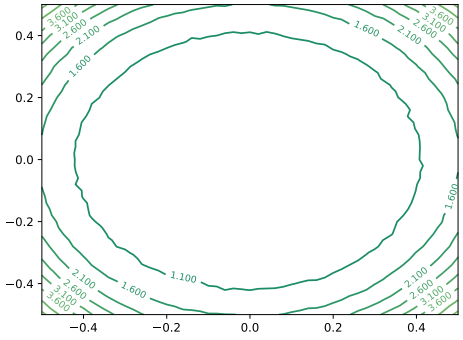


(D) 4-bit ResNet-18 obtained by SAQ

FIGURE 3.6: The loss landscapes of the 2/4-bit ResNet-18 obtained by different methods on ImageNet.



(A) 4-bit ViT-B/16 obtained by SGD



(B) 4-bit ViT-B/16 obtained by SAQ

FIGURE 3.7: The loss landscapes of the 4-bit ViT-B/32 obtained by different methods on ImageNet.

## 3.6 Conclusion

In this chapter, we have devised a new training approach, called Sharpness-Aware Quantization (SAQ), to improve the generalization capability of the quantized models, which jointly performs compression (*i.e.*, quantization) and loss landscape smoothing for the first time. To this end, we have provided a unified view for the loss landscape smoothing

of the quantized models by formulating quantization and SAM as introducing quantization noises and adversarial perturbations to the model weights. According to whether the quantization noises and adversarial perturbations are dependent on each other, we have formulated SAQ into three cases, which have been fully studied and compared. We have further introduced an efficient training strategy that substantially reduces the training overhead, allowing SAQ to achieve comparable training speed to that of the default optimization method. Extensive experiments on various datasets with different architectures including CNNs and Transformers have demonstrated that SAQ consistently improves the performance of the quantized models and yields the state-of-the-art uniform quantization results. In the future, our method can be extended to jointly perform pruning, quantization, and loss landscape smoothing to obtain more compact models with better performance.

# Chapter 4

# EcoFormer: Energy-Saving Attention with Linear Complexity

## 4.1 Overview

Transformer has achieved remarkable performance on a wide range of tasks, but with high computational and energy cost. To improve its efficiency, a popular choice is to compress the models via binarization which constrains the floating-point values into binary ones to save resource consumption owing to cheap bitwise operations significantly. However, existing binarization methods only aim at minimizing the information loss for the input distribution statistically, while ignoring the pairwise similarity modeling at the core of the attention mechanism.

In this chapter, we propose a new binarization paradigm customized to high-dimensional softmax attention via kernelized hashing, called EcoFormer, to map the original queries and keys into low-dimensional binary codes in Hamming space. The kernelized hash functions are learned to match the ground-truth similarity relations extracted from the attention map in a self-supervised way. Based on the equivalence between the inner product of binary codes and the Hamming distance as well as the associative property of matrix multiplication, we can approximate the attention in linear complexity by expressing it as a dot-product of binary codes. Moreover, the compact binary representations of queries and keys in EcoFormer enable us to replace most of the expensive

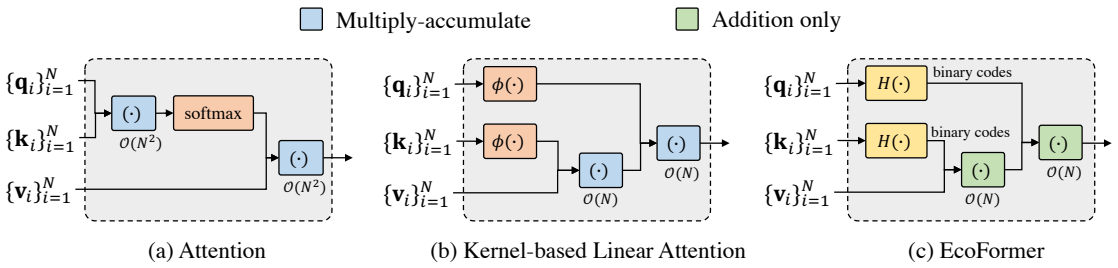



FIGURE 4.1: Computational graphs for standard attention (left), kernel-based linear attention (middle) and the proposed EcoFormer based on kernelized hashing (right).

TABLE 4.1: Energy cost for different operations (on 45nm CMOS technology) [6–8].

| Operation | 16-bit FP Add | 16-bit FP Mult | 32-bit FP Add | 32-bit FP Mult |
|---|---|---|---|---|
| Energy (pJ) | 0.4 | 1.1 | 0.9 | 3.7 |
| Area ($\mu m^2$) | 1,360 | 1,640 | 4,184 | 7,700 |

multiply-accumulate operations in attention with simple accumulations to save considerable on-chip energy footprint on edge devices. Extensive experiments on both vision and language tasks show that EcoFormer consistently achieves comparable performance with standard attentions while consuming much fewer resources.

## 4.2 Introduction

Recently, Transformers [23] have shown rapid and exciting progress in NLP [67, 223] and CV [45, 224] due to its extraordinary representational power. Compared with CNNs [24], Transformer models are generally more scalable to massive amounts of data and better at capturing long-dependency global information with less inductive bias, thus achieving better performance in many tasks [47, 225]. However, the efficiency bottlenecks, especially the high energy consumption, greatly hamper the massive deployment of Transformer models to resource-constrained edge devices, such as mobile phones and unmanned aerial vehicles, for solving a variety of real-world applications.

To reduce the energy consumption, quantization has been actively studied to lower the bit-width representation of network weights [130, 226, 227] and/or activations [31, 228, 229]. With the most aggressive bit-width, binary quantization [41, 42, 133] has attracted much attention since it enables efficient bit-wise operations by representing values with a single bit (*e.g.*, +1 or -1). When we only binarize weights in analogous to BinaryConnect [226], as shown in Table 4.1, it brings great benefits to dedicated hardware by replacing

a large number of energy-hungry multiply-accumulate operations with simple energy-efficient accumulations, which saves significant on-chip area and energy required to run inference with Transformers, making them feasible to be deployed on mobile platforms with limited resources. However, the conventional binarization process typically targets at minimizing quantization errors between the original full-precision data distribution and the quantized Bernoulli distribution statistically. In other words, each token is binarized separately, where the binary representations may not well preserve the original similarity relations among tokens. This motivates us to customize the binarization process to softmax attention, the core mechanism in Transformer that encodes the pairwise similarity between tokens. To this end, we can adapt the well-established hashing methods, to map the high-dimensional queries and keys into compact binary codes that are able to preserve the similarity relations in Hamming space. A simple solution is to use the locality-sensitive hashing [230] to substitute the binary quantization counterparts.

Nevertheless, another energy bottleneck exists in Transformers. Specially, given a sequence of tokens, the softmax attention obtains the attention weights by computing the inner product between a query token and all key tokens, leading to the quadratic time complexity $\mathcal{O}\left(N^2\right)$ regarding the number of tokens $N$, as shown in Figure 4.1 (a). This problem is even worse for a long sequence length $N$, especially for high resolution images in dense prediction tasks. To reduce the complexity of the softmax attention, some prior works propose to express the attention as a linear dot product of kernelized feature embeddings [98, 231]. With the associative property of matrix multiplication, the attention operation can be approximated in linear complexity $\mathcal{O}(N)$, as illustrated by Figure 4.1 (b).

Based on the hashing mechanism and kernel-based formulation of attention, we devise a simple yet effective energy-saving attention, called EcoFormer, which is shown in Figure 4.1 (c). In particular, we propose to use kernelized hashing with RBF kernel to map the queries and keys to compact binary codes. The resulting codes are valid for similarity preserving based on the good property that the codes' inner product (*i.e.*, Hamming affinity) and Hamming distance have one-to-one correspondence [232]. Thanks to the associative property of the linear dot-product between the binary codes, the kernelized hashing attention is in *linear complexity* with significant energy saving. Moreover, the pairwise similarity matrix in attention can be directly used to obtain the supervision labels for hash function learning, delivering a novel *self-supervised* hashing paradigm.

By maximizing the Hamming affinity on the similar pairs of tokens and simultaneously minimizing on the dissimilar pairs of tokens, the pairwise similarity relations between tokens can be preserved. With *low-dimensional binary* queries and keys, we can replace most of the energy-hungry floating-point multiplications in attention with simple additions, which greatly saves the on-chip energy footprint.

Our main contributions are summarized as follows:

- We propose a new binarization paradigm to better preserve the pairwise similarity in softmax attention. In particular, we present EcoFormer, an energy-saving attention with linear complexity powered by kernelized hashing to map the queries and keys into compact binary codes.
- We learn the kernelized hash functions based on the ground-truth Hamming affinity extracted from the attention scores in a self-supervised way.
- Extensive experiments on CIFAR-100, ImageNet-1K and Long Range Arena show that EcoFormer is able to significantly reduce the on-chip energy cost while preserving accuracy. For example, based on PVTv2-B0 and ImageNet-1K, EcoFormer achieves a 73% reduction in on-chip energy footprint with only a marginal performance drop of 0.33% compared to standard attention.

## 4.3 Preliminaries

### 4.3.1 Attention Mechanism

Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be the input sequence into a standard MSA layer, where $N$ is the length of the input sequence and $D$ is the number of hidden dimensions. A standard MSA layer calculates a sequence of query, key and value vectors with three learnable projection matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{D \times D_p}$, which can be formulated as

$$\{\mathbf{q}_t\}_{t=1}^{N} = \mathbf{X}\mathbf{W}_q, \{\mathbf{k}_t\}_{t=1}^{N} = \mathbf{X}\mathbf{W}_k, \{\mathbf{v}_t\}_{t=1}^{N} = \mathbf{X}\mathbf{W}_v, \tag{4.1}$$

where $D_p$ refers to the number of dimensions for each head. For each query vector, the attention output is a weighted-sum over all value vectors as

$$\text{Attention}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) = \sum_i \frac{\exp(\mathbf{q}_t \cdot \mathbf{k}_i/\tau)}{\sum_j \exp(\mathbf{q}_t \cdot \mathbf{k}_j/\tau)} \mathbf{v}_i, \tag{4.2}$$

where $\tau$ is the temperature for controlling the flatness of softmax and $\exp(\langle \cdot, \cdot \rangle)$ is an exponential function. With $N$ token, the computation of attention has a quadratic complexity of $O(N^2)$ in both space and time, which results in huge computational cost when dealing with long sequences.

### 4.3.2 Kernel-based Linear Attention

The idea behind kernel-based linear attention is to express the similarity measure in Eq. (4.2) as a linear dot-product of kernel embeddings, such as polynomial kernel, exponential or RBF kernel. A particular choice is to employ the finite random mapping [233] $\boldsymbol{\phi}(\cdot)$ to approximate the infinite dimensional RBF kernel. Then, according to the theorem from Rahimi [233], the inner product between a pair of transformed vectors $\mathbf{x}$ and $\mathbf{y}$ with $\boldsymbol{\phi}(\cdot)$ can approximate a Guassian RBF kernel. This gives rise to an unbiased estimation to $\exp(\langle \cdot, \cdot \rangle)$ in Eq. (4.2), which can be expressed as

$$\begin{aligned} \exp\left(\mathbf{x} \cdot \mathbf{y}/\sigma^2\right) &= \exp\left(\|\mathbf{x}\|^2/2\sigma^2 + \|\mathbf{y}\|^2/2\sigma^2\right) \exp\left(-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2\right) \\ &\approx \exp\left(\|\mathbf{x}\|^2/2\sigma^2 + \|\mathbf{y}\|^2/2\sigma^2\right) \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{y}). \end{aligned} \tag{4.3}$$

Assume that the queries and keys are unit vectors, then the attention computation in Eq. (4.2) can be approximated by

$$\text{Attention}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) \approx \sum_i \frac{\boldsymbol{\phi}(\mathbf{q}_t)^\top \boldsymbol{\phi}(\mathbf{k}_i)\, \mathbf{v}_i}{\sum_j \boldsymbol{\phi}(\mathbf{q}_t)^\top \boldsymbol{\phi}(\mathbf{k}_j)} \tag{4.4a}$$

$$= \frac{\boldsymbol{\phi}(\mathbf{q}_t)^\top \sum_i \boldsymbol{\phi}(\mathbf{k}_i) \otimes \mathbf{v}_i}{\boldsymbol{\phi}(\mathbf{q}_t)^\top \sum_j \boldsymbol{\phi}(\mathbf{k}_j)}, \tag{4.4b}$$

where $\otimes$ refers to the outer product. Recent works have shown that kernel-based linear attentions perform favorably against the original softmax attention on machine translation [231] and protein sequence modeling [98]. However, although the complexity is reduced into linear, the floating-point multiplications in Eq. (4.4b) still consume a large amount of energy, which can quickly drain the batteries on mobile/edge platforms.

### 4.3.3 Binary Quantization

Following [41], binary quantization typically estimates the full-precision $\mathbf{u} \in \mathbb{R}^n$ using a binary $\hat{\mathbf{u}} \in \{+1, -1\}^n$ and a scaling factor $\alpha \in \mathbb{R}^+$ such that $\mathbf{u} \approx \alpha\hat{\mathbf{u}}$ holds. To find an accurate estimation, existing methods [41, 42, 133, 234, 235] minimize the quantization error as

$$\alpha^*, \hat{\mathbf{u}}^* = \arg\min \|\mathbf{u} - \alpha\hat{\mathbf{u}}\|. \tag{4.5}$$

By solving Problem (4.5), we have $\hat{\mathbf{u}} = \text{sign}(\mathbf{u})$ and $\alpha = \frac{1}{n}\|\mathbf{u}\|_{\ell 1}$, where $\text{sign}(u)$ returns 1 if $u \geq 0$ and -1 if $u < 0$. Since the sign function is non-differentiable, the STE [198] is applied to approximate the gradient such as using the gradient of hard tanh [228] or piecewise polynomial function [143].

## 4.4 Proposed Method

To reduce the energy consumption of self-attention, one may perform binary quantization [42, 133] on the queries $\{\mathbf{q}_t\}_{t=1}^N$ and keys $\{\mathbf{k}_t\}_{t=1}^N$. In this case, we can replace most of the energy-expensive multiplications with the energy-efficient bit-wise operations. However, existing binary quantization methods only focus on minimizing the quantization error between the original full-precision values and the binary ones as in Eq. (4.5), which fails to preserve the pairwise semantic similarity between different tokens in attention, leading to performance degradation.

Note that the attention can be seen as applying kernel smoother over pairwise tokens where the kernel scores denote the similarity of the token pairs, as mentioned in Section 4.3.2. Motivated by this, we propose a new binarization method that applies kernelized hashing with Gaussian RBF to map the original high-dimensional queries/keys to low-dimensional similarity-preserving binary codes in Hamming space. The proposed framework, which we dub *EcoFormer*, is depicted in Figure 4.1 (c). To maintain the semantic similarity in attention, we learn the hash functions in a self-supervised manner. By exploiting the associative property of the linear dot-product between binary codes and the equivalence between the code inner products (*i.e.*, Hamming affinity) and the Hamming distances, we are able to approximate the self-attention in linear time with low energy cost. In the following, we first introduce the kernelized hashing attention in

Section 4.4.1 and then show how to learn the hash functions in a self-supervised way in Section 4.4.2.

### 4.4.1 Kernelized Hashing Attention

Before applying hash functions, we let the queries $\{\mathbf{q}_t\}_{t=1}^N$ and keys $\{\mathbf{k}_t\}_{t=1}^N$ to be identical following [92, 100]. In this way, we can then apply kernelized hash functions $H : \mathbb{R}^{D_p} \mapsto \{1, -1\}^b$ without explicitly applying transformation $\boldsymbol{\phi}(\cdot)$ mentioned in Section 4.3.2 to map $\mathbf{q}_i$ and $\mathbf{k}_j$ into $b$-bit binary codes $H(\mathbf{q}_i)$ and $H(\mathbf{k}_j)$, respectively (see Section 4.4.2). In this case, the Hamming distance between them can be defined as

$$\mathcal{D}\left(H(\mathbf{q}_i), H(\mathbf{k}_j)\right) = \textstyle\sum_{r=1}^{b} \mathbb{1}\{H_r(\mathbf{q}_i) \neq H_r(\mathbf{k}_j)\}, \tag{4.6}$$

where $H_r(\cdot)$ is the $r$-th bit of the binary codes; $\mathbb{1}\{A\}$ is an indicator function that returns 1 if $A$ is satisfied and otherwise returns 0. With $\mathcal{D}\left(H(\mathbf{q}_i), H(\mathbf{k}_j)\right)$, the codes inner product between $H(\mathbf{q}_i)$ and $H(\mathbf{k}_j)$ can be formulated as

$$\begin{aligned} H(\mathbf{q}_i)^\top H(\mathbf{k}_j) &= \sum_{r=1}^{b} \mathbb{1}\{H_r(\mathbf{q}_i) = H_r(\mathbf{k}_j)\} - \sum_{r=1}^{b} \mathbb{1}\{H_r(\mathbf{q}_i) \neq H_r(\mathbf{k}_j)\} \\ &= b - 2\sum_{r=1}^{b} \mathbb{1}\{H_r(\mathbf{q}_i) \neq H_r(\mathbf{k}_j)\} = b - 2\mathcal{D}\left(H(\mathbf{q}_i), H(\mathbf{k}_j)\right). \end{aligned} \tag{4.7}$$

Importantly, Eq. (4.7) shows the equivalence between the Hamming distance and the codes inner product since there is a one-to-one correspondence. By substituting with the hashed queries and keys in Eq. (4.4a), we can approximate the self-attention as

$$\text{Attention}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) \approx \sum_i \frac{H(\mathbf{q}_t)^\top H(\mathbf{k}_i)\mathbf{v}_i}{\sum_j H(\mathbf{q}_t)^\top H(\mathbf{k}_j)}. \tag{4.8}$$

Note that $H(\mathbf{q}_t)^\top H(\mathbf{k}_j) \in [-b, b]$. To avoid zero in denominator, we introduce a bias term $2^c$ to each inner product so that $H(\mathbf{q}_t)^\top H(\mathbf{k}_j) + 2^c > 0$, having no effect on the similarity measure. Here, we can simply set $c$ to $\lceil \log_2(b+1) \rceil$ where $\lceil u \rceil$ returns the least integer greater than or equal to $u$. Using the associative property of matrix

multiplication, we approximate the self-attention as

$$\begin{aligned}\text{Attention}(\mathbf{q}_t, \{\mathbf{k}_i\}, \{\mathbf{v}_i\}) &\approx \sum_i \frac{\left(H(\mathbf{q}_t)^\top H(\mathbf{k}_i) + 2^c\right)\mathbf{v}_i}{\sum_j \left(H(\mathbf{q}_t)^\top H(\mathbf{k}_j) + 2^c\right)} \\ &= \frac{H(\mathbf{q}_t)^\top \sum_i H(\mathbf{k}_i) \otimes \mathbf{v}_i + \sum_i 2^c \mathbf{v}_i^\top}{H(\mathbf{q}_t)^\top \sum_j H(\mathbf{k}_j) + 2^c N}.\end{aligned} \tag{4.9}$$

In practice, the multiplications between the binary codes and the full-precision values in Eq. (4.9) can be replaced by simple additions and subtractions, which greatly reduce the computational overhead in terms of on-chip energy footprint. Moreover, the multiplications with a powers-of-two $2^c$ can also be implemented by efficient *bit-shift* operations. As a result, the only multiplications come from the element-wise divisions between the numerator and denominator.

### 4.4.2 Self-supervised Hash Function Learning

Given queries $\mathcal{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_N\} \subset \mathbb{R}^{D_p}$, we seek to learn a group of hash functions $h : \mathbb{R}^{D_p} \mapsto \{1, -1\}$. Instead of explicitly applying the transformation function $\boldsymbol{\phi}(\cdot)$ mentioned in Section 4.3.2, we compute the hash functions with a kernel function $\kappa(\mathbf{q}_i, \mathbf{q}_j) : \mathbb{R}^{D_p} \times \mathbb{R}^{D_p} \mapsto \mathbb{R}$. Given $\mathbf{Q} = [\mathbf{q}_1, \cdots, \mathbf{q}_N]^\top \in \mathbb{R}^{N \times D_p}$, we randomly sample $m$ queries $\mathbf{q}_{(1)}, \ldots, \mathbf{q}_{(m)}$ from $\mathcal{Q}$ as support samples following the kernel-based supervised hashing (KSH) [232] and define a hash function $h$ as

$$h(\mathbf{Q}) = \text{sign}\left(\sum_{j=1}^m \left(\kappa\left(\mathbf{q}_{(j)}, \mathbf{Q}\right) - \mu_j\right) a_j\right) = \text{sign}\left(\mathbf{g}(\mathbf{Q})\mathbf{a}\right), \tag{4.10}$$

where $\mathbf{a} = [a_1, \cdots, a_m]^\top$ is the weight of $h$, $\mu_j = \frac{1}{n}\sum_{i=1}^N \kappa\left(\mathbf{q}_{(j)}, \mathbf{q}_i\right)$ is to normalize the kernel function to zero-mean, and $\mathbf{g} : \mathbb{R}^{D_p} \mapsto \mathbb{R}^m$ is a mapping defined by $\mathbf{g}(\mathbf{Q}) = \left[\kappa\left(\mathbf{q}_{(1)}, \mathbf{Q}\right) - \mu_1, \ldots, \kappa(\mathbf{q}_{(m)}, \mathbf{Q}) - \mu_m\right] \in \mathbb{R}^{N \times m}$. Then, we define the kernelized hash function $H(\cdot)$ as

$$H(\mathbf{Q}) = [h_1(\mathbf{Q}), \cdots, h_b(\mathbf{Q})] = \begin{bmatrix} h_1(\mathbf{q}_1), \cdots, h_b(\mathbf{q}_1) \\ \cdots\cdots \\ h_1(\mathbf{q}_N), \cdots, h_b(\mathbf{q}_N) \end{bmatrix} = \text{sign}\left(\mathbf{g}(\mathbf{Q})\mathbf{A}\right), \tag{4.11}$$

where $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_b] \in \mathbb{R}^{m \times b}$, and $h_r(\mathbf{Q}) = \text{sign}\left(\mathbf{g}(\mathbf{Q})\mathbf{a}_r\right)$ is the hash function for the $r$-th bit.

To guide the learning of the binary codes, we hope that similar token pairs will have the minimal Hamming distance while dissimilar token pairs will have the maximal distance. Nevertheless, directly optimizing the Hamming distance is difficult due to the non-convex and non-smooth formulation in Eq. (4.6). Utilizing the equivalence between the code inner products and the Hamming distances in Eq. (4.7), we instead conduct optimization based on the Hamming affinity to minimize the reconstruction error as

$$\min_{\mathbf{A}} \left\| H(\mathbf{Q})H(\mathbf{Q})^\top - b\mathbf{Y} \right\|_F^2 = \min_{\mathbf{A}} \left\| \sum_{r=1}^{b} h_r(\mathbf{Q})h_r(\mathbf{Q})^\top - b\mathbf{Y} \right\|_F^2, \tag{4.12}$$

where $\|\cdot\|_F$ is the Frobenius norm and $\mathbf{Y} \in \mathbb{R}^{N\times N}$ is the target Hamming affinity matrix. To preserve the similarity relations between queries and keys, we use the attention scores as the *self-supervised* information to construct $\mathbf{Y}$. Let $\mathcal{S}$ and $\mathcal{U}$ be the similar and dissimilar pairs of tokens. We obtain $\mathcal{S}$ and $\mathcal{U}$ by selecting the token pairs with the Top-$l$ largest and smallest attention scores. We then construct pairwise labels $\mathbf{Y}$ as

$$\mathbf{Y}_{ij} = \begin{cases} 1, & (\mathbf{q}_i, \mathbf{q}_j) \in \mathcal{S} \\ -1, & (\mathbf{q}_i, \mathbf{q}_j) \in \mathcal{U} \\ 0, & \text{otherwise.} \end{cases} \tag{4.13}$$

However, Problem (4.12) is NP-hard. To solve it efficiently, we adapt discrete cyclic coordinate descent to learn binary codes sequentially. Specifically, we only solve $\mathbf{a}_r$ once the previous $\mathbf{a}_1, \cdots, \mathbf{a}_{r-1}$ have been optimized. Let $\hat{\mathbf{Y}}_{r-1} = b\mathbf{Y} - \sum_{t=1}^{r-1} h_t(\mathbf{Q})h_t(\mathbf{Q})^\top$ be the residual matrix, where $\hat{\mathbf{Y}}_0 = b\mathbf{Y}$. Then, we can minimize the following objective to obtain $\mathbf{a}_r$

$$\min_{\mathbf{a}_r} \quad \left\| h_r(\mathbf{Q})h_r(\mathbf{Q})^\top - \hat{\mathbf{Y}}_{r-1} \right\|_F^2 = \min_{\mathbf{a}_r} -2h_r(\mathbf{Q})^\top \hat{\mathbf{Y}}_{r-1} h_r(\mathbf{Q}) + C, \tag{4.14}$$

where $C = \left(h_r(\mathbf{Q})^\top h_r(\mathbf{Q})\right)^2 + \mathrm{tr}\left(\hat{\mathbf{Y}}_{r-1}\right)$ is a constant. Note that $\hat{\mathbf{Y}}_{r-1}$ is a symmetric matrix. Therefore, Problem (4.14) is a standard binary quadratic programming problem, which can be efficiently solved by many existing methods, such as the LBFGS-B solver [236] and block graph cuts [237]. To learn $\mathbf{a}_r$ in conjunction with network parameters, we propose to solve Problem (4.14) using the gradient-based methods. For the non-differentiable sign function, we use STE [198] to approximate the gradient using hard tanh as mentioned in Section 4.3.3. Note that learning the hash functions for each epoch

is computationally expensive yet unnecessary. We only learn the hash functions per $\tau$ epoch.

TABLE 4.2: Main results on ImageNet-1K. The number of multiplications, additions, as well as on-chip energy consumption are calculated based on an image of resolution $224 \times 224$. The throughput is measured with a mini-batch size of 32 on a single NVIDIA RTX 3090 GPU.

| Model | Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Throughput (images/s) | Top-1 Acc. (%) |
|---|---|---|---|---|---|---|
| PVTv2-B0 [9] | MSA | 2.02 | 1.99 | 9.25 | 850 | 70.77 |
| | **Ours** | **0.54** | **0.56** | **2.49** | **1379** | **70.44** |
| PVTv2-B1 | MSA | 5.02 | 5.00 | 23.07 | 621 | 78.83 |
| | **Ours** | **2.03** | **2.09** | **9.39** | **874** | **78.38** |
| PVTv2-B2 | MSA | 8.64 | 8.60 | 39.71 | 404 | 81.82 |
| | **Ours** | **3.85** | **3.97** | **17.82** | **483** | **81.28** |
| PVTv2-B3 | MSA | 11.86 | 11.82 | 54.56 | 310 | 82.26 |
| | **Ours** | **6.54** | **6.72** | **30.25** | **325** | **81.96** |
| PVTv2-B4 | MSA | 15.97 | 15.93 | 73.43 | 247 | 82.42 |
| | **Ours** | **9.57** | **9.82** | **44.25** | **249** | **81.90** |
| Twins-SVT-S [238] | MSA | 5.96 | 5.91 | 27.36 | 426 | 81.66 |
| | **Ours** | **2.72** | **2.81** | **12.59** | **576** | **80.22** |

## 4.5 Experiments

### 4.5.1 Comparisons on ImageNet-1K

To investigate the effectiveness of the proposed method, we conduct experiments on ImageNet-1K [24], a large-scale image classification dataset that contains ∼1.2M training images from 1K categories and 50K validation images. We compare our kernelized hashing attention with standard MSA by adapting the two attention approaches into two popular vision Transformer frameworks PVTv2 [9] and Twins [238]. We measure model performance by the Top-1 accuracy. Furthermore, as FLOPs cannot accurately reflect the computational cost in our proposed method, we measure the model complexity by the number of multiplications and additions, separately, as done in [107]. Specifically, we calculate FLOPs following [46], where we count the multiply-accumulate operations for all layers. In this case, each multiply-accumulate operation consists of an addition and a multiplication. We also count the multiplications in the scaling operations. Moreover, we report the on-chip energy consumption according to Table 4.1 and the throughput with a mini-batch size of 32 on a single NVIDIA RTX 3090 GPU.

**Implementation details.** All training images are resized to $256 \times 256$, and $224 \times 224$ patches are randomly cropped from an image or its horizontal flip, with the per-pixel

mean subtracted. To obtain the MSA baselines, we first replace the original attention layers in PVTv2 [9] and Twins [238] with standard MSAs and initialize the models with the pretrained weights on ImageNet-1K. Next, we finetune each model on ImageNet-1K with 100 epochs. Based the pretrained MSA weights, we then apply our approach to each model and finetune on ImageNet-1K with 30 epochs. All models in this experiment are trained on 8 V100 GPUs with a total batch size of 256. We set the initial learning rate to $2.5 \times 10^{-5}$ for PVTv2 and $1.25 \times 10^{-4}$ for Twins. We use AdamW optimizer [206] with a cosine decay learning rate scheduler. All other hyperparameters are the same as in PVTv2. Also note that recent hierarchical ViTs [9, 47, 238] have multiple stages to incorporate pyramid feature maps. At the last stage, they usually apply standard MSAs due to the comparably low-resolution feature maps. This design is also adopted in PVTv2 and Twins. Therefore, we follow the common practice and do not modify the attention layers at the last stage. For the hash functions learning, we set the number of support samples $m$ and update interval $\tau$ to 25 and 30, respectively. The hyper-parameter $l$ in constructing pairwise labels $\mathbf{Y}$ is set to 10. The hash bit hyper-parameter, denoted as $b$, is set to 16.

**Results analysis.** We report the results in Table 4.2. In general, our baseline MSA has more multiplications than additions. In contrast, our EcoFormer replaces most of the floating-point multiplications in attention with simple additions. Therefore, there are more additions than multiplications in our EcoFormer. Compared to MSA, our method achieves lower computational complexity, less energy consumption and higher throughput with comparable performance. For example, based on PVTv2-B0, our method saves around 73% multiplications and 72% additions, as well as reducing 73% on-chip energy consumption, which demonstrates the energy-efficiency of our approach. With more efficient accumulation implementation, the throughput of our EcoFormer can be further improved. Besides, a larger model comes with a larger proportion of computational and on-chip energy cost dominated by FFNs, as shown in Figure 4.2. In this case, as our approach focuses on the attention layers, the energy-saving from larger models is comparably less than smaller models (*e.g.*, PVTv2-B0 vs. PVTv2-B4). Nonetheless, we still reduce significant computational cost and on-chip energy consumption. For example, on PVTv2-B4, we save around 40% on-chip energy consumption. Compared with PVTv2, the performance drop of our method is slightly larger on Twins. One possible reason is that our method may be sensitive to the conditional positional encodings in Twins.

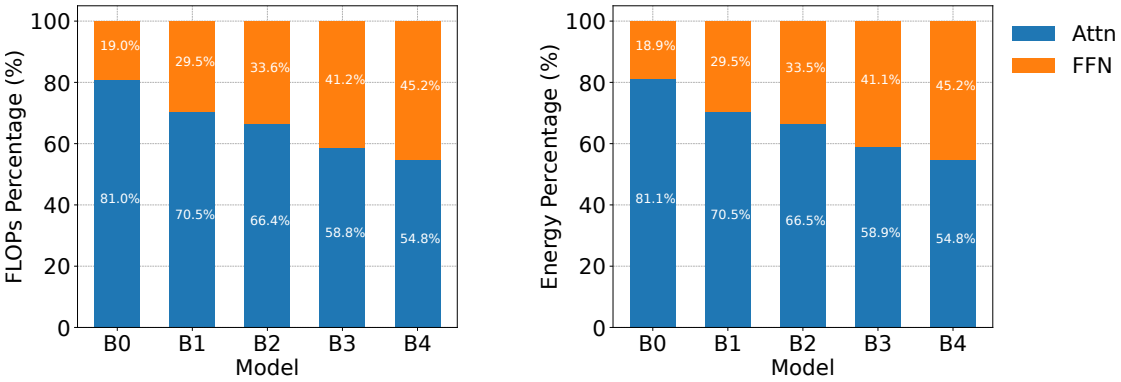



FIGURE 4.2: FLOPs and on-chip energy footprint percentage of attention layers (Attn) and feed-forward layers (FFN) in different variants of PVTv2 with standard MSAs. For a bigger model, FFN takes a larger proportion of the computational cost and on-chip energy footprint.

### 4.5.2 Comparisons on Long Range Arena

To evaluate the performance of different efficient attentions under long-context scenarios, we train our EcoFormer on two tasks, **Text** and **Retrieval** from the Long Range Arena (LRA) benchmark [239] following the settings of [240]. Our implementations are based on the released code of [99]. We use the same hyper-parameters $m$, $\tau$ and $b$ as in ImageNet-1K experiments. We show the results in Table 4.3. From the table, our EcoFormer achieves comparable performance with much lower on-chip energy consumption. For example, on **Text**, compared with MSA, our method saves around 94.6% multiplications and 93.7% additions as well as 94.5% on-chip energy consumption, which is more efficient than existing attention mechanisms.

TABLE 4.3: Comparisons of different methods on Long Range Arena (LRA). We report the classification accuracy (%) for **Text** as well as **Retrieval** and average accuracy across two tasks. * denotes that we obtain the results from the original paper.

| Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Text (4K) | Retrieval (4K) | Average |
|---|---|---|---|---|---|---|
| Transformer | 4.63 | 4.57 | 21.25 | 64.87 | 79.62 | 72.25 |
| Performer [98] | 0.83 | 0.84 | 3.83 | 64.82 | 79.08 | 71.95 |
| Linformer [93] | 0.81 | 0.81 | 3.74 | 57.03 | 78.11 | 67.57 |
| Reformer [92] | 0.54 | 0.54 | 2.49 | 65.19 | 79.46 | 72.33 |
| Combiner* [241] | 0.51 | 0.51 | 2.34 | 64.36 | 56.10 | 60.23 |
| **EcoFormer** | **0.25** | **0.29** | **1.17** | 64.79 | 78.67 | 71.73 |

### 4.5.3 Ablation Study

In this section, we evaluate the effectiveness of our EcoFormer by comparing it with different binarization approaches and efficient attention mechanisms. By default, we

train each model from scratch on CIFAR-100 with 2 GPUs for 300 epochs. The total batch size is 64. The initial learning rate is $6.25 \times 10^{-6}$. For the hash functions learning, we set update interval $\tau$ to 300. All the other hyperparameters are the same as in ImageNet-1K experiments.

TABLE 4.4: Performance comparisons with different binarization methods on CIFAR-100.

| Model | Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Top-1 Acc. (%) |
|---|---|---|---|---|---|
| PVTv2-B0 | FP-EcoFormer | 0.94 | 0.94 | 4.33 | 70.78 |
| | Bi-EcoFormer | 0.63 | 0.83 | 3.09 | 70.06 |
| | **EcoFormer** | **0.54** | **0.56** | **2.49** | **71.23** |
| Twins-SVT-S | FP-EcoFormer | 5.96 | 5.91 | 27.36 | 80.04 |
| | Bi-EcoFormer | 3.01 | 3.59 | 14.38 | 80.04 |
| | **EcoFormer** | **2.72** | **2.81** | **12.58** | **80.31** |

**Quantization vs. hashing.** To investigate the effect of different binarization methods, we compare our EcoFormer with the following methods: **FP-EcoFormer**: Based on EcoFormer, we do not binarize queries and keys in attentions. **Bi-EcoFormer**: Relying on EcoFormer, we use the same binary quantization [228] as BinaryBERT [42] and BiBERT [133] to obtain binarized queries and keys instead of our proposed hash functions. For fair comparisons, the attention operations in the compared method are in linear complexity. We apply different methods to PVTv2-B0 and Twins-SVT-S and report the results in Table 4.4. We observe that our EcoFormer consistently outperforms Bi-EcoFormer on different frameworks. For example, based on PVTv2-B0, our EcoFormer surpasses Bi-EcoFormer by 1.17% in terms of the Top-1 accuracy. Compared with binary quantization, our proposed self-supervised hash functions preserve the pairwise similarity of attention, leading to better performance. Moreover, our EcoFormer does not need to explicitly compute transformation $\phi(\cdot)$ as in Eq. (4.4b). Therefore, the energy cost of our EcoFormer is lower than Bi-EcoFormer.

TABLE 4.5: Performance comparisons with different hash functions regarding PVTv2-B0 on CIFAR-100.

| Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Top-1 Acc. (%) |
|---|---|---|---|---|
| LSH-EcoFormer | 0.68 | 0.69 | 3.12 | 70.18 |
| KLSH-EcoFormer | 0.54 | 0.56 | 2.49 | 70.66 |
| **EcoFormer** | **0.54** | **0.56** | **2.49** | **71.23** |

**Effect of different hash functions.** To investigate the effect of different hash functions, we include the following methods for comparisons: **LSH-EcoFormer**: Relying on EcoFormer, we use Locality-Sensitive Hashing (LSH) [242] rather than our proposed

kernelized hash function. **KLSH-EcoFormer**: Based on EcoFormer, we replace the proposed hash function with Kernelized Locality-Sensitive Hashing (KLSH) [243]. We report the results in Table 4.5. We can observe that KLSH-EcoFormer outperforms LSH-EcoFormer by 0.48% in terms of the Top-1 accuracy with less on-chip energy consumption. The reason can be attributed to that LSH is based on random linear projection, which can not deal with the non-linear softmax attention well. Critically, our EcoFormer further improves the performance by 0.57% on the Top-1 accuracy. Compared with random hashing in KLSH, our EcoFormer learns the hash functions with additional self-supervised information. Therefore, our learned binary codes are better at preserving the token similarity.

TABLE 4.6: Comparison with other efficient attention methods regarding PVTv2-B0 [9] on CIFAR-100.

| Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Top-1 Acc. (%) |
|---|---|---|---|---|
| Transformer | 2.02 | 1.99 | 9.25 | 71.44 |
| Performer [98] | 0.94 | 0.94 | 4.33 | 70.78 |
| Linformer [93] | 0.69 | 0.69 | 3.18 | 71.17 |
| Reformer [92] | 1.62 | 1.63 | 7.44 | 70.56 |
| **EcoFormer** | **0.54** | **0.56** | **2.49** | **71.23** |

TABLE 4.7: Latency and energy comparisons with different attention methods. We measure the latency and energy of an attention layer with a batch size of 16, a sequence length of 3,136 and an embedding dimension of 32 on a BitFusion [10] simulator.

| Method | Latency (ms) | Energy (pJ) |
|---|---|---|
| Transformer | 0.0036 | 85,692.18 |
| Performer [98] | 0.0019 | 41,113.64 |
| Linformer [93] | 0.0018 | 45,770.61 |
| Reformer [92] | 0.0024 | 57,305.47 |
| **EcoFormer** | **0.0010** | **24,990.75** |

**Comparing with other efficient attention mechanisms.** To compare EcoFormer with different attention mechanisms, we conduct experiments on CIFAR-100 based on PVTv2-B0 in Table 4.6. In our experiments, we directly replace the attention layers with each compared method in PVTv2-B0 [9]. In general, compared to other efficient attention mechanisms, EcoFormer saves more computations and reduces more on-chip energy consumption while achieving better performance. Particularly, benefiting from the multiplication-saving operations and low-dimensional binary queries and keys, EcoFormer saves more on-chip energy than Performer. Also note that since the proposed kernelized hash function $H(\cdot)$ does not need to explicitly apply transformation $\phi(\cdot)$ to

the queries and keys as in Eq. (4.4b), EcoFormer simultaneously reduces more multiplications and additions than Performer. Besides, Linformer achieves competitive results. However, as the size of the learnable low-rank projection parameters depends on the length of the input sequence, Linformer is not scalable to different image resolutions, whereas EcoFormer with sufficient bits is agnostic to the sequence length.

**Latency and energy on BitFusion [10].** To show the actual energy consumption and latency, we test different methods on a simulator of BitFusion, a bit-flexible microarchitecture synthesized in 45 nm technology. From Table 4.7, EcoFormer shows much lower latency and on-chip energy than the other efficient attention methods, which further verifies the advantage of EcoFormer.

**Effect of training from scratch on ImageNet-1K.** To explore the effect of training from scratch, we apply EcoFormer to PVTv2-B0 and PVTv2-B1. We follow the experimental settings mentioned in Section 4.5.1 except that we train the model from scratch with 300 epochs. The initial learning rate is set to $2.5\times10^{-4}$. From Table 4.8, EcoFormer achieves comparable performance while significantly reducing the computational complexity and on-chip energy consumption. The accuracy drop from discretization comes from the gradient approximation for the non-differentiable sign function, which can be mitigated by more advanced optimization methods, such as regularization [148], knowledge distillation [133, 244], relaxed optimization [245, 246], appending full-precision branches [143, 234], *etc.*

TABLE 4.8: Performance comparisons of different methods on ImageNet-1K. All the models are trained from scratch. The number of multiplications, additions, and on-chip energy consumption are calculated based on an image of resolution $224 \times 224$.

| Model | Method | #Mul. (B) | #Add. (B) | Energy (B pJ) | Top-1 Acc. (%) |
|---|---|---|---|---|---|
| PVTv2-B0 | MSA | 2.02 | 1.99 | 9.25 | 69.72 |
| | **Ours** | **0.54** | **0.56** | **2.49** | **68.70** |
| PVTv2-B1 | MSA | 5.02 | 5.00 | 23.07 | 78.34 |
| | **Ours** | **2.03** | **2.09** | **9.39** | **77.49** |

**Effect of different $m$.** To investigate the effect of different numbers of support samples $m$, we train EcoFormer with different $m$ based on PVTv2-B0. We report the results on CIFAR-100 in Table 4.9. As we increase $m$, the performance becomes better along with the increase in on-chip energy consumption. For example, the model obtained with $m = 15$ outperforms that of $m = 10$ by 0.19% on the Top-1 accuracy with little additional energy cost. We speculate that, with more support samples, we can capture

more accurate statistics in Eq. (4.10) and hence lead to better performance. Since our EcoFormer achieves the best performance with $m = 25$, we use it by default in our experiments.

TABLE 4.9: Performance comparisons with different #support samples $m$. We report the results of PVTv2-B0 on CIFAR-100.

| $m$ | #Mul. (B) | #Add. (B) | Energy (B pJ) | Top-1 Acc. (%) |
|---|---|---|---|---|
| 10 | 0.53 | 0.55 | 2.46 | 70.73 |
| 15 | 0.53 | 0.56 | 2.47 | 70.92 |
| 20 | 0.53 | 0.56 | 2.48 | 70.81 |
| 25 | 0.54 | 0.56 | 2.49 | **71.23** |

**Cost saving for attentions only.** We show the results of cost saving for attentions only using different architectures on ImageNet-1K. From Table 4.10, EcoFormer consistently saves massive on-chip energy footprint. In particular, on PVTv2-B0, we save 93% on-chip energy footprint. On larger models, the saving is still significant. Note that larger models also come with more computational cost from the linear projection layers in MSAs, as shown in Figure 4.2. Since EcoFormer does not target these projection layers, larger models save slightly less energy in MSAs compared to those of smaller models.

TABLE 4.10: The cost saving for attentions only, excluding other types of layers. The number of multiplications, additions, as well as on-chip energy consumption are calculated based on an image of resolution $224 \times 224$.

| Model | Method | #Mul. (B) | #Add. (B) | Energy (B pJ) |
|---|---|---|---|---|
| PVTv2-B0 [9] | MSA | 1.58 | 1.56 | 7.26 |
| | **Ours** | **0.10** | **0.13** | **0.50** (**-93%**) |
| PVTv2-B1 | MSA | 3.38 | 3.36 | 15.52 |
| | **Ours** | **0.39** | **0.45** | **1.84** (**-88%**) |
| PVTv2-B2 | MSA | 5.59 | 5.56 | 25.69 |
| | **Ours** | **0.80** | **0.92** | **3.80** (**-85%**) |
| PVTv2-B3 | MSA | 6.85 | 6.81 | 31.49 |
| | **Ours** | **1.53** | **1.71** | **7.21** (**-77%**) |
| PVTv2-B4 | MSA | 8.63 | 8.59 | 39.69 |
| | **Ours** | **2.24** | **2.49** | **10.52** (**-74%**) |
| Twins-SVT-S [238] | MSA | 4.01 | 3.96 | 18.41 |
| | **Ours** | **0.77** | **0.86** | **3.63** (**-80%**) |

**More throughput results on ImageNet-1K.** To show the actual inference speed on a hardware device, we measure the throughput of different methods on a single NVIDIA RTX 3090 GPU. We compare EcoFormer with the standard multi-head self-attention (MSA) and kernel-based linear attention (KLA) [98]. From Table 4.11, KLA shows

higher throughput than MSA, while our EcoFormer achieves even faster throughput than KLA, thanks to the reduced feature dimensions ($b$ vs. $D_p$) of queries and keys. With efficient energy-efficient accumulation implementation, the throughput of our EcoFormer can be further improved, which will be explored in the future.

TABLE 4.11: Throughput (images/s) of different methods on ImageNet-1K. MSA denotes the standard multi-head self-attention and KLA represents the kernel-based linear attention. The throughput is measured with a mini-batch size of 32 and an image resolution of $224 \times 224$ on a single NVIDIA RTX 3090 GPU.

| Method | PVTv2-B0 | PVTv2-B1 | PVTv2-B2 | Twins-SVT-S |
|---|---|---|---|---|
| MSA | 850 | 621 | 404 | 426 |
| KLA [98] | 1166 | 769 | 444 | 489 |
| Ours | **1379** | **874** | **483** | **576** |

## 4.6 Conclusion and Future Work

In this chapter, we have presented a novel energy-saving attention mechanism with linear complexity to save the vast majority of multiplications from a new binarization perspective, making the deployment of Transformer models at scale feasible on edge devices. We are inspired by the fact that conventional binarization methods are built upon statistical quantization error minimization without considering to preserve the pairwise similarity relations between tokens. To this end, we customize binarization to softmax attention by mapping the original token features into compact binary codes in Hamming space using a set of kernel-based hash functions, where the similarity can be measured by codes dot product. The hash functions for queries/keys are learned to encourage the Hamming affinity of a token pair to be close to the target obtained from the attention scores, in a self-supervised way. Extensive experiments have demonstrated that EcoFormer saves significant on-chip energy footprint while achieving comparable performance with standard attentions on ImageNet-1K, Long Range Arena and CIFAR-100. In terms of the future work, we can further binarize the value vectors in attention, multi-layer perceptrons and non-linearities in Transformer to make it fully binarized for more significant on-chip energy-saving. We may also extend EcoFormer to other NLP tasks such as machine translation and speech analysis tasks to make it more impactful to wider communities.

# Chapter 5

# QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models

## 5.1 Overview

LLMs have demonstrated unparalleled efficacy in natural language processing. However, their high computational demands and memory overheads hinder their broad deployment. To address this, two quantization strategies emerge, including QAT and PTQ. For LLMs, the billions of parameters make the QAT impractical due to the prohibitive training cost and thus PTQ becomes more prevalent. In existing studies, activation outliers in particular channels are identified as the biggest challenge to PTQ accuracy. They propose to transform the magnitudes from activations to weights, which however offers limited alleviation or suffers from unstable gradients, resulting in a severe performance drop at low-bitwidth.

In this chapter, we propose QLLM, an accurate and efficient low-bitwidth PTQ method designed for LLMs. QLLM introduces an adaptive channel reassembly technique that reallocates the magnitude of outliers to other channels, thereby mitigating their impact on the quantization range. This is achieved by channel disassembly and channel assembly, which first breaks down the outlier channels into several sub-channels to ensure a more balanced distribution of activation magnitudes. Then similar channels are merged to

maintain the original channel number for efficiency. Additionally, an adaptive strategy is designed to autonomously determine the optimal number of sub-channels for channel disassembly. To further compensate for the performance loss caused by quantization, we propose an efficient tuning method that only learns a small number of low-rank weights while freezing the pre-trained quantized model. After training, these low-rank parameters can be fused into the frozen weights without affecting inference. Extensive experiments on LLaMA-1 and LLaMA-2 show that QLLM is able to obtain accurate quantized models efficiently.

## 5.2 Introduction

Recently, LLMs such as GPT-4 [26] and LLaMA [39, 81] have achieved unprecedented advancements in NLP. These models excel in a range of tasks, from advanced reasoning in code and mathematics to classification and question answering. However, their extraordinary performance is accompanied by substantial computational demands and vast model sizes. For example, GPT-3 [27], the precursor to GPT-4, already contains a stunning 175 billion parameters, requiring a minimum of 325 GB of memory for storage in half-precision (FP16) format. This necessitates the use of at least 5×80GB NVIDIA A100 or 8×48GB NVIDIA A40 GPUs during the inference phase. As a result, deploying these models to real-world applications poses significant challenges.

In light of the aforementioned challenges, network quantization [31] emerges as a compelling solution, which maps weights and/or activations to lower-bit representations, resulting in a much lower memory footprint and faster inference. Existing quantization methods for LLMs can be classified into two types: QAT [34] and PTQ [2, 36, 180]. Although with promising performance, QAT suffers from unbearable training costs as it needs to fine-tune the whole quantized model with quantization parameters using a large amount of data, rendering it impractical for the efficient deployment of LLMs. This practical limitation has shifted the spotlight towards PTQ which only uses a little data to tune the quantized weights. However, when it comes to extremely low-bitwidth quantization for LLMs, *e.g.*, 4-bit weight and/or activation quantization, existing PTQ methods [2, 43] suffer from significant performance degradation.

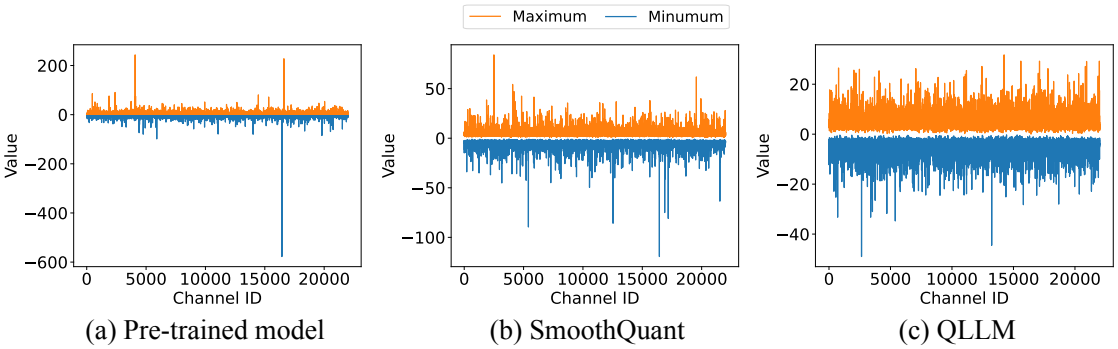



FIGURE 5.1: An illustration of the channel-wise maximum and minimum values for the input activations of a linear layer in LLaMA-65B for (a) original pre-trained model (b) after SmoothQuant [2] and (c) after our channel reassembly.

Recent studies [2, 43, 180] have revealed a unique pattern in LLMs' activations that is they contain specific outlier channels with significantly large magnitudes. This renders existing quantization methods less effective, as the outliers amplify the quantization range of layer activations, causing the vast majority of normal activation values to be quantized imprecisely and consequently leading to notable performance degradation. This issue will worsen with the prevalent use of layer-wise or token-wise activation quantization, a common practice for maximizing hardware efficiency. To tackle this challenge, recent studies [2, 36, 164, 180] have focused on smoothing activation outliers by transitioning the magnitudes from activations to weights through a mathematically equivalent transformation. Such a transformation can be learned using either gradient-free methods [2, 36, 180] or gradient-based methods [164]. However, as shown in Figure 5.1, for exceedingly pronounced activation outliers (those 50× larger than others), the former offers only limited alleviation while the latter suffers from unstable gradients. As a result, both methods leads to significant performance degradation in low-bitwidth quantization. To compensate for the performance drop of quantization, a widely adopted PTQ strategy [164, 180, 181] further proposes to tune the quantized LLM directly by minimizing the block-wise reconstruction error. In LLMs, the tuned block refers to the Attention-FFN module. However, considering the huge number of parameters in an LLM, this approach still requires substantial training overheads and demands a significant amount of GPU memory.

In this chapter, we propose QLLM, an accurate and efficient low-bitwidth post-training quantization method tailored for LLMs. To handle the outlier issue, we introduce a gradient-free channel reassembly technique that redistributes the large activation magnitude of the outlier channels across the channels. Specifically, we first disassemble the

outlier channels into several sub-channels. By spreading the magnitude of outliers, it ensures a more uniform activation range across channels, facilitating a balanced and precise quantization and thus improving the performance of quantized LLMs. We then introduce channel assembly, which fuses similar channels together to maintain the original channel count. Moreover, given the varying outlier patterns across different layers and the existence of extreme outliers, we propose an adaptive strategy to determine the optimal number of disassembled channels for each layer, which is based on minimizing the reassembly error between the original output activations and the counterpart with the reassembled input activations.

To further improve the performance of the quantized LLMs, motivated by low-rank parameter-efficient fine-tuning paradigm LoRA [175, 247], we further propose an efficient gradient-based error correction strategy that freezes the pre-trained model and introduces a small set of learnable low-rank weights into each layer of the LLM. Then, QLLM learns the low-rank weights by minimizing block-wise quantization error sequentially. Owing to the reduced number of trainable parameters, both the training time and GPU memory requirements are significantly reduced. Such efficiency gain enables us to perform a multi-block reconstruction that simultaneously reconstructs a collection of consecutive Attention-FFN blocks, further mitigating the quantization error accumulation during propagation in low-bit LLMs. Notably, after training, these learnable low-rank weights can be seamlessly merged with the frozen weights followed by quantization, thereby ensuring no additional computational burden during inference.

Our contributions can be summarized as follows:

- We introduce a simple yet effective channel reassembly method to suppress activation outliers in LLMs, which is accomplished by initially disassembling the outlier channels to make activations more quantization-friendly and subsequently merging similar channels so as to preserve the original channel count for efficiency. We also propose to determine the optimal number of disassembled channels for each layer, considering the diverse outlier patterns across layers and the presence of extreme outliers. The overall process is gradient-free and enjoys high efficiency.
- An efficient error correction mechanism is proposed to further enhance the performance of gradient-free channel reassembly. It leverages the learning of low-rank parameters to counteract quantization error in a structured way, leading to a substantial

reduction in training time and GPU memory requirements without incurring any additional inference overhead.

- Extensive experiments show the promising performance and training efficiency of QLLM. For example, QLLM quantizes 4-bit LLaMA-2-70B within 10 hours, and outperforms previous state-of-the-art methods by 7.89% on the average accuracy across five zero-shot tasks.

## 5.3 Preliminaries

**Basic notations.** Matrix is marked as $\mathbf{X}$ and vector is denoted by $\mathbf{x}$. The LLMs usually have two core parts: MSA layers and FFN layers, which are mainly composed of linear layers. Here, we give the formulation of linear layers at the output channel $k$:

$$\mathbf{y}_k = \sum_{i=1}^{M} \mathbf{x}_i \mathbf{W}_{ik}, \tag{5.1}$$

where $\mathbf{x} \in \mathbb{R}^M$ refers to input, $\mathbf{W} \in \mathbb{R}^{M \times N}$ denotes the weight, and $\mathbf{y} \in \mathbb{R}^N$ stands for the output. In this way, the numbers of input and output channels are $M$ and $N$, respectively.

**Quantization.** We adopt uniform quantization for both weights and activations because of its hardware-friendly nature [32]. For matrix $\mathbf{X}$ with floating-point values such as FP16, the $b$-bit quantization quantizes it in the following way:

$$\begin{aligned} &\mathbf{X}_q = \text{quant}(\mathbf{X}) = \text{clamp}\left(\lfloor \tfrac{\mathbf{X}}{\alpha} \rceil + \beta, 0, 2^b - 1\right), \\ &\text{where } \alpha = \tfrac{\max(\mathbf{X}) - \min(\mathbf{X})}{2^b - 1}, \beta = -\left\lfloor \tfrac{\min(\mathbf{X})}{\alpha} \right\rceil, \end{aligned} \tag{5.2}$$

where $\text{clamp}(v, v_{\min}, v_{\max})$ clips any value $v$ into the range of $[v_{\min}, v_{\max}]$ and $\lfloor \cdot \rceil$ is a rounding operator that returns the nearest integer of a given value. Here, $\alpha$ denotes the scaling factor and $\beta$ represents the zero-point value.

Recent studies [2, 36, 43] point out that there are extremely large outliers in certain channels of activations in LLMs, which makes the quantization challenging to balance the accurate representation for large values and small numbers. To tackle this problem, some approaches [183, 248] adopt fine-grained quantization scheme, which assigns different quantization parameters for different channels. However, such a way needs delicate

kernel design and clearly increases computation overhead for inference. Also, some works [2, 36] propose to use channel-wise scaling between activation and weights, which still remains outliers under extreme cases, as shown in Figure 5.1.

## 5.4 Proposed Method

In this section, we propose the adaptive channel reassembly framework to redistribute input activation outliers across multiple channels. The framework consists of three components: channel disassembly for decomposing the outlier channel, channel assembly for balancing the efficiency, and an adaptive strategy to find the suitable reassembly ratio for each layer. The channel reassembly technique is gradient-free and efficient to implement. Furthermore, it can be equipped with a gradient-based and efficient error correction module for further enhancement.

### 5.4.1 Adaptive Channel Reassembly

**Channel Disassembly.** In this part, we introduce our channel disassembly to decompose the input outlier channels into several sub-channels, which can reduce the outlier magnitude and make the activations more quantization-friendly without altering the layer output.

Considering that outliers tend to be concentrated in specific channels across various inputs and the desire to preserve their information during quantization, we propose to break down these outlier channels into several sub-channels to redistribute their large values. Without loss of generality, by assuming the $M$-th channel as the outlier channel, we can disassemble it into $\frac{\mathbf{x}_M}{T}$ and replicate this channel $T$ times, reducing the outlier magnitude by a factor of $T$. Simultaneously, it is also natural to duplicate the corresponding weight channel $T$ times, enabling us to maintain the equivalent output:

$$\mathbf{y}_k = \sum_{i=1}^{M-1} \mathbf{x}_i \mathbf{W}_{ik} + \underbrace{\frac{\mathbf{x}_M}{T}\mathbf{W}_{Mk} + \cdots + \frac{\mathbf{x}_M}{T}\mathbf{W}_{Mk}}_{T \text{ times}}. \tag{5.3}$$

The equation above produces the same output with the original linear layer equation in Eq. (5.1) and introduces an additional $T-1$ channels for both the input and the weight.

Taking into account that the quantization range impacts accuracy, we introduce an outlier threshold, denoted as $\theta$, to identify the outlier channels and determine the number of sub-channels together, with $T = \lceil \max(|\mathbf{x}_M|)/\theta \rceil$. This approach ensures that channels with values smaller than $\theta$ remain unchanged with $T = 1$, while the magnitude of outliers are divided by $T$.

Our channel disassembly method allows us to retain outlier information with an equivalent output and ease the quantization difficulty with a smaller value range. Its only drawback is the increase in the number of channels, which may lead to additional computational costs and will be addressed in the next subsection.

**Channel Assembly.** Note that the input channel count increases to $M + T - 1$ after channel disassembly. Given the substantial quantity of channels in LLMs, it is possible to omit some unimportant channels or merge similar input channels to keep the original channel count $M$ for efficiency while maintaining outputs. To achieve this, a straightforward method is to use channel pruning [116, 117] that removes the unimportant channels directly. However, such method may result in substantial information loss, especially when $T$ is large. Motivated by recent studies [249, 250] that combine similar tokens, we propose a channel assembly method that delves into merging $T - 1$ similar input channels. Given channels $i$ and $j$, in alignment with token merging [249, 250], our goal is to aggregate them by calculating the average of their input features, denoted as $\frac{\mathbf{x}_i + \mathbf{x}_j}{2}$ and utilizing the aggregated feature in subsequent computations, which is defined as:

$$\mathbf{x}_i \mathbf{W}_{ik} + \mathbf{x}_j \mathbf{W}_{jk} \approx \frac{\mathbf{x}_i + \mathbf{x}_j}{2} \left( \mathbf{W}_{ik} + \mathbf{W}_{jk} \right), \tag{5.4}$$

where $\mathbf{W}_{ik} + \mathbf{W}_{jk}$ represents the merged weight. With the aim of minimizing the information loss of channel assembly in Eq. (5.4), we can define a distance metric $D(i, j)$ between channels $i$ and $j$ as

$$D(i, j) = \left\| \frac{\mathbf{x}_i (\mathbf{W}_{ik} - \mathbf{W}_{jk})}{2} + \frac{\mathbf{x}_j (\mathbf{W}_{jk} - \mathbf{W}_{ik})}{2} \right\|_2^2, \tag{5.5}$$

where $\|\cdot\|_2$ represents the $\ell_2$ norm. The above distance metric takes into account the difference in both input activations and weights between the two channels.

With the channel distance defined, the next step is to determine which channels to aggregate efficiently, with the goal of reducing the total channel count by $T - 1$. To

address this, we propose using bipartite soft matching [249, 250] that first partitions the channels into two sets, each containing roughly equal sizes, and subsequently finds the $T-1$ most similar pairs between these two sets (see Section 5.7.1 of the appendix for details). Note that we do not assemble the channels that are disassembled from the outlier channels since they play a critical role in the performance of LLMs. After the channel reassembly, including both disassembly and assembly, we acquire the reassembled input activations that are more amenable to quantization, along with the corresponding reassembled weights for layer $l$.

**Adaptive Reassembly.** In this section, we present a method to adaptively determine the appropriate reassembly ratio for each layer. For channel disassembly, selecting a high value for $T$ with a small $\theta$ substantially reduces outlier magnitudes and benefits quantization, while resulting in a larger increase in channel merging error due to a higher merging ratio. Conversely, choosing a small $T$ with a large $\theta$ will not increase the channel count much, making it easier for the assembly stage to keep the information while likely still retaining outliers, causing significant quantization errors. Therefore, it is crucial to carefully determine the outlier threshold $\theta$ or the reassembly channel number $T$.

However, it is hard to choose $\theta$ in practice as distinct layers have different patterns of outliers, as shown in Figure 5.5. Motivated by [180], we propose an adaptive strategy to find the optimal $\theta$ by minimizing the reassembly error between the original output activations and their counterparts generated with the reassembled input activations for each layer.

Note that our channel reassembly technique can yield the reassembled activation $\hat{\mathbf{X}} \in \mathbb{R}^{L\times M}$ with a sequence length of $L$, which can then be fed into a MSA layer or a FFN layer. For example, let us consider a case where $\hat{\mathbf{X}}$ is fed into a MSA layer. A standard MSA layer calculates queries, keys and values with three learnable projection matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{M\times N}$ as $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \mathbf{K} = \mathbf{X}\mathbf{W}_K, \mathbf{V} = \mathbf{X}\mathbf{W}_V$, where $\mathbf{X} \in \mathbb{R}^{L\times M}$ represents the original input activation. Let $\hat{\mathbf{W}}_Q$, $\hat{\mathbf{W}}_K$, $\hat{\mathbf{W}}_V$ be the reassembled projection weights. In this way, the reconstructed queries, keys, and values can be formulated as $\tilde{\mathbf{Q}} = \text{quant}(\hat{\mathbf{X}})\text{quant}(\hat{\mathbf{W}}_Q), \tilde{\mathbf{K}} = \text{quant}(\hat{\mathbf{X}})\text{quant}(\hat{\mathbf{W}}_K), \tilde{\mathbf{V}} = \text{quant}(\hat{\mathbf{X}})\text{quant}(\hat{\mathbf{W}}_V)$. We then find $\theta$ by solving the problem as

$$\arg\min_\theta \left\| \text{Softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V} - \text{Softmax}(\tilde{\mathbf{Q}}\tilde{\mathbf{K}}^\top)\hat{\mathbf{V}} \right\|_F^2, \tag{5.6}$$

where $\|\cdot\|_F$ denotes the Frobenius norm. To solve problem (5.6) efficiently, we use grid search following [167, 180] (see Algorithm 2).

**Algorithm 2:** Algorithm of Adaptive Channel Reassembly for one layer in LLM.

**Input:** Input activation $\mathbf{x} \in \mathbb{R}^M$, linear layer weight $\mathbf{W} \in \mathbb{R}^{M \times N}$, grid search iteration $P$.

Set $\mathcal{L}^*$ to $\infty$.
**Function** `ReAssembly(`$\theta$`):`
    // Channel disassembly
    Calculate the total sub-channels number by $n = \sum_{i=1}^{M} \lceil \max(|\mathbf{x}_i|)/\theta \rceil$.
    Perform channel disassembly using Eq. (5.3).
    // Channel assembly
    Find the $n$ most similar channel pairs using bipartite soft matching with the distance metric in Eq. (5.5).
    Perform channel assembly using Eq. (5.4).
    **return** ReAssembly Error $\mathcal{L}$ using Eq. (5.6);
Calculate the max value of each channel $\mathbf{m} \in \mathbb{R}^M$.
**for** $p \in \{1, 2, \ldots, P\}$ **do**
    Calculate the threshold by $\theta = \min(\mathbf{m}) + \frac{p}{P} \cdot (\max(\mathbf{m}) - \min(\mathbf{m}))$.
    $\mathcal{L}$=`ReAssembly(`$\theta$`)`;
    **if** $\mathcal{L} < \mathcal{L}^*$ **then**
        $\mathcal{L}^* \leftarrow \mathcal{L}$, $\theta^* \leftarrow \theta$.
Adopt the final channel reassembly using the found $\theta^*$: `ReAssembly(`$\theta^*$`)`.
**return** One reassembled layer of LLM.

### 5.4.2 Efficient Gradient-based Error Correction

Based on the above gradient-free adaptive channel reassembly, an efficient gradient-based error correction technique is further proposed for improving the performance of the quantized LLMs using a small set of calibration data.

Inspired by recent developments in parameter-efficient fine-tuning methods [175, 247], the efficient error correction introduces two low-rank parameters $\mathbf{A} \in \mathbb{R}^{M \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times N}$ with a rank of $r$ into each projection layer of our QLLM. Then, we can obtain the output $\mathbf{Y}$ of a quantized linear layer by $\mathbf{Y} = \text{quant}(\mathbf{X})\text{quant}(\mathbf{W}) + \text{quant}(\mathbf{X})\mathbf{AB}$. Instead of directly tuning the quantized weights, we learn the introduced low-rank parameters by minimizing the reconstruction error between the original and the quantized outputs of the Attention-FFN block. Thanks to the reduced number of trainable parameters, both the optimization cost and GPU memory usage can be significantly reduced,

Such efficiency gain allows us to further suppress the accumulation of quantization error during forward propagation via a structured reconstruction, *i.e.*, performing multi-block reconstruction for QLLM, which simultaneously adjusts a collection of consecutive Attention-FFN blocks by focusing on reconstructing the final block output.

After the reconstruction, we only need to store the quantized weight quant($\mathbf{W} + \mathbf{AB}$), which does not introduce extra inference costs. Note that it is inevitable that the absorption process will introduce additional quantization errors. To counteract this, following [35, 161, 251], we perform reconstruction sequentially rather than in parallel, enabling us to account for the quantization error stemming from the previous layers.

### 5.4.3 Efficiency Discussion

**Reassembly efficiency.** Our adaptive channel reassembly stands out for its efficiency, mainly attributed to its gradient-free nature, which excludes the need for backward propagation. The main source of computational expense of our method comes from the channel assembly, which requires the calculation of pairwise distances. Fortunately, the utilization of efficient bipartite soft matching eliminates the need to compute distances for every pair of channels, enhancing the efficiency. For the gradient-based error correction, the reduced number of parameters significantly lowers its optimization cost, rendering it more efficient than directly adjusting the quantized weights.

**Inference efficiency.** The inference overhead of channel disassembly and assembly is small for two reasons. 1) recent studies [2, 180] have revealed that activation outliers are often concentrated in specific channels across various inputs. This property is also reflected in similar channels for assembly as well. Therefore, we are able to pre-calculate the channel indices for disassembly and assembly using a small number of calibration data, significantly reducing runtime overhead. 2) Both channel disassembly and assembly can be implemented efficiently if the previous layer $l-1$ is a linear layer. Please refer to Section 5.7.2 for more details. In cases where the preceding layer $l-1$ is a non-linear layer, such as a layer normalization [44], we introduce additional disassembly and assembly layers that are designed to decompose and aggregate channels during runtime, with the channel indexes for decomposition and aggregation calculated offline using calibration data. The pseudo codes of channel disassembly and assembly during

runtime can be found at Section 5.7.3 of appendix. Moreover, benefiting from our efficient kernel implemented by Triton [252] and limited reassembly ratio searched by our adaptive strategy (See Figure 5.4), the introduced inference cost is controlled within a small level.

## 5.5 Experiments

**Models and datasets.** We apply QLLM to quantize the LLaMA-1 [39] and LLaMA-2 [81] families. To evaluate the performance of the quantized LLM, we report the zero-shot accuracy on various benchmarks, including PIQA [253], ARC [254], HellaSwag [255], and WinoGrande [256]. Additionally, we evaluate the perplexity, a key indicator of a model's generative performance that correlates with zero-shot outcomes, on WikiText2 [11], PTB [12] and C4 [13].

**Quantization settings.** In alignment with prior research [43, 164], we use per-channel weight quantization and per-token activation quantization. Following [34, 164], we quantize all weights and intermediate activations, with the exception of the Softmax output probability, which is maintained at full precision. Following OmniQuant [164], we focus on 4- and 6-bit weights and activations quantization. Additionally, we also explore 4-bit weights and 8-bit activations quantization, aiming for hardware-friendly configurations while maintaining high performance. We exclude 8-bit quantization as SmoothQuant [2] is able to achieve lossless performance.

**Compared methods.** We compare our QLLM with several state-of-the-art PTQ quantization methods, such as OmniQuant [164], SmoothQuant (SQ) [2], Outlier Suppression+ (OS+) [180] and recent QAT method LLM-QAT [34]. For fair comparisons, we reproduce SmoothQuant and Outlier Suppression+ with per-channel weight quantization and per-token activation quantization.

**Implementation details.** Following OmniQuant [164], we construct the calibration set with 128 randomly sampled sequences from WikiText2, each with a sequence length of 2048. QLLM begins by applying channel reassembly prior to all linear projection layers, excluding the attention output projection layer, followed by performing error correction on the resulting model. The rank $r$ of the introduced low-rank parameters is set to 4, and these parameters are trained for 10 epochs with a mini-batch size of

TABLE 5.1: Performance comparisons of different methods for weights and activations quantization on LLaMA-1 model family. PPL denotes the perplexity.

| Model | #Bits | Method | PPL ↓ | | | Accuracy (%) ↑ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | WikiText2 | C4 | Avg. | PIQA | ARC-e | ARC-c | HellaSwag | Winogrande | Avg. |
| LLaMA-1-7B | W16A16 | - | 5.68 | 7.08 | 6.38 | 77.37 | 52.48 | 41.38 | 72.99 | 66.93 | 62.23 |
| | W6A6 | SQ | 6.15 | 7.61 | 6.88 | 76.65 | 53.11 | 40.10 | 71.52 | 61.88 | 60.65 |
| | W6A6 | OS+ | 5.90 | - | - | 76.82 | 51.35 | 41.13 | 71.42 | 65.98 | 61.34 |
| | W6A6 | OmniQuant | 5.96 | 7.43 | 6.70 | 77.09 | 51.89 | 40.87 | 71.61 | 65.03 | 61.30 |
| | W6A6 | QLLM | 5.89 | 7.34 | **6.62** | 77.26 | 52.02 | 41.04 | 71.40 | 65.19 | **61.38** |
| | W4A8 | QLLM | 5.96 | 7.49 | 6.73 | 76.17 | 50.84 | 40.02 | 70.75 | 66.22 | 60.80 |
| | W4A4 | SQ | 52.85 | 104.35 | 78.60 | 49.80 | 30.40 | 25.80 | 27.40 | 48.00 | 36.28 |
| | W4A4 | LLM-QAT | - | - | - | 51.50 | 27.90 | 23.90 | 31.10 | 51.90 | 37.26 |
| | W4A4 | LLM-QAT+SQ | - | - | - | 55.90 | 35.50 | 26.40 | 47.80 | 50.60 | 43.24 |
| | W4A4 | OS+ | 40.32 | - | - | 62.73 | 39.98 | 30.29 | 44.39 | 52.96 | 46.07 |
| | W4A4 | OmniQuant | 11.26 | 14.51 | 12.89 | 66.15 | 45.20 | 31.14 | 56.44 | 53.43 | 50.47 |
| | W4A4 | QLLM | 9.65 | 12.29 | **10.97** | 68.77 | 45.20 | 31.14 | 57.43 | 56.67 | **51.84** |
| LLaMA-1-13B | W16A16 | - | 5.09 | 6.61 | 5.85 | 79.05 | 59.84 | 44.62 | 76.22 | 70.09 | 65.96 |
| | W6A6 | SQ | 5.50 | 7.03 | 6.27 | 77.80 | 56.36 | 42.58 | 75.11 | 68.11 | 63.99 |
| | W6A6 | OS+ | 5.37 | - | - | 78.29 | 56.90 | 43.09 | 75.09 | 69.22 | 64.52 |
| | W6A6 | OmniQuant | 5.28 | 6.84 | 6.06 | 78.40 | 57.28 | 42.91 | 75.82 | 68.27 | **64.54** |
| | W6A6 | QLLM | 5.28 | 6.82 | **6.05** | 77.91 | 57.70 | 42.92 | 75.02 | 69.14 | **64.54** |
| | W4A8 | QLLM | 5.33 | 6.91 | 6.12 | 78.29 | 57.03 | 42.75 | 74.46 | 68.35 | 64.18 |
| | W4A4 | SQ | 79.35 | 120.24 | 99.80 | 55.55 | 34.51 | 26.71 | 41.56 | 48.70 | 41.41 |
| | W4A4 | OS+ | 53.64 | - | - | 63.00 | 40.32 | 30.38 | 53.61 | 51.54 | 47.77 |
| | W4A4 | OmniQuant | 10.87 | 13.78 | 12.33 | 69.69 | 47.39 | 33.10 | 58.96 | 55.80 | 52.99 |
| | W4A4 | QLLM | 8.41 | 10.58 | **9.50** | 71.38 | 47.60 | 34.30 | 63.70 | 59.43 | **55.28** |
| LLaMA-1-30B | W16A16 | - | 4.10 | 5.98 | 5.04 | 80.09 | 58.92 | 45.39 | 79.21 | 72.77 | 67.28 |
| | W6A6 | SQ | 5.37 | - | - | 77.14 | 57.61 | 42.91 | 78.07 | 69.92 | 65.13 |
| | W6A6 | OS+ | 4.48 | - | - | 80.14 | 58.92 | 45.05 | 77.96 | 71.98 | 66.81 |
| | W6A6 | OmniQuant | 4.38 | 6.22 | 5.30 | 79.81 | 58.79 | 45.22 | 78.95 | 72.21 | **67.00** |
| | W6A6 | QLLM | 4.30 | 6.17 | **5.24** | 79.65 | 58.08 | 44.11 | 78.38 | 73.24 | 66.69 |
| | W4A8 | QLLM | 4.40 | 6.22 | 5.31 | 79.11 | 57.87 | 44.62 | 78.03 | 72.22 | 66.37 |
| | W4A4 | SQ | 399.65 | 245.87 | 322.76 | 50.16 | 28.11 | 26.71 | 31.97 | 51.14 | 37.62 |
| | W4A4 | OS+ | 112.33 | - | - | 67.63 | 46.17 | 34.30 | 54.32 | 52.64 | 51.01 |
| | W4A4 | OmniQuant | 10.33 | 12.49 | 11.41 | 71.21 | 49.45 | 34.47 | 64.65 | 59.19 | 55.79 |
| | W4A4 | QLLM | 8.37 | 11.51 | **9.94** | 73.83 | 50.67 | 38.40 | 67.91 | 58.56 | **57.87** |
| LLaMA-1-65B | W16A16 | - | 3.56 | 5.62 | 4.59 | 80.85 | 58.75 | 46.25 | 80.73 | 77.11 | 68.74 |
| | W6A6 | SQ | 4.00 | 6.08 | 5.04 | 77.97 | 54.67 | 44.62 | 77.51 | 72.61 | 65.48 |
| | W6A6 | OS+ | - | - | - | 79.67 | 55.68 | 45.22 | 78.03 | 73.95 | 66.51 |
| | W6A6 | OmniQuant | 3.75 | 5.82 | 4.79 | 81.01 | 58.12 | 46.33 | 79.91 | 75.69 | **68.21** |
| | W6A6 | QLLM | 3.73 | 5.80 | **4.77** | 80.14 | 57.79 | 45.05 | 79.74 | 74.59 | 67.46 |
| | W4A8 | QLLM | 3.78 | 8.82 | 6.30 | 80.14 | 58.59 | 46.42 | 79.71 | 74.66 | 67.90 |
| | W4A4 | SQ | 112.02 | 118.96 | 115.49 | 61.81 | 40.15 | 32.08 | 46.19 | 50.83 | 46.21 |
| | W4A4 | OS+ | 32.60 | - | - | 68.06 | 43.98 | 35.32 | 50.73 | 54.30 | 50.48 |
| | W4A4 | OmniQuant | 9.17 | 11.28 | 10.23 | 71.81 | 48.02 | 35.92 | 66.81 | 59.51 | 56.41 |
| | W4A4 | QLLM | 6.87 | 8.98 | **7.93** | 73.56 | 52.06 | 39.68 | 70.94 | 62.9 | **59.83** |

1. We carry out the reconstruction using 4 Attention-FFN blocks. AdamW [206] with a linear learning rate decay scheduler is used following [181]. The learning rate is set to $5 \times 10^{-4}$ in most experiments; for LLaMA-2-70B, it is set to $1 \times 10^{-4}$. All training experiments are conducted on a single NVIDIA A100 80G GPU. We use the Language Model Evaluation Harness toolbox [257] for evaluation.

### 5.5.1 Main Results

We report the results on LLaMA-1 family in Table 5.1. Note that W6A6 has limited hardware support in real-world applications. However, our QLLM still demonstrates performance benefits in these settings, consistently surpassing OmniQuant in terms of lower perplexity across all models on both WikiText2 and C4 and achieving comparable accuracy on 5 zero-shot tasks. Remarkably, with W4A8 quantization, our method incurs only a minimal performance reduction. While the absolute performance gains with 6-bit quantization might seem modest, this is partly due to the less pronounced effect of activation outliers at this bitwidth. When focusing on extremely low-bitwidth quantization (*i.e.*, 4-bit), activation outliers serve as the performance bottleneck, thereby highlighting the importance of suppressing the outliers. In this case, QLLM achieves significantly higher zero-shot accuracy and much lower perplexity than the contenders. For example, QLLM quantized 4-bit LLaMA-1-65B outperforms OmniQuant counterpart by an average of 3.42% in accuracy across five zero-shot tasks. Remarkably, for LLaMA-7B, QLLM even surpasses the QAT method, LLM-QAT + SQ, by 8.6% on the average accuracy, which strongly demonstrates the efficacy of QLLM.

We provide additional results for the LLaMA-2 family in Table 5.2. The observations from these results are consistent with the phenomena identified in the LLaMA-1 family. Note that the varied performance of OmniQuant on W6A6 and W4A4 for LLaMA-2-70B can be attributed to the architecture of LLaMA-2-70B, which employs grouped-query attention (Ainslie et al., 2023) where each group of queries shares a single key and value head. Such architecture makes the learnable equivalent transformation in OmniQuant incompatible with grouped-query attention. In W6A6 settings, the impact of activation outliers is relatively minor, enabling partial learnable equivalent transformations to suffice in maintaining performance. However, in the W4A4 settings, the effect of activation outliers becomes more prominent. Under these conditions, the partial learnable equivalent transformation is insufficient to address the outlier issue, leading to notably poorer performance. Notably, our QLLM significantly outperforms the state-of-the-art post-training quantization (PTQ) methods, demonstrating a substantial margin of improvement in 4-bit quantization. For example, QLLM quantized 4-bit LLaMA-2-70B outperforms SmoothQuant counterpart by an average of 7.89% on the accuracy, which shows the promising results of our method.

TABLE 5.2: Performance comparisons of different methods for weights and activations quantization on LLaMA-2 model family.

| Model | #Bits | Method | PPL ↓ | | | Accuracy (%) ↑ | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | WikiText2 | C4 | Avg. | PIQA | ARC-e | ARC-c | HellaSwag | Winogrande | Avg. |
| LLaMA-2-7B | W16A16 | - | 5.47 | 6.97 | 6.22 | 76.82 | 53.62 | 40.53 | 72.87 | 67.25 | 62.22 |
| | W6A6 | SQ | 6.37 | 7.84 | 7.11 | 75.57 | 53.62 | 39.93 | 71.76 | 66.14 | 61.40 |
| | W6A6 | OS+ | - | - | - | 76.22 | 52.74 | 40.70 | 71.89 | 65.19 | 61.35 |
| | W6A6 | OmniQuant | 5.87 | 7.48 | 6.68 | 76.77 | 52.90 | 40.61 | 71.86 | 64.09 | 61.25 |
| | W6A6 | QLLM | 5.72 | 7.31 | **6.52** | 77.48 | 52.99 | 39.33 | 71.38 | 65.98 | **61.43** |
| | W4A8 | QLLM | 5.91 | 7.50 | 6.71 | 76.11 | 51.73 | 39.33 | 71.27 | 65.59 | 60.81 |
| | W4A4 | SQ | 101.77 | 93.21 | 97.49 | 60.17 | 35.23 | 27.13 | 37.08 | 49.57 | 41.84 |
| | W4A4 | OS+ | - | - | - | 63.11 | 39.10 | 28.84 | 47.31 | 51.3 | 45.93 |
| | W4A4 | OmniQuant | 14.61 | 18.39 | 16.50 | 65.94 | 43.94 | 30.80 | 53.53 | 55.09 | 49.86 |
| | W4A4 | QLLM | 11.75 | 13.26 | **12.51** | 67.68 | 44.40 | 30.89 | 58.45 | 56.59 | **51.60** |
| LLaMA-2-13B | W16A16 | - | 4.88 | 6.47 | 5.68 | 78.84 | 57.91 | 44.28 | 76.63 | 69.85 | 65.50 |
| | W6A6 | SQ | 5.19 | 6.77 | 5.98 | 78.29 | 57.41 | 43.86 | 75.02 | 66.93 | 64.30 |
| | W6A6 | OS+ | - | - | - | 78.29 | 59.13 | 43.34 | 75.37 | 67.56 | 64.74 |
| | W6A6 | OmniQuant | 5.14 | 6.74 | 5.94 | 78.56 | 57.11 | 43.60 | 75.36 | 68.35 | 64.60 |
| | W6A6 | QLLM | 5.08 | 6.71 | **5.90** | 78.78 | 58.29 | 43.77 | 75.10 | 68.43 | **64.87** |
| | W4A8 | QLLM | 5.17 | 6.78 | 5.98 | 78.67 | 57.11 | 41.89 | 75.33 | 68.75 | 64.35 |
| | W4A4 | SQ | 29.82 | 44.08 | 36.95 | 62.30 | 40.28 | 30.72 | 42.24 | 49.96 | 45.10 |
| | W4A4 | OS+ | - | - | - | 64.47 | 41.46 | 32.17 | 59.30 | 51.38 | 49.76 |
| | W4A4 | OmniQuant | 12.28 | 14.64 | 13.46 | 69.80 | 47.22 | 33.79 | 59.34 | 55.49 | 53.13 |
| | W4A4 | QLLM | 9.09 | 11.13 | **10.11** | 70.46 | 48.48 | 34.39 | 62.80 | 55.41 | **54.31** |
| LLaMA-2-70B | W16A16 | - | 3.32 | 5.52 | 4.42 | 81.01 | 59.68 | 47.95 | 80.87 | 76.95 | 69.29 |
| | W6A6 | SQ | 3.69 | 5.88 | 4.79 | 79.87 | 57.32 | 45.65 | 79.01 | 74.03 | 67.18 |
| | W6A6 | OS+ | - | - | - | 79.33 | 59.09 | 47.18 | 79.46 | 75.06 | 68.02 |
| | W6A6 | OmniQuant* | 3.71 | 5.91 | 4.81 | 80.20 | 60.27 | 46.84 | 80.55 | 76.01 | **68.77** |
| | W6A6 | QLLM | 3.55 | 5.76 | **4.66** | 80.63 | 59.01 | 45.99 | 79.64 | 75.37 | 68.13 |
| | W4A8 | QLLM | 3.60 | 5.76 | 4.68 | 80.79 | 58.59 | 47.44 | 79.42 | 75.77 | 68.40 |
| | W4A4 | SQ | 26.01 | 34.61 | 30.31 | 64.09 | 41.84 | 32.00 | 54.21 | 51.07 | 48.64 |
| | W4A4 | OS+ | - | - | - | 66.16 | 42.72 | 34.90 | 56.93 | 52.96 | 50.73 |
| | W4A4 | OmniQuant* | 41.10 | 54.33 | 47.72 | 52.99 | 31.14 | 23.89 | 33.88 | 52.01 | 38.78 |
| | W4A4 | QLLM | 7.00 | 8.89 | **7.95** | 74.27 | 50.59 | 37.2 | 71.62 | 59.43 | **58.62** |

* indicates no learnable equivalent transformation [164] on queries, keys, values, or attention output due to incompatibility with grouped-query attention [258] in LLaMA-2-70B model.

### 5.5.2 Results on Chat Models

To demonstrate the generalization ability of our QLLM on chat models, we apply QLLM to quantize LLaMA-2-7B-Chat and LLaMA-2-13B-Chat to 4-bit. These models are instruction-tuned and optimized for dialogue use cases. We include the concurent state-of-the-art quantizaiton method, OmniQuant, for comparisons. We use GPT-4 to assess the performance of the quantized models on a set of 80 sample questions in Vicuna benchmark [259]. To eliminate the potential position bias [260], we conducted the comparisons in both orders (a *vs.* b and b *vs.* a) for each pair, amounting to a total of 160 trials. From Table 5.3, our QLLM consistently achieves much better performance than OmniQuant.

TABLE 5.3: Performance comparisons between QLLM and OmniQuant for chat models.

| Model | Case | Former Win | Tie | Former Lost |
|---|---|---|---|---|
| LLaMA-2-7B-Chat | QLLM vs. OmniQuant | **137** | 19 | 4 |
| LLaMA-2-13B-Chat | QLLM vs. OmniQuant | **116** | 24 | 20 |

### 5.5.3 Ablation Studies

**Effect of different components in channel reassembly.** To show the effectiveness of the diverse components in channel reassembly, we apply different methods with our efficient error correction to yield 4-bit LLaMA-13B and show the results in Table 5.4. For channel disassembly, we determine $\theta$ by exploring different channel expansion ratios $\gamma$. We observe that our method with channel disassembly significantly surpasses the counterpart that does not utilize it. With the increasing $\gamma$, the performance of the quantized model can be further improved. These results strongly show that channel disassembly is able to make activations more quantization-friendly by decomposing the outlier channels.

TABLE 5.4: Perplexity results of different components in channel reassembly. "CD" stands for channel disassembly. "CA" represents channel assembly. "CP" indicates channel pruning. "Adaptive" refers to the adaptive strategy. "$\gamma$" is the channel expansion ratio.

| CD | CA | CP | Adaptive | $\gamma$ | LLaMA-1-13B | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | WikiText2 | PTB | C4 | Avg. |
| ✓ | | | | 0.00 | 189.35 | 539.59 | 303.45 | 344.13 |
| ✓ | | | | 0.01 | 8.31 | 14.44 | 10.74 | 11.16 |
| ✓ | | | | 0.03 | 8.01 | 13.52 | 10.27 | 10.60 |
| ✓ | | | | 0.05 | 7.85 | 13.38 | 10.13 | 10.45 |
| ✓ | | | | 0.07 | 7.81 | 13.35 | 10.11 | **10.42** |
| ✓ | ✓ | | | 0.01 | 8.68 | 15.16 | 11.12 | 11.65 |
| ✓ | ✓ | | | 0.03 | 8.72 | 14.99 | 11.03 | **11.58** |
| ✓ | ✓ | | | 0.05 | 8.95 | 15.34 | 11.29 | 11.86 |
| ✓ | ✓ | | | 0.07 | 9.39 | 15.98 | 11.84 | 12.40 |
| ✓ | | ✓ | | 0.01 | 8.98 | 16.34 | 11.37 | **12.23** |
| ✓ | | ✓ | | 0.03 | 9.51 | 18.29 | 12.7 | 13.50 |
| ✓ | | ✓ | | 0.05 | 9.60 | 18.11 | 13.4 | 13.70 |
| ✓ | | ✓ | | 0.07 | 11.23 | 21.61 | 19.79 | 17.54 |
| ✓ | ✓ | - | ✓ | - | 8.41 | 14.38 | 10.58 | **11.12** |

Furthermore, by incorporating channel assembly, our method manages to preserve the original channel count with little performance drop. In comparison to channel pruning, our channel assembly leads to lower information loss, thereby achieving much better performance, especially at higher expansion ratio $\gamma$. Rather than determining $\theta$ using a

predefined expansion ratio, our method, equipped with an adaptive strategy, is capable of autonomously finding optimal $\theta$, resulting in near-lossless performance compared to the approach utilizing only channel disassembly. The resulting expansion ratios for different layers are shown in Figure 5.4 of Section 5.7.15 in the appendix.

**Effect of efficient gradient-based error correction.** After channel reassembly, we implement our QLLM to produce 4-bit LLaMA-7B models with our efficient gradient-based error correction (EEC) and tuning quantized weights directly (TQW) outlined in Section 5.4.2 to further improve the performance of quantized LLMs and show the results in Table 5.5. Compared with TQW which tunes all quantized weights, EEC focuses on learning a small set of low-rank weights, which significantly reduces training costs and GPU memory usage while delivering comparable performance. Moreover, the reduced GPU memory demand allows EEC to quantize LLaMA-1-65B on a single 24GB consumer-grade GPU, such as the NVIDIA RTX 4090, a task that is not feasible with TQW. We put more results in Section 5.7.9 of the appendix.

TABLE 5.5: Comparisons between efficient error correction (EEC) and tuning quantized weights directly (TQW) for 4-bit LLaMA-1-65B. "OOM" indicates out of memory.

| #Attn-FFN Block | Method | WikiText2 | PTB | C4 | Avg. | Training Time (GPU Hours) | GPU Memory (GB) |
|---|---|---|---|---|---|---|---|
| 1 | TQW | 6.34 | 17.61 | 9.56 | 11.17 | 12.16 | 30.84 |
| 1 | EEC | 8.31 | 13.77 | 10.76 | 10.95 | **7.79** | **19.00** |
| 2 | TQW | 6.25 | 11.18 | 8.56 | 8.66 | 12.13 | 52.45 |
| 2 | EEC | 7.62 | 11.47 | 9.39 | 9.49 | **7.79** | **28.60** |
| 4 | TQW | - | - | - | - | - | OOM |
| 4 | EEC | 6.87 | 11.36 | 8.98 | 9.07 | **7.77** | **47.71** |

**Inference efficiency.** To assess the inference efficiency of our channel reassembly technique, we measure the inference speed of QLLM on NVIDIA RTX 3090 GPUs. We employ W4A4 kernels from QUIK [261] codebase. We also conduct a comparative analysis using weight quantization only, utilizing CUDA kernels from AutoGPTQ[1]. As shown in Table 5.6, our 4-bit QLLM only incurs 4% additional cost relative to W4A4 but achieves a notable 1.96× speedup over FP16. Notably, our channel reassembly strategy substantially mitigates losses attributed to quantizing outliers (see Table 5.9), with only a slight extra computational overhead. For the detailed inference cost of channel disassembly and assembly, please refer to Section 5.7.11 of the appendix.

[1]https://github.com/PanQiWei/AutoGPTQ

TABLE 5.6: Inference throughput comparisons using a 2048-token segment on RTX 3090 GPUs: 1x GPU for LLaMA-1-7B and 2x GPUs for LLaMA-1-13B.

| Model | Method | Throughput (tokens/s) |
|---|---|---|
| LLaMA-1-7B | FP16 | 3252 |
| | W8A8 | 5676 |
| | W4A16 | 5708 |
| | W4A4 | 6667 |
| | QLLM | 6385 |
| LLaMA-1-13B | FP16 | 1910 |
| | W8A8 | 3179 |
| | W4A16 | 2026 |
| | W4A4 | 3873 |
| | QLLM | 3730 |

## 5.6 Conclusion and Future Work

In this chapter, we have proposed an accurate and efficient post-training quantization approach for low-bit LLMs, dubbed QLLM. The core of our QLLM lies in a novel adaptive channel reassembly paradigm that effectively addresses activation outliers, a pivotal factor contributing to the performance bottleneck in quantizing LLMs. The key idea involves reallocating outlier magnitudes to other channels, accomplished through a process of channel disassembly followed by assembly. We have further proposed a quantization-aware, parameter-efficient fine-tuning strategy that leverages calibration data to compensate for the information loss resulting from quantization. Extensive experiments on LLaMA model series have demonstrated the promising performance and training efficiency of QLLM. In terms of limitations, our proposed channel reassembly involves introducing additional operations to decompose and aggregate channels during runtime, thereby incurring additional inference costs. A potential solution to improve inference efficiency is to explore kernel fusing [262], aiming to fuse disassembly, assembly and layer normalization into a single operator. Another way is to aggregate more similar or unimportant channels [117] than those disassembled to achieve higher speedup.

## 5.7 Appendix

### 5.7.1 More Details About Bipartite Soft Matching

As mentioned in Section 5.4.1, we use bipartite soft matching [249] to determine which channels to aggregate efficiently. Support that we want to aggregate $T - 1$ channels.

The step-by-step bipartite soft matching algorithm is shown as follows:

1. Divide the channels into two sets $\mathbb{A}$ and $\mathbb{B}$, each of approximately equal size.
2. For each channel in $\mathbb{A}$, construct an edge to its most similar counterpart in $\mathbb{B}$.
3. Select the $T-1$ most similar edges.
4. Aggregate the channels that remain connected, according to Eq. (5.4).
5. Concatenate the two sets to form the assembled channel set.

### 5.7.2 More Details About the Efficient Implementation for Channel Reassembly

As mentioned in Section 5.4.3, the channel disassembly and assembly can be implemented efficiently if the previous layer $l-1$ is a linear layer. Specifically, let $\mathbf{W}^{l-1} \in \mathbb{R}^{C\times M}$ be the weights of preceding linear layer, where $C$ and $M$ denotes the input and output channel number for layer $l-1$, respectively. For channel disassembly, we enlarge the output channels of the preceding layer weights by:

$$\mathbf{W}^{l-1}_{:i} = \begin{cases} \mathbf{W}^{l-1}_{:i} & \text{if } i \leq M-1 \\ \frac{\mathbf{W}^{l-1}_{:M}}{T}, & \text{otherwise,} \end{cases} \tag{5.7}$$

and adjust the input channels of the current layer's weight by:

$$\mathbf{W}^{l}_{i:} = \begin{cases} \mathbf{W}^{l}_{i:} & \text{if } i \leq M-1 \\ \mathbf{W}^{l}_{M:}, & \text{otherwise.} \end{cases} \tag{5.8}$$

Similarly, for channel assembly, suppose that we are aggregating channel $j$ to channel $i$. Then, channel assembly can be implemented by reducing the output weight channels of the preceding linear layer $l-1$ by:

$$\mathbf{W}^{l-1}_{:i} = \frac{\mathbf{W}^{l-1}_{:i} + \mathbf{W}^{l-1}_{:j}}{2}, \tag{5.9}$$

and adjusting the input channels of the current layer's weight $l$ by:

$$\mathbf{W}^{l}_{i:} = \mathbf{W}^{l}_{i:} + \mathbf{W}^{l}_{j:}. \tag{5.10}$$

### 5.7.3 Pseudo-codes of Channel Disassembly and Assembly

We show the PyTorch style pseudo-codes of channel disassembly and assembly during runtime in Figure 5.2.

```
def channel_disassembly(x, num_split):
"""
    x: input with shape of [batch, tokens, channels]
    num_split: the number of sub-channels for each channel with shape of [channels]
"""

    B, N, C = x.shape
    x = x.view(B * N, C)
    scaling = 1.0 / num_split   # compute the scaling factor of each channel
    x = x / scaling                    # scale each channel
    x = torch.repeat_interleave(x, num_split, dim=1) # perform channel decomposition
    C = x.shape[1]
    x = x.view(B, N, C)
    return x

def channel_assembly(x, src_idx, dst_idx):
"""
    x: input with shape of [batch, tokens, channels]
    src_idx: the channel index that will be merged in set A with shape of [#num_merged_channels]
    dst_idx: the channel index that will be merged in set B with shape of [#num_merged_channels]
"""
    B, N, C = x.shape
    ori_src_idx = torch.arange(0, C, 2, device=x.device)  # get index for set A
    ori_dst_idx = torch.arange(1, C, 2, device=x.device)  # get index for set B
    src, dst = x[..., ori_src_idx], x[..., ori_dst_idx]    # divide the channels into two sets A and B
    src_C = src.shape[-1]    # get the channel number in set A
    dst_C = dst.shape[-1]    # get the channel number in set B

    # A mask that indicates whether a channel is merged
    channel_mask = torch.ones(C, device=x.device, dtype=x.dtype)
    m_idx = ori_src_idx[src_idx]
    channel_mask[m_idx] = 0.0

    n, t1, c = src.shape
    sub_src = src.gather(dim=-1, index=src_idx.expand(n, t1, r))   # get channels that will be merged in
     set A
    dst = dst.scatter_reduce(-1, dst_idx.expand(n, t1, r), sub_src, reduce=mode)  # merge channels
    src = src.view(B, N, src_C, 1)
    dst = dst.view(B, N, dst_C, 1)

    # concat set A and set B
    if src_C == dst_C:
        merged_x = torch.cat([src, dst], dim=-1).view(B, N, C)
    else:
        merged_x = torch.cat([src[..., :-1, :], dst], dim=-1).view(
            B, N, src_C + dst_C - 1
        )
        merged_x = torch.cat([merged_x, src[..., -1, :].reshape(B, N, 1)], dim=-1).view(
            B, N, src_C + dst_C
        )
    # remove the merged channels
    merged_x = merged_x.index_select(-1, (channel_mask != 0).nonzero().squeeze())
    return merged_x
```

FIGURE 5.2: PyTorch style pseudo codes of channel disassembly and assembly during runtime.

### 5.7.4 More Results in Terms of Channel Reassembly

To further show the effectiveness of our channel reassembly (CR), we compare the average block-wise reconstruction error across the entire network before and after applying

CR and show the results on a calibration set with 128 randomly selected 2048-token segments from WikiText2 in Table 5.7. The results clearly demonstrate that using CR significantly lowers the reconstruction error, and thus improves the performance of the quantized models.

TABLE 5.7: Block-wise reconstruction error before and after channel reassembly (CR).

| Model | Method | Reconstruction Error |
|---|---|---|
| LLaMA-1-7B | w/o CR | 4.71 |
| LLaMA-1-7B | w/ CR | **2.74** |
| LLaMA-1-13B | w/o CR | 7.67 |
| LLaMA-1-13B | w/ CR | **1.71** |

### 5.7.5 More Results in Terms of Channel Disassembly Only

To further demonstrate the effectiveness of channel disassembly (CD), we apply CD without efficient error correction (EEC) to obtain 4-bit LLaMA-1-13B and show the results in Table 5.8. We observe that the absence of both CD and EEC leads to a significant decline in the performance of the quantized model. Notably, using CD alone substantially reduces the performance degradation associated with quantization. Moreover, increasing the channel expansion ratio $\gamma$ further improves the model's performance, which strongly shows the benefits of using CD to decompose the outlier channels. By incorporating both CD and EEC, the performance improvement is even more pronounced, underscoring the efficacy of EEC in conjunction with CD.

TABLE 5.8: Perplexity results of channel disassembly (CD) with and without efficient error correction (EEC). "$\gamma$" is the channel expansion ratio. We report the perplexity of W4A4 LLaMA-1-13B on WikiText2 [11], PTB [12] and C4 [13].

| CD | EEC | $\gamma$ | WikiText2 | PTB | C4 | Avg. |
|---|---|---|---|---|---|---|
| | | - | 1702.34 | 1853.58 | 1159.41 | 1571.78 |
| ✓ | | 0.01 | 19.34 | 45.36 | 23.25 | 29.32 |
| ✓ | ✓ | 0.01 | **8.31** | **14.44** | **10.74** | **11.16** |
| ✓ | | 0.03 | 12.11 | 24.73 | 14.38 | 17.07 |
| ✓ | ✓ | 0.03 | **8.01** | **13.52** | **10.27** | **10.60** |
| ✓ | | 0.05 | 11.4 | 23.53 | 13.62 | 16.18 |
| ✓ | ✓ | 0.05 | **7.85** | **13.38** | **10.13** | **10.45** |
| ✓ | | 0.07 | 11.13 | 23.47 | 13.45 | 16.02 |
| ✓ | ✓ | 0.07 | **7.81** | **13.35** | **10.11** | **10.42** |

### 5.7.6 More Comparisons With Other Outlier Handling Methods

To further show the effectiveness of channel reassembly, we compare our method with previous outlier handling methods which employ gradient-free methods to learn mathematically equivalent transformations. For fair comparisons, we do not apply efficient error correction. From Table 5.9, all methods exhibit comparable performance at 6-bit quantization. However, for 4-bit quantization, channel reassembly significantly surpasses other methods by a large margin, particularly for larger models.

TABLE 5.9: Performance comparisons of our channel reassembly (CR) with previous outlier handling methods across five zero-shot tasks.

| Model | #Bits | Method | PIQA | ARC-e | ARC-c | HellaSwag | Winogrande | Avg. |
|---|---|---|---|---|---|---|---|---|
| LLaMA-1-7B | W6A6 | SQ | 76.65 | 53.11 | 40.10 | 71.52 | 61.88 | 60.65 |
| | W6A6 | OS+ | 76.82 | 51.35 | 41.13 | 71.42 | 65.98 | **61.34** |
| | W6A6 | CR | 76.88 | 52.31 | 40.87 | 71.37 | 64.33 | 61.15 |
| | W4A4 | SQ | 49.80 | 30.40 | 25.80 | 27.40 | 48.00 | 36.28 |
| | W4A4 | OS+ | 62.73 | 39.98 | 30.29 | 44.39 | 52.96 | 46.07 |
| | W4A4 | CR | 66.92 | 42.55 | 32.34 | 54.31 | 50.04 | **49.23** |
| LLaMA-1-13B | W6A6 | SQ | 77.80 | 56.36 | 42.58 | 75.11 | 68.11 | 63.99 |
| | W6A6 | OS+ | 78.29 | 56.90 | 43.09 | 75.09 | 69.22 | **64.52** |
| | W6A6 | CR | 78.02 | 56.69 | 42.41 | 74.70 | 70.01 | 64.37 |
| | W4A4 | SQ | 55.55 | 34.51 | 26.71 | 41.56 | 48.70 | 41.41 |
| | W4A4 | OS+ | 63.00 | 40.32 | 30.38 | 53.61 | 51.54 | 47.77 |
| | W4A4 | CR | 67.57 | 43.77 | 31.48 | 60.78 | 56.04 | **51.93** |

### 5.7.7 More Results in Terms of Efficient Error Correction Only

Using EEC only without our channel reassembly results in suboptimal performance as it suffers from activation outlier issues. To demonstrate this, we applied EEC only to quantize LLaMA-1-7B to 4-bit, using the same training settings as our QLLM but with varying numbers of calibration samples. From Table 5.10, even with an increased amount of calibration data, the performance of the EEC only significantly lags behind our QLLM. These results strongly demonstrate the effectiveness of channel reassembly in addressing activation outliers, thereby substantially improving performance.

TABLE 5.10: Performance comparisons with different methods under various # calibration samples. We report the perplexity of W4A4 LLaMA-1-7B on WikiText2 [11], PTB [12] and C4 [13].

| Method | #Samples | WikiText2 | PTB | C4 | Avg. |
|---|---|---|---|---|---|
| EEC | 128 | 16.64 | 38.58 | 28.33 | 27.85 |
| EEC | 256 | 14.94 | 37.70 | 33.62 | 28.75 |
| EEC | 512 | 12.35 | 29.39 | 31.59 | 24.44 |
| QLLM | 128 | **9.65** | **16.56** | **12.29** | **12.83** |

### 5.7.8 More Results in Terms of Tuning Quantized Weights Only

The effectiveness of TQW is highly dependent on our channel reassembly. To show this, we applied TQW only to quantize LLaMA-1-7B to 4-bit using the same settings as QLLM and show the results in Table 5.11. The results clearly indicate that the absence of our adaptive channel reassembly results in significantly reduced performance for TQW. This underscores the vital role of channel reassembly in addressing activation outliers and thus improving model performance.

TABLE 5.11: Performance comparisons with different methods. We report the perplexity of 4-bit LLaMA-1-7B on WikiText2 [11], PTB [12] and C4 [13]. "CR" denotes our adaptive channel reassembly.

| Method | WikiText2 | PTB | C4 | Avg. |
|---|---|---|---|---|
| TQW w/o CR | 13.13 | 42.81 | 32.07 | 29.34 |
| TQW w/ CR | **8.90** | **14.75** | **11.63** | **11.76** |

### 5.7.9 More Comparisons Between Efficient Error Correction and Tuning Quantized Weights Directly

To further show the effectiveness of our efficient error correction (EEC), we conduct more comparisons between EEC and tuning quantized weights (TQW) directly on small model and report the results in Table 5.12. The results show that employing EEC not only maintains comparable performance but also markedly improves training speed and significantly reduces GPU memory usage over TQW. It is worth noting that there is a trade-off between GPU memory (*i.e.*, #Attn-FFN blocks) and performance. Leveraging EEC even allows us to perform reconstruction for 16 Attention-FFN blocks simultaneously, thereby significantly improving performance while preserving a similar training speed and a reasonable increase in GPU memory.

TABLE 5.12: Perplexity comparisons between efficient error correction (EEC) and tuning quantized weights directly (TQW) for 4-bit LLaMA-1-7B. "OOM" indicates out of memory.

| #Attn-FFN Block | Method | WikiText2 | PTB | C4 | Avg. | Training Time (GPU Hours) | GPU Memory (GB) |
|---|---|---|---|---|---|---|---|
| - | CR | 14.12 | 25.30 | 16.58 | 18.67 | - | - |
| 1 | TQW | 10.10 | 16.19 | 12.95 | 13.08 | 1.49 | 10.9 |
| | EEC | 11.21 | 19.29 | 14.06 | 14.85 | **1.06** | **7.95** |
| 2 | TQW | 9.74 | 14.79 | 11.68 | 12.07 | 1.49 | 17.62 |
| | EEC | 10.61 | 18.56 | 13.64 | 14.27 | **1.05** | **11.62** |
| 4 | TQW | 8.90 | 14.75 | 11.63 | 11.76 | 1.48 | 30.95 |
| | EEC | 9.65 | 16.56 | 12.29 | 12.83 | **1.05** | **18.91** |
| 8 | TQW | - | - | - | - | - | OOM |
| | EEC | 9.18 | 14.98 | 11.63 | 11.93 | **1.05** | **33.50** |
| 16 | TQW | - | - | - | - | - | OOM |
| | EEC | 9.13 | 14.95 | 11.60 | 11.89 | **1.05** | **62.70** |

### 5.7.10 Effect of Weight Merging in Efficient Error Correction

As explained in Section 5.4.2, for quant($\mathbf{X}$)quant($\mathbf{W}$) + quant($\mathbf{X}$)$\mathbf{AB}$, the low-rank weights $\mathbf{A}$ and $\mathbf{B}$ bring not only additional inference overhead due to the matrix multiplication between the full-precision $\mathbf{AB}$ and quant($\mathbf{X}$) but also extra storage burden. To address this, we perform weight merging by quant($\mathbf{W} + \mathbf{AB}$) after the reconstruction, which effectively avoids overhead but introduces additional quantization error. For 4-bit quantization, we empirically observe that merging the low-rank weights into the frozen weights using quant($\mathbf{W} + \mathbf{AB}$) does not lead to an increase in outliers. This finding is supported by the notably low MSE levels for channel-wise $P_{99}$, $P_{999}$, and maximum/minimum values before and after the weight merging process across in Table 5.13. Moreover, our weight merging only leads to small quantization error, as shown in Table 5.14. Note that even small deviations can aggregate throughout the network, leading to the performance drop. To address this, as shown in Section 5.4.2, we further employ sequential reconstruction to mitigate errors from previous layers, resulting in only a negligible performance drop. To demonstrate this, we compare the performance of QLLM with and without the weight merging. From Table 5.15, the weight merging only leads to a slight increase in perplexity.

TABLE 5.13: The maximum MSE of channel-wise $P_{99}$, $P_{999}$, and maximum/minimum values before and after the merging process across all layers of 4-bit LLaMA-1-7B.

| $\text{MSE}_{P_{99}}$ | $\text{MSE}_{P_{999}}$ | $\text{MSE}_{\text{max}}$ | $\text{MSE}_{\text{min}}$ |
|---|---|---|---|
| $5.42 \times 10^{-6}$ | $3.48 \times 10^{-6}$ | $7.11 \times 10^{-6}$ | $8.71 \times 10^{-6}$ |

TABLE 5.14: The maximum MSE of channel-wise $P_{99}$, $P_{999}$, and maximum/minimum values before and after the merging process across all layers of 4-bit LLaMA-1-7B.

| Model | MSE of Weight Merging |
|---|---|
| LLaMA-1-7B | $4.04 \times 10^{-7}$ |
| LLaMA-1-13B | $3.64 \times 10^{-7}$ |

TABLE 5.15: Effect of the weight merging (WM) in the efficient error correction. We report the perplexity on WikiText2 [11], PTB [12] and C4 [13].

| Model | Method | WikiText2 | PTB | C4 | Avg. |
|---|---|---|---|---|---|
| LLaMA-1-7B | w/o WM | 9.35 | 15.93 | 11.93 | 12.40 |
| LLaMA-1-7B | w/ WM | 9.65 | 16.56 | 12.29 | 12.83 |
| LLaMA-1-13B | w/o WM | 8.29 | 13.69 | 10.40 | 10.79 |
| LLaMA-1-13B | w/ WM | 8.41 | 14.38 | 10.58 | 11.12 |

### 5.7.11 More Results Regarding Inference Efficiency

Following [263, 264], we use Bit-Operation (BOP) count to measure the theoretical inference complexity of our QLLM. From Table 5.16, our 8-bit QLLM incurs only a marginal increase in BOPs when compared to the INT8 model but substantially lower than those of the FP16 counterpart, which shows the efficiency of our method.

TABLE 5.16: Bit-Operation (BOP) count comparisons of different models. We report the results of LLaMA-1-7B with a mini-batch size of 1. "$L$" denotes the sequence length.

| $L$ | 256 | 512 | 1024 | 2048 |
|---|---|---|---|---|
| FP16 | 875.52T | 1,766.40T | 3,604.48T | 7,493.12T |
| INT8 | 231.58T | 467.64T | 952.56T | 1976.16T |
| QLLM | 231.58T+1.07M | 467.64T+2.14M | 952.56T+4.28M | 1976.16T+8.56M |

We further show the inference time of channel disassembly and assembly of our QLLM in Table 5.17. From the results, channel disassembly results in additional inference costs due to the extra channels. These additional channels often don't align with GPU-friendly multiples like 32 or 64, leading to less efficient GPU use. Using our channel assembly maintains the original channel count, ensuring better GPU utilization and mitigating the extra inference costs from disassembly. As a result, the quantized models with both channel disassembly and assembly achieve higher throughput compared to the ones with disassembly only, which demonstrates the necessity of channel assembly.

TABLE 5.17: Inference throughput (tokens/s) comparisons of different models. The throughput is measured with a 2048-token segment on NVIDIA RTX 3090 GPUs: 1x GPU for LLaMA-1-7B and 2x GPUs for LLaMA-1-13B. "CD" stands for channel disassembly. "CA" represents channel assembly. "Adaptive" refers to the adaptive strategy. "$\gamma$" is the channel expansion ratio. "OOM" indicates out of memory.

| Model | Method | CD | CA | Adaptive | $\gamma$ | Inference Throughput (tokens/s) |
|---|---|---|---|---|---|---|
| LLaMA-1-7B | FP16 | | | | - | 3252 |
| | W8A8 | | | | - | 5676 |
| | W4A16 | | | | - | 5708 |
| | W4A4 | | | | - | 6667 |
| | W4A4 | ✓ | | | 0.01 | 6322 |
| | W4A4 | ✓ | | | 0.05 | 6315 |
| | W4A4 | ✓ | | | 0.1 | 6310 |
| | W4A4 | ✓ | ✓ | | 0.01 | 6365 |
| | W4A4 | ✓ | ✓ | | 0.05 | 6334 |
| | W4A4 | ✓ | ✓ | | 0.1 | 6318 |
| | W4A4 | ✓ | ✓ | ✓ | - | 6385 |
| LLaMA-1-13B | FP16 | | | | - | 1910 |
| | W8A8 | | | | - | 3179 |
| | W4A16 | | | | - | 2026 |
| | W4A4 | | | | - | 3873 |
| | W4A4 | ✓ | | | 0.01 | 3728 |
| | W4A4 | ✓ | | | 0.05 | 3725 |
| | W4A4 | ✓ | | | 0.1 | 3678 |
| | W4A4 | ✓ | ✓ | | 0.01 | 3731 |
| | W4A4 | ✓ | ✓ | | 0.05 | 3728 |
| | W4A4 | ✓ | ✓ | | 0.1 | 3681 |
| | W4A4 | ✓ | ✓ | ✓ | - | 3730 |

### 5.7.12 More Results Regarding Training Efficiency

We assess the training efficiency of our method in comparison to OmniQuant on a single NVIDIA A100 80G GPU. The GPU training hours for both methods are presented in Table 5.18. The results reveal that the training cost of our QLLM can be up to 1.93× faster than OmniQuant, showing the exceptional training efficiency of our QLLM.

TABLE 5.18: The training time (GPU Hours) comparisons of our QLLM with OmniQuant.

| Method | OmniQuant | QLLM |
|---|---|---|
| LLaMA-2-7B | 1.98 | **1.05** |
| LLaMA-2-13B | 3.46 | **1.79** |
| LLaMA-2-70B | 14.52 | **9.05** |

### 5.7.13 Effect of Different Calibration Sets

We apply QLLM to yield 4-bit LLaMA-7B using different calibration sets and report the results in Table 5.19. From the results, we observe that the choices of calibration set have a minor effect, as the performance remains relatively consistent across different sets. For fair comparisons, following OmniQuant [164], we use WikiText2 as a calibration set by default.

TABLE 5.19: Effect of different calibration sets. We report the perplexity ↓ of W4A4 LLaMA-7B on WikiText2 [11], PTB [12] and C4 [13].

| | | Evaluation Set | | | |
|---|---|---|---|---|---|
| | | WikiText2 | PTB | C4 | Average |
| Calibration Set | WikiText2 | 9.65 | 16.56 | 12.29 | 12.83 |
| | PTB | 10.73 | 13.89 | 12.44 | **12.35** |
| | C4 | 10.56 | 17.44 | 12.08 | 13.36 |

### 5.7.14 Effect of Different Numbers of Calibration Samples

We apply QLLM to yield 4-bit LLaMA-7B using different numbers of calibration samples from WikiText2 and show the results in Table 5.20. The results reveal a positive correlation between the performance of the quantized model and the number of calibration samples, indicating that utilizing more samples generally leads to better performance. This trend underscores the importance of the calibration phase, where leveraging a larger sample pool can provide a more comprehensive representation of the data distribution, enabling more accurate quantization. However, it is also imperative to consider the computational and memory overhead associated with an increasing number of calibration samples. There is an inherent trade-off between achieving higher model performance and maintaining computational efficiency.

TABLE 5.20: Effect of different # calibration samples. We report the perplexity ↓ of W4A4 LLaMA-7B on WikiText2 [11], PTB [12] and C4 [13].

| #Samples | WikiText2 | PTB | C4 | Average |
|---|---|---|---|---|
| 16 | 10.72 | 18.43 | 13.66 | 14.27 |
| 32 | 10.12 | 17.35 | 12.84 | 13.44 |
| 64 | 10.15 | 16.23 | 12.26 | 12.88 |
| 128 | 9.65 | 16.56 | 12.29 | 12.83 |
| 256 | 9.60 | 15.75 | 11.79 | **12.38** |

### 5.7.15 More Results About the Expansion Ratios of the Quantized LLMs

In this section, we illustrate the detailed expansion ratios for the input activations of different layers in the 4-bit LLaMA-1-7B and LLaMA-1-13B obtained by our adaptive strategy in Figures 5.3 and 5.4. From the results, our adaptive strategy allocates higher expansion ratios to the shallower MSA layers and to the deeper down projection layer in the FFN, which indicates that these layers possess a greater number of outliers. To substantiate this observation, we further plot the channel-wise maximum and minimum values for the input activations across different layers in Figure 5.5. These visual representations further underscore the effectiveness of our adaptive strategy in identifying and addressing the presence of outliers in different layers.

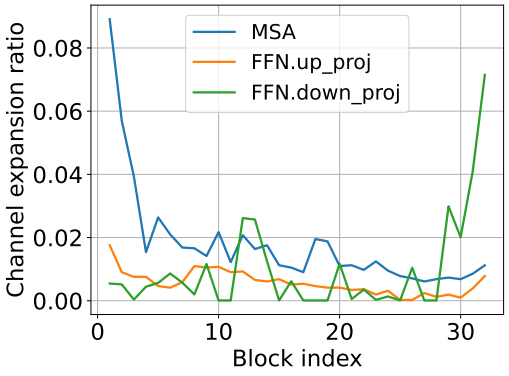



FIGURE 5.3: An illustration of the searched expansion ratios using our adaptive strategy for 4-bit LLaMA-1-7B.

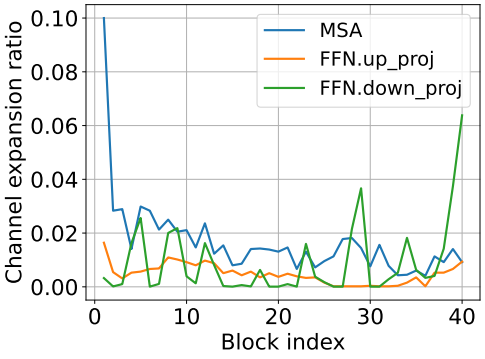



FIGURE 5.4: An illustration of the searched expansion ratios using our adaptive strategy for 4-bit LLaMA-1-13B.

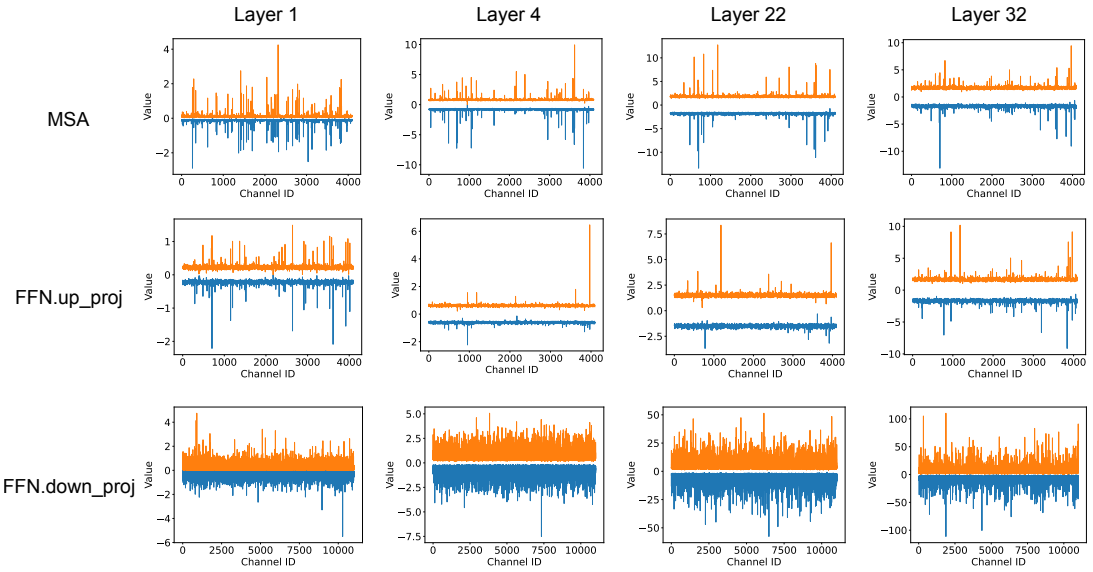



FIGURE 5.5: An illustration of the channel-wise maximum and minimum input activation values for the MSA, up projection and down projection layers in FFN of different blocks in LLaMA-1-13B.

# Chapter 6

# ME-Switch: A Memory-Efficient Expert Switching Framework for Large Language Models

## 6.1 Overview

The typical process for LLM's development involves pre-training a general foundation model on massive data, followed by fine-tuning on task-specific data to obtain a series of specialized experts. Serving these experts can pose significant memory challenges, as loading all experts onto devices is impractical, and frequent switching between experts in response to user requests can incur substantial I/O costs. Previous approaches decompose the expert weights as the pre-trained weights plus delta weights, followed by quantizing the delta weights using output channel-wise step sizes to reduce the model size. However, these methods overlook the fact that certain input channels of delta weights can cause significant quantization errors at extremely low bitwidths. Additionally, existing methods assume that the appropriate model for a user request is known in advance, which is not the case in practice.

In this chapter, we introduce ME-Switch, a memory-efficient expert switching framework tailored for serving multiple LLMs. To condense the number of bits required for describing the delta weights, we propose a salient-aware delta compression method that first identifies which input channels of delta weights are salient based on reconstruction error

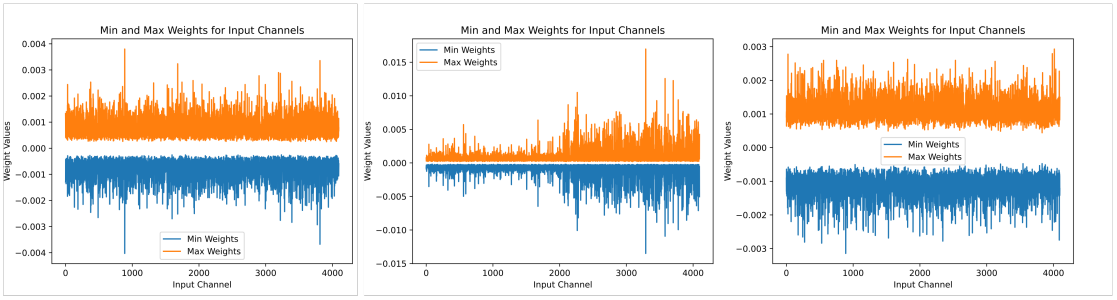



FIGURE 6.1: An illustration of the input channel-wise maximum and minimum values for the delta weights of Speechless-Code-Mistral-7B. The variability across input channels highlights that certain salient channels can cause significant quantization errors when quantized with ultra low-bitwidth, which underscores their critical role in preserving performance.

and then employs mixed-precision quantization that selectively quantizes non-salient input channels of delta weights to extremely low bits while keeping the salient ones intact, significantly reducing storage demand while maintaining performance. Moreover, we develop a model-level routing method that efficiently directs user queries to the most suitable expert by performing domain classification. Extensive experiments show the promising memory efficiency and routing performance of ME-Switch.

## 6.2 Introduction

LLMs such as GPT-4 [26] and Gemini [83], have achieved significant advancements in NLP. Trained on large-scale text datasets, these models develop a broad foundation of language understanding, enabling them to excel at tasks requiring common-sense knowledge. To acquire task-specific knowledge, LLMs can be further fine-tuned on specialized tasks, thereby adapting them for diverse applications such as interactive agents [81, 265], code generation [266, 267], and mathematical problem solving [267, 268], showing the remarkable versatility of LLMs. Nevertheless, given that no single model can master all tasks simultaneously, serving multiple LLMs, each precisely tailored for specific tasks, becomes essential for good performance.

However, serving multiple models poses several major challenges. First, even with a relatively small number of models, the storage demands are significant due to the extensive number of parameters each model contains. For example, three LLaMA-2-70B models would collectively require over 384GB of storage, calculated as 128GB per model

times three. Second, the substantial memory requirements of these models may make it impractical to load all of them into GPU memory simultaneously. While dynamically swapping model weights in and out of GPU memory as needed is feasible, the large size of the models makes this process slow and inefficient, significantly delaying response times and adversely affecting user experience.

To address the above challenges, existing methods [269, 270] decompose the weights of fine-tuned models into pre-trained weights and delta weights introduced during fine-tuning. While low-rank approximation [247] can compress these delta weights, it falls short for full fine-tuned models whose delta weights lack low-rank properties [269, 271, 272]. As a result, prior work has adopted per-tensor quantization [269] as an alternative, significantly reducing storage needs and facilitating efficient sharing of the base model's storage across multiple models. Nevertheless, this method ignores the distinction in delta weight values across different input and output channels, resulting in substantial information loss. To mitigate this issue, output channel-wise quantization [2, 135, 180] where each output channel is allocated its own learnable step size, still neglects input channel variations, as shown in Figure 6.1. To further mitigate information loss, rescaling input channels before quantization can be employed, but this provides only limited alleviation at extremely low bitwidths. In addition, existing methods assume that the appropriate model to utilize in response to a user request is known in advance. In reality, dynamically selecting the optimal model is essential for effectively handling diverse queries. This gap highlights the need for an intelligent model selection mechanism that can identify the most suitable model based on specific user needs.

In this chapter, we propose ME-Switch, a memory-efficient expert switching framework tailored for LLMs. To reduce the quantization error while simultaneously reducing storage needs, we develop a mixed-precision quantization method that quantizes non-salient input channels of delta weights to extremely low bits while preserving those salient ones, which could substantially increase quantization errors when quantized at very low bitwidths, in full precision. Since the number of salient input channels is relatively small, incorporating a limited amount of high-precision delta weights incurs negligible memory overhead and inference cost. To identify the important input channels of delta weights, one may select based on their magnitudes [43], as shown in Figure 6.2(a), which however fails to capture those leading to high quantization error. In contrast, our approach

identifies salient input channels of delta weights based on their impact on reconstruction errors in the output activations, as illustrated in Figure 6.2(b). Additionally, we develop a routing method that selects the most suitable LLM in response to a user request, enabling efficient and accurate model switching. To simplify the routing problem, we assume that each LLM specializes in a distinct domain, such as code generation or mathematics reasoning. This allows us to treat model selection as a multiple-choice question-answering task where each option represents a target domain, effectively transforming the problem into a domain classification task, as shown in Figure 6.3. Notably, dialogue LLMs exhibit strong instruction-following capabilities. Leveraging this, we propose using a small pre-trained dialogue LLM as a model-level router to solve the multiple-choice question-answering task. We then construct a multiple-choice question-answering dataset and fine-tune the router to improve its performance. Given its smaller size compared to the specialized LLMs, the router operates efficiently without adding significant overhead.

Our contributions can be summarized as follows.

- We introduce ME-Switch, a memory-efficient framework designed for serving multiple LLMs. It only stores a single full-precision pre-trained model and dynamically loads the appropriate compressed delta model weights in response to user queries.
- We develop a mixed-precision quantization method that significantly reduces the storage demands of serving multiple LLMs while maintaining performance, which is achieved by selectively quantizing the non-salient input channels of delta weights and leaving the salient ones unchanged. We further introduce a routing method that dynamically selects the most appropriate LLM for a given query, which is accomplished by transforming the model selection problem into a domain classification task and then solving it with a small LLM.
- We conduct extensive experiments demonstrating the promising memory efficiency and routing performance of ME-Switch. Remarkably, when serving three models from the Mistral-7B family, ME-Switch not only delivers near-lossless performance on instruction, mathematical reasoning, and code generation tasks but also reduces the model size by 1.74×. More impressively, our method is able to serves up to 16 Mistral-7B models on a single NVIDIA A100 GPU without running out of memory.

## 6.3 Preliminaries

For simplicity, we employ uniform quantization [32] to compress the models. Given a matrix $\mathbf{X}$ with floating-point values (*e.g.*, FP16), the quantization process can be expressed as:

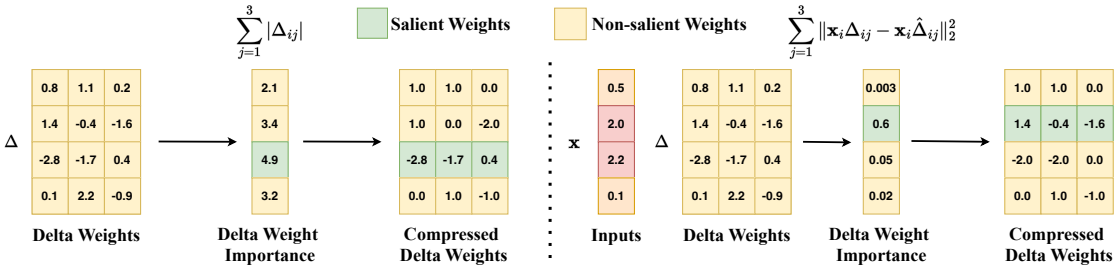
$$\hat{\mathbf{X}} = \text{quant}(\mathbf{X}) = \text{clamp}\left(\lfloor \frac{\mathbf{X}}{s} \rceil, -Q_N, Q_P\right) \times s, \tag{6.1}$$

where the function $\text{clamp}(\mathbf{V}, \mathbf{V}_{\min}, \mathbf{V}_{\max})$ clamps all elements in $\mathbf{V}$ within the range $[\mathbf{V}_{\min}, \mathbf{V}_{\max}]$, the operator $\lfloor \cdot \rceil$ rounds a given value to the nearest integer, and $s$ is a learnable quantization step size initialized by $\max(|\mathbf{X}|)/(2^{b-1}-1)$. Here, $Q_N$ and $Q_P$ denote the number of negative and positive quantization levels, respectively. For $b$-bit quantized weights, $Q_N$ and $Q_P$ are set to $2^{b-1}$ and $2^{b-1}-1$, respectively. Since the rounding function is not differentiable, we use the straight-through estimator (STE) [33] for gradient approximation following [33]. For binary quantization, we use the quantization method following [41].

Recent studies [269, 270] have shown that the weights of a fine-tuned model can be decomposed into the weights of the pre-trained model and the delta weights introduced during fine-tuning. Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{W}_{\text{FT}} \in \mathbb{R}^{m \times n}$ be the weight matrices of the pre-trained model and the fine-tuned model, respectively, where $m$ represents the number of input channels and $n$ denotes the output number of channels. The delta weight is defined as $\Delta = \mathbf{W}_{\text{FT}} - \mathbf{W}$. To reduce storage requirements, one can perform quantization using Eq. (6.1) to compress $\Delta$. However, this method often results in significant performance degradation because it assumes all delta weight channels are equally sensitive to quantization noise. Additionally, existing methods assume the appropriate model for a user request is predetermined. In practice, dynamically choosing the most suitable model based on the specific nature of each user's query is more effective.

## 6.4 Proposed Method

In this section, we propose ME-Switch, a memory-efficient expert switching framework designed to optimize the deployment of multiple LLMs. ME-Switch reduces memory demands while maintaining performance by using mixed-precision quantization, which quantizes non-salient weights to extremely low bitwidths while keeping salient weights




**(a) Magnitude-based selection** **(b) Our reconstruction-error-based selection**

FIGURE 6.2: An illustration comparison between the magnitude-based selection of salient delta weights and our reconstruction-error-based selection method. Given a delta weight matrix $\Delta \in \mathbb{R}^{m\times n}$, its quantized version $\hat{\Delta}$, and input $\mathbf{x} \in \mathbb{R}^m$, where $m$ and $n$ denote the number of input and output channels, respectively, our method measures the importance of each input delta channel by $\sum_{j=1}^{n}\|\mathbf{x}_i\Delta_{ij}-\mathbf{x}_i\hat{\Delta}_{ij}\|_2^2$.

intact. Additionally, ME-Switch introduces model-level routing to efficiently handle task-specific queries. We will detail ME-Switch in the following sections.

### 6.4.1 Salient-Aware Delta Compression

In this section, we introduce our salient-aware delta compression approach, which specifically targets the preservation of salient input channels during quantization. While quantization methods with learnable step sizes for each output channel can handle variations in output channels effectively [2, 135], variations in input channels pose a significant challenge to maintaining model performance, as shown in Figure 6.1. Some salient input channels of delta weights, regardless of their magnitude, can cause substantial information loss and degrade model performance when quantized to a very low bitwidth. To address this, rescaling the input channels of delta weights before quantization [171] offers a solution to mitigate some of the quantization error, which however provides only limited alleviation under the context of extremely low-bit quantization.

**Salient-aware delta selection.** To protect salient input channels of delta weights, we develop a mixed-precision quantization method, where the majority of input channels of delta weights are quantized to low-bit precision, while only a small number of critical input channels are represented in their original precision. This approach achieves significant reductions in storage requirements with minimal performance loss. The remaining challenge lies in identifying these salient input channels. A naive approach might involve selecting the input channels with large magnitudes as the salient delta weights [43], as shown in Figure 6.2(a). However, this method neglects the influence of input activations

on the outputs, failing to identify channels that lead to high quantization error. To address this, inspired by the pruning metric [117], we introduce a salient delta weights selection metric based on reconstruction errors in the outputs, considering both weights and input activations, as shown in Figure 6.2(b). Specifically, given an input $\mathbf{x} \in \mathbb{R}^m$, the reconstruction error for the input channel $i$ can be calculated by

$$\sum_{j=1}^{n} \|\mathbf{x}_i \Delta_{ij} - \mathbf{x}_i \hat{\Delta}_{ij}\|_2^2, \tag{6.2}$$

where $\hat{\Delta}$ is the quantized delta weights and $n$ denotes the output channel number. With the reconstruction error defined, we choose those input channels with the top-$k$ largest reconstruction errors as the salient ones and retain them in 16-bit representation to maintain performance.

**Efficient distillation.** The quantization step size in Eq. (6.1) plays an important role in the final performance. To learn the quantization step size, we employ knowledge distillation to guide the alignment of the output logits of the quantized model with those of the full-precision fine-tuned model following [269]. To reduce training overhead, we freeze the delta weights and focus solely on optimizing the quantization step size $\mathbf{s}$ using a small calibration dataset $\mathcal{X}$ by solving

$$\arg\min_{\mathbf{s}} \|f(\mathcal{X}) - \hat{f}(\mathcal{X}; \mathbf{s})\|_2^2, \tag{6.3}$$

where $f(\cdot)$ and $\hat{f}(\cdot)$ denote the output logits of the fine-tuned and quantized models, respectively. Thanks to the reduced number of trainable parameters, this training process is highly efficient.

**On-demand swapping.** The extremely low-bit compression for delta weights significantly reduces the model size, alleviating storage demands. This approach enables us to maintain a single pre-trained model while storing multiple sets of compressed delta weights, facilitating efficient on-demand swapping. In this scenario, the pre-trained model remains in GPU memory, and the corresponding compressed delta weights are loaded dynamically based on the user query. Compared with directly serving multiple models, our method is more GPU memory-efficient. Since LLM decoding is memory-bound [115, 273] due to the auto-regressive nature, reducing model size effectively decreases the parameter loading time, thereby improving decoding latency. To achieve fast

inference, we decouple the matrix multiplication during inference into two components:

$$\mathbf{y} = \mathbf{x}\mathbf{W}_{\text{FT}} = \mathbf{x}(\mathbf{W} + \Delta) \approx \mathbf{x}\mathbf{W} + \mathbf{x}\tilde{\Delta}, \tag{6.4}$$

where $\tilde{\Delta}$ is the compressed delta weights, including both the quantized unsalient and the full-precision salient delta weights. For $\mathbf{x}\mathbf{W}$, the computation is performed using a FP16 batched GEMM kernel. For $\mathbf{x}\tilde{\Delta}$, we implement an efficient Triton kernel [252] that fuses dequantization and matrix multiplication for efficient computation. The reduced memory footprint of the compressed delta weights enables our method to perform batched forward passes across multiple models simultaneously. This batching at the model level significantly enhances efficiency compared to the traditional approach, which processes each model individually—especially beneficial when serving multiple models (See Figure 6.8).

### 6.4.2 Model-level Routing

Given a user query, we present a method to determine the appropriate model. Consider a set of LLMs represented as $\mathcal{F} = \{f_1, f_2, \cdots, f_M\}$, where $M$ denotes the number of models. Given a user query $q$, we aim to find the most suitable LLM by solving the following problem $\arg\max_{f \in \mathcal{F}} P(q, f(q))$, where $P$ is a function that measures the quality or performance of the LLM response. To simplify the routing process, we assume that each LLM in $\mathcal{F}$ specializes in distinct domains such as code generation or mathematical problem solving. This setup allows us to treat the routing challenge as a multiple-choice question-answering task, where each option corresponds to a specific domain, thereby transforming the problem into a domain classification problem. Note that dialogue LLMs like Qwen1.5-1.8B-Chat [274] exhibit capabilities in following instructions, which inspires us to utilize a small pre-trained LLM as a model-level router. As illustrated in Figure 6.3, we first prompt the router with the user's question using a template designed to elicit domain classification. For the prompt template, please refer to Section 6.9.1 of the appendix. Based on the router's response, we then dynamically load the corresponding compressed delta weights for the selected domain-specific model, such as a mathematical model, to generate outputs.

Since the router is not explicitly trained for query domain classification, its initial routing performance may be suboptimal. To improve the routing performance, we construct a

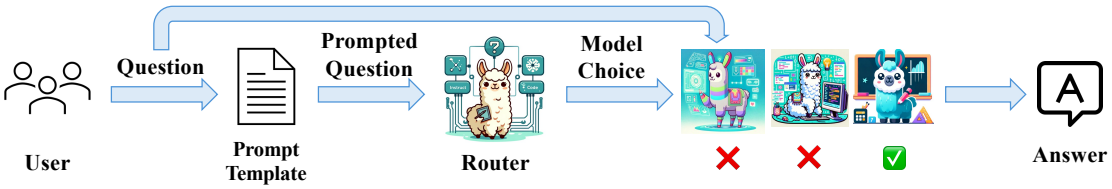



FIGURE 6.3: An illustration of the model-level routing. We first prompt the model-level router with the user query using a template (See Section 6.9.1) that presents a list of potential domains. The router then assesses these options and selects the most relevant domain by answering a multiple-choice question, effectively classifying the query into the corresponding category.

multiple-choice question-answering dataset tailored for our routing problem. We collect instruction-following data from various domains and insert the query into the prompt template as shown in Figure 6.3. The responses are constructed by considering the correct domain-specific model choice. We then fine-tune the router with our constructed dataset using supervised fine-tuning to further improve the routing accuracy.

### 6.4.3 Discussions

**Comparisons with routing in Mixture of Experts (MoE).** Besides model-level routing, another approach to handle diverse user queries efficiently is to construct a MoE using a set of pre-trained expert models. This can be achieved by integrating the feedforward layers from all pre-trained LLMs into a single MoE module at each attention-FFN block, and merging other layers, such as self-attention layers, by simply averaging their weights [275]. An additional gate network is introduced for each MoE module to perform token-level routing. However, this approach requires extensive fine-tuning of the entire network parameters and the gating network, as there is a significant gap between MoE experts and pre-trained LLMs. Notably, each expert in an MoE tends to become a generalist across all domains due to the load balancing loss, which encourages an even distribution of the workload among experts [73, 276]. This contrasts with pre-trained expert models, which are typically specialized for specific domains. Compared with MoE, our model-level routing has a significantly lower training cost, as we only need to train a router while keeping the expert models frozen.

**Model size reduction analysis.** Let $\Psi$ be the model size of an FP16 pre-trained model. When storing $M$ FP16 models, the total model size is $M\Psi$. In contrast, using our method, we store a single base model, a model-level router and $M$ compressed delta

models. The total model size is $\Psi + M\tilde{\Psi} + \Phi$, where $\tilde{\Psi}$ represents the size of the compressed delta model and $\Phi$ denotes the storage requirement for the router. Therefore, the compression ratio can be computed by $M\Psi/(\Psi + M\tilde{\Psi} + \Phi)$. For example, when serving 9 LLaMA-13B models, our method achieves a compression ratio of 3.63×. Empirical studies on compression ratios for varying model numbers are shown in Figures 6.6 and 6.7.

## 6.5 Experiments

**Candidate LLMs.** We apply our ME-Switch to two model families, Mistral-7B [265] and LLaMA-2-13B [81]. For the Mistral family, we include Dolphin-2.2.1-Mistral-7B [1] as the instruction expert, Speechless-Code-Mistral-7B [2] as the code expert, and MetaMath-Mistral-7B [3] as the math expert. For the LLaMA-2-13B family, LLaMA-2-13B-Chat [81] serves as the instruction expert, MetaMath-13B [4] as the math expert, and LLaMA2-Chinese-13B-Chat [5] as the Chinese expert. The above models are fine-tuned based on pre-trained backbones. We use Qwen1.5-1.8B-Chat [274] as the model-level router.

**Training and testing datasets.** We collect a diverse set of instruction samples from various open-source datasets, including Alpaca [277] for the instruction, MetaMathQA [278] for the mathematics, Code-74k-ShareGPT [6] for the code, and Chinese Alpaca [7] for the Chinese. To measure the performance of the resulting LLMs, we report accuracy on several benchmarks across different domains: MMLU [88] for the instruction, GSM8K [87] and MATH [279] for the mathematics, HumanEval [86] and MBPP [280] for the code, and C-Eval [281] and C-MMLU [282] for the Chinese. We use the WizardCoder toolbox to evaluate on HumanEval and MBPP, and the OpenCompass toolbox [283] to evaluate on other datasets. For MMLU, we report accuracy based on 5-shot in-context learning. To determine the answer for each question, we assess the perplexity of various response options and select the one with the lowest perplexity. For GSM8K and MATH, we adopt a 4-shot Chain of Thought (CoT) methodology to

[1] https://huggingface.co/cognitivecomputations/dolphin-2.2.1-mistral-7b
[2] https://huggingface.co/uukuguy/speechless-code-mistral-7b-v1.0
[3] https://huggingface.co/meta-math/MetaMath-Mistral-7B
[4] https://huggingface.co/meta-math/MetaMath-13B-V1.0
[5] https://huggingface.co/FlagAlpha/Llama2-Chinese-13b-Chat
[6] https://huggingface.co/datasets/ajibawa-2023/Code-74k-ShareGPT
[7] https://huggingface.co/datasets/hfl/alpaca_zh_51k

TABLE 6.1: Main results for Mistral-7B and LLaMA-13B families.

| Model | MMLU (%) ↑ | | | | | Mathematical Reasoning (%) ↑ | | | Code Generation (%) ↑ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | STEM | Hums. | Social | Other | Avg. | GSM8K | Math | Avg. | HumanEval | MBPP | Avg. |
| Dolphin-2.2.1-Mistral-7B | 52.05 | 68.83 | 73.42 | 65.43 | 63.43 | 63.68 | 12.80 | 38.24 | 42.70 | 54.90 | 48.80 |
| MetaMath-Mistral-7B | 50.45 | 66.82 | 71.63 | 64.60 | 61.87 | 73.92 | 20.62 | **47.27** | 0.00 | 21.60 | 10.80 |
| Speechless-Code-Mistral-7B | 51.82 | 68.35 | 73.74 | 65.69 | 63.36 | 61.18 | 13.52 | 37.35 | 51.20 | 60.40 | 55.80 |
| ME-Switch w/o Routing | 53.17 | 69.09 | 73.88 | 65.40 | **63.95** | 73.62 | 20.48 | 47.05 | 51.80 | 60.70 | **56.25** |
| ME-Switch | 51.49 | 68.37 | 73.60 | 66.08 | 63.32 | 73.39 | 20.30 | 46.85 | 51.80 | 60.70 | **56.25** |
| Model | MMLU (%) ↑ | | | | | Mathematical Reasoning (%) ↑ | | | Chinese (%) ↑ | | |
| | STEM | Hums. | Social | Other | Avg. | GSM8K | Math | Avg. | C-Eval | C-MMLU | Avg. |
| LLaMA-2-13B-Chat | 44.26 | 59.79 | 63.20 | 56.57 | 54.60 | 43.75 | 5.20 | 24.48 | 36.13 | 38.71 | 37.42 |
| MetaMath-13B | 37.81 | 52.77 | 56.00 | 50.05 | 47.84 | 69.14 | 8.48 | 38.81 | 33.62 | 32.70 | 33.16 |
| LLaMA2-Chinese-13B-Chat | 45.24 | 60.01 | 62.47 | 55.92 | 54.67 | 38.89 | 4.54 | 21.72 | 40.28 | 39.16 | 39.72 |
| ME-Switch w/o Routing | 44.57 | 60.87 | 64.00 | 58.04 | **55.45** | 70.05 | 13.20 | **41.63** | 40.13 | 39.91 | **40.02** |
| ME-Switch | 44.51 | 60.87 | 64.00 | 58.04 | 55.43 | 69.90 | 13.14 | 41.52 | 40.13 | 39.84 | 39.99 |

obtain the final answer following [284]. For the HumanEval and MBPP datasets, we employ a 0-shot configuration and generate answers using greedy decoding. We assess the functional correctness using the pass@1 metric following [285].

**Implementation details.** For salient-aware delta compression, we construct a calibration set from each domain-specific dataset and use these calibration sets to compress the delta weights for each respective domain. Each calibration set consists of 1600 randomly sampled sequences, each with a length of 128 tokens. The bitwidth $b$ and the number of FP16 input channels $k$ are set to 2 and 8, respectively. We use the AdamW optimizer [206] with a learning rate of $10^{-5}$ and a mini-batch size of 4 for training over 1 epoch. Delta weights compression experiments for the Mistral family are conducted on two NVIDIA A100 80G GPUs, while for the LLaMA-2-13B model, we use four NVIDIA A100 80G GPUs.

For the model-level router training, we construct the training data using samples from various domains, as mentioned in Section 6.4.2. To balance the dataset, we extract an equal number of samples from each domain-specific dataset. This constructed dataset is used to fine-tune model-level router through supervised fine-tuning for 4 epochs on a machine with 8 $\times$ A100 GPUs. We use the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, setting the learning rate to $3 \times 10^{-4}$ and applying a linear learning rate warmup. The weight decay is set to 0.01. We set the per-device mini-batch size to 8 and use gradient accumulation steps of 2.

## 6.6 Main Results

To evaluate the efficacy of our proposed model, we apply ME-Switch to the Mistral-7B and LLaMA-13B model families. We include ME-Switch without model-level routing for comparisons. The experimental results, detailed in Table 6.1, demonstrate that ME-Switch without routing, even with extremely compressed delta weights, achieves performance comparable to that of the respective unquantized expert models across various downstream tasks. For the Mistral-7B family, on MMLU, ME-Switch without routing lags behind the math expert by just 0.22% in mathematical reasoning tasks. Notably, ME-Switch without routing consistently outperforms the code expert in code generation tasks. The performance improvements over uncompressed expert models are primarily attributed to additional training through efficient distillation, which improves the models' task-specific performance by optimizing the quantization step size. With model-level routing, ME-Switch nearly matches the performance of the ones without it, demonstrating its precise routing for user queries.

We also applied our method to the LLaMA-3-8B family using the same experimental setup. We employ LLaMA-3-8B-Instruct as the instruction expert and LLaMA-8B-Chinese-Chat as the Chinese expert. The results, presented in Table 6.2, demonstrate that without model-level routing, our ME-Switch achieves nearly lossless performance compared to the unquantized expert models across various downstream tasks, underscoring the superior performance of our salient-aware delta compression. When model-level routing is integrated, performance remains nearly unchanged, highlighting its precise handling of user queries.

TABLE 6.2: Main results for LLaMA-3-8B family. "FP Baseline" refers to the performance metrics of experts without compression.

| Method | MMLU (%) ↑ | | | | | Chinese (%) ↑ | | |
|---|---|---|---|---|---|---|---|---|
| | STEM | Hums. | Social | Other | Avg. | C-Eval | C-MMLU | Avg. |
| FP Baseline | 57.30 | 71.64 | 77.83 | 71.13 | **68.05** | 51.99 | 52.25 | 52.12 |
| ME-Switch w/o Routing | 57.12 | 70.89 | 78.60 | 70.92 | 67.93 | 52.67 | 52.70 | **52.69** |
| ME-Switch | 57.16 | 70.44 | 78.53 | 71.25 | 67.90 | 52.67 | 52.70 | **52.69** |

## 6.7 Ablation Studies

**Fixed-precision quantization *vs.* Mixed-precision quantization.** To validate the effect of mixed-precision quantization, we compress the delta weights of MetaMath-Mistral-7B and Speechless-Code-Mistral-7B using both fixed-precision quantization and our salient-aware mixed-precision quantization. We evaluate their performance on mathematical reasoning and code generation tasks, respectively. The bitwidth $b$ and the number of FP16 input channels $k$ are set to default values mentioned in the implementation details. The results in Table 6.3 indicate that despite retaining a minimal number of FP16 channels, the model size of the mixed-precision model (2.13GB) is nearly identical compared to fixed-precision model (2.11GB). However, introducing a small number of FP channels significantly improves performance. For example, our compressed Speechless-Code-Mistral-7B, with a model size reduced by 6.33×, even outperforms the full-precision counterpart by 0.45% in the average accuracy on code generation. This underscores the capability of salient-aware mixed-precision quantization to minimize model size while preserving model performance.

TABLE 6.3: Performance comparisons between fixed-precision and mixed-precision quantization for MetaMath-Mistral-7B and Speechless-Code-Mistral-7B.

| Method | Model Size (GB) | Mathematical Reasoning (%) ↑ | | | Code Generation (%) ↑ | | |
|---|---|---|---|---|---|---|---|
| | | GSM8K | Math | Avg. | HumanEval | MBPP | Avg. |
| Full-precision | 13.48 | 73.92 | 20.62 | 47.27 | 51.20 | 60.40 | 55.80 |
| Fixed-precision | 2.11 | 73.31 | 20.44 | 46.88 | 47.00 | 59.10 | 53.05 |
| Mixed-precision | 2.13 | 73.62 | 20.48 | **47.05** | 51.80 | 60.70 | **56.25** |

**Low-rank adaptation *vs.* our salient-aware delta compression.** In addition to mixed-precision quantization, we can also employ low-rank adaptation (LoRA) to compress delta weights. Specifically, we decompose the delta weights as $\Delta = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$ and approximate delta weights using low-rank approximation $\tilde{\Delta} = \mathbf{A}\mathbf{B}$ where $\mathbf{A} = \tilde{\mathbf{U}}\sqrt{\tilde{\mathbf{\Sigma}}}$ and $\mathbf{B} = \sqrt{\tilde{\mathbf{\Sigma}}}\tilde{\mathbf{V}}$. Subsequently, $\mathbf{A}$ and $\mathbf{B}$ are refined using our efficient distillation mentioned in Section 6.4.1. To compare the effectiveness of LoRA against mixed-precision quantization, we apply both methods to the delta weights of Dolphin-2.2.1-Mistral-7B and MetaMath-Mistral-7B and evaluate performance on instructional and mathematical reasoning tasks. For LoRA, we set the rank to 512. The results are detailed in Table 6.4. Our approach with a much smaller model size outperforms LoRA, especially on Math. These results reveal that LoRA cannot accurately approximate delta weights for full

fine-tuned models like Dolphin-2.2.1-Mistral-7B and MetaMath-Mistral-7B, given that their delta weights lack low-rank properties, as also shown in Figure 6.9 in the appendix.

TABLE 6.4: Performance comparisons between LoRA and our salient-aware delta compression for Dolphin-2.2.1-Mistral-7B and MetaMath-Mistral-7B.

| Model | Model Size (GB) | MMLU (%) ↑ | | | | | Mathematical Reasoning (%) ↑ | | |
|---|---|---|---|---|---|---|---|---|---|
| | | STEM | Hums. | Social | Other | Avg. | GSM8K | Math | Avg. |
| LoRA | 2.99 | 52.16 | 68.38 | 73.15 | 65.52 | 63.33 | 73.62 | 17.30 | 45.46 |
| Ours | **2.11** | 53.17 | 69.09 | 73.88 | 65.40 | **63.95** | 73.62 | 20.48 | **47.05** |

**Effect of different number of FP16 channels.** To assess the impact of varying FP16 input channel counts $k$, we compress the delta weights of Speechless-Code-Mistral-7B and evaluate performance on code generation tasks. The bitwidth $b$ is set at 2. From Table 6.5, our method achieves lossless performance with $k = 8$. Increasing $k$ further yields no significant performance improvements, indicating that performance has plateaued. Therefore, we set $k$ to 8 by default.

TABLE 6.5: Effect of different FP16 input channel numbers $k$ for Speechless-Code-Mistral-7B.

| Model | Model Size (GB) | HumanEval | MBPP | Avg. |
|---|---|---|---|---|
| FP | 13.48 | 51.20 | 60.40 | 55.80 |
| $k = 8$ | 2.13 | 51.80 | 60.70 | 56.25 |
| $k = 16$ | 2.15 | 49.40 | 61.90 | 55.65 |
| $k = 32$ | 2.19 | 51.20 | 61.20 | 56.20 |
| $k = 64$ | 2.26 | 51.20 | 61.60 | **56.40** |

**Effect of different quantization bitwidths.** To investigate the impact of varying bitwidths $b$, we compress the delta weights of Speechless-Code-Mistral-7B and evaluate the performance on a code generation task. Table 6.6 shows that increasing $b$ from 1 to 2 significantly improve performance, achieving lossless results. Given that further increasing $b$ lead to negligible performance differences due to saturation, we set $b$ to 2 as the default.

TABLE 6.6: Effect of different bitwidths $b$ for Speechless-Code-Mistral-7B.

| Model | HumanEval | MBPP | Avg. |
|---|---|---|---|
| FP | 51.20 | 60.40 | 55.80 |
| $b = 1$ | 49.40 | 49.90 | 49.65 |
| $b = 2$ | 51.80 | 60.70 | **56.25** |
| $b = 4$ | 51.80 | 60.20 | 56.00 |

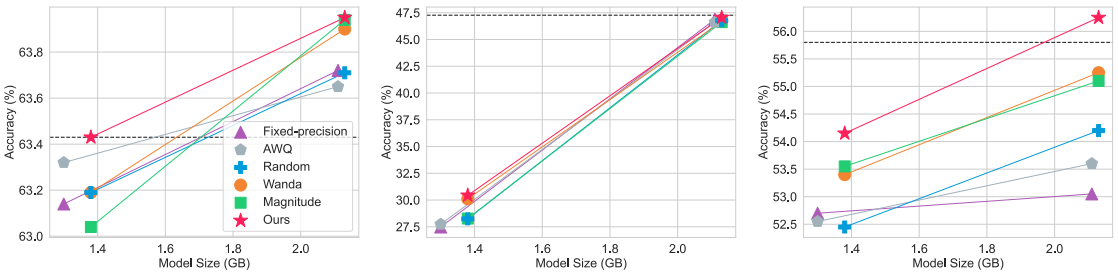



FIGURE 6.4: Average accuracy *vs.* delta weights size across different domains. "Baseline" refers to the fixed-precision quantization baseline. The dashed line indicates the full-precision counterpart.

**Performance comparisons with other weight-only quantization methods.** To demonstrate the promising performance of our salient-aware delta compression, we include the following weight-only quantization methods: **AWQ**: we use AWQ [171] to rescale input channels of delta weights before quantization to mitigate quantization errors. **Random**: using our salient-aware delta compression, we randomly select some input channels from the delta weights as important channels. **Wanda**: leveraging our salient-aware delta compression, we select important input channels of delta weights using the pruning metric from Wanda [117]. **Magnitude**: within our salient-aware delta compression framework, we select sensitive input channels of delta weights based on their weight magnitude, following the method proposed by [43]. We also include fixed-precision quantization for comparisons. We applied all methods to compress the delta weights of Dolphin-2.2.1-Mistral-7B, MetaMath-Mistral-7B, and Speechless-Code-Mistral-7B, using bitwidths $b = 1$ and $b = 2$. The results are shown in Figure 6.4. The detailed number of different methods can be found at Section 6.9.2 of the appendix. We observe that AWQ achieves comparable performance to the 1-bit baseline on code domain, highlighting the limitations of rescaling in extremely low-bitwidth quantization. In contrast, keeping the salient delta input channels performs favourably against the rescaling input channel counterpart. Moreover, our salient channel selection demonstrates superior performance than Random, Wanda, and Magnitude metrics across various bitwidths and tasks. For example, for $b = 2$, our salient-aware delta compression outperforms Wanda by 1.0% on the average accuracy on code domain, underscoring the effectiveness and superiority of our approach in selecting the salient delta weights.

**Effect of supervised fine-tuning in model-level routing.** To investigate the effect of supervised fine-tuning (SFT) on model-level routing, we evaluate the domain

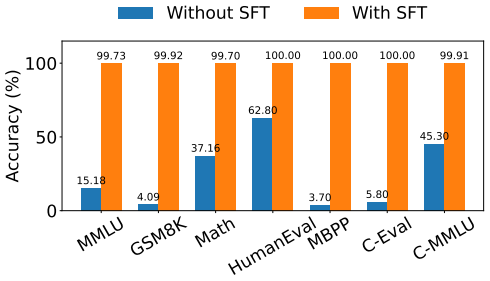



FIGURE 6.5: Effect of supervised fine-tuning (SFT) in model-level routing. We assess the performance of routing by measuring the accuracy on a 4-domain classification task (instruction, mathematics, code, and Chinese).

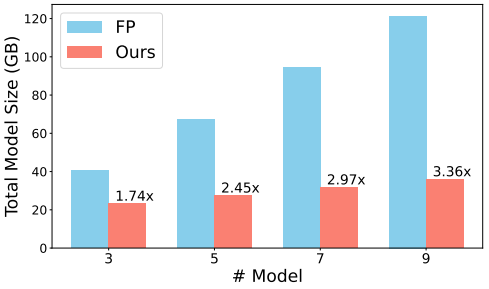



FIGURE 6.6: Model size reduction results in terms of Mistral-7B family. The model sizes for the a single 16-bit floating-point model, a compressed model, and the router are 13.48 GB, 2.13 GB, and 3.42 GB, respectively.

classification performance of the router (*i.e.*, Qwen1.5-1.8B-Chat) across four domains: instruction, mathematics, code, and Chinese. As shown in Figure 6.5, the pre-trained router performs poorly in domain classification without fine-tuning, achieving a Top-1 accuracy of only 5.80% on C-Eval. However, with SFT, the router's performance improves significantly, reaching nearly 100% accuracy across all domains. This demonstrates that supervised fine-tuning greatly enhances the instruction-following capabilities of the router, thereby improving its routing performance.

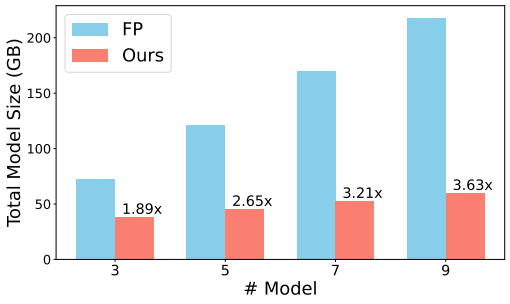



FIGURE 6.7: Model size reduction results in terms of LLaMA-13B family. The model sizes for a single FP16 model, a compressed model, and the router are 24.23 GB, 3.60 GB, and 3.42 GB, respectively.

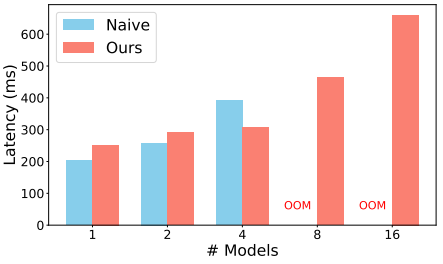



FIGURE 6.8: Decoding latency for Mistral-7B. "Naive" denotes the naive inference with $M$ fine-tuned models. "Ours" represents batch inference with our method. Out-of-memory scenarios are indicated as "OOM".

**Model size reduction analysis.** To investigate the model size reduction as discussed in Section 6.4.2, we compare the total storage requirements of full-precision models with those of our compressed models for the Mistral-7B and LLaMA-13B families across varying model counts, as shown in Figures 6.6 and 6.7. As the number of models increases, the compression ratios improve substantially. For instance, with nine models, our method achieves a 3.63× reduction compared to full-precision models for Mistral-7B

family. These savings become even more pronounced with larger model sizes, reaching up to a 3.63× reduction for LLaMA-13B family.

**Latency analysis.** To assess the latency improvements from delta weights compression, we measured the end-to-end decoding latency of the Mistral-7B model with an input sequence length of 128 on a single NVIDIA A100. Additional latency decomposition results are provided in Section 6.9.5 of the appendix. Decoding latency is critical, as it typically dominates processing time in LLM operations [115, 171]. Our efficient Triton kernel, which enables batched matrix multiplication between multiple compressed weight matrices and high-precision input activations, is compared against the conventional approach of individually processing multiple models. Results depicted in Figure 6.8 illustrate that while our method may perform slightly slower than the naive approach for a small number of models due to additional dequantization overhead, it provides lower latency as the number of models is greater than 4. More importantly, unlike the naive approach, our method is able to simultaneously serve 16 models on GPUs without running into out-of-memory (OOM) issues, demonstrating better scalability and efficiency in high-load scenarios.

## 6.8 Conclusion and Future Work

In this chapter, we have introduced ME-Switch, a memory-efficient expert switching framework designed for LLMs. Our method has addressed the critical challenge of balancing model performance with storage efficiency. The core of our ME-Switch lies in a novel mixed-precision quantization method that selectively compresses non-salient delta weights to extremely low-bit precision while preserving salient delta weights. Additionally, we have developed a model-level router that dynamically selects the most suitable LLM for a given query by transforming the model selection problem into a domain classification task. Extensive experiments on Mistral-7B and LLaMA-13B families have demonstrated that ME-Switch achieves performance comparable to unquantized expert models across various tasks while significantly reducing model size. In terms of limitations, our ME-Switch relies on accurate domain classification by the model-level router, which can sometimes misclassify and impact performance. Future work could explore more advanced or hybrid routing strategies, such as Mixture of Experts (MoE), to enhance accuracy. Additionally, quantizing the base model itself could further reduce the

overall model size. This approach would require careful consideration of the combined effects of quantizing both the base model and the delta weights to ensure performance is maintained. Furthermore, reducing the bitwidth of the KV Cache could accelerate the decoding speed, offering additional efficiency improvements.

## 6.9 Appendix

### 6.9.1 More Details About Prompt Template for Model-level Routing

In this section, we presents the prompt template for our model-level routing, as illustrated in Table 6.7. When a user query is received, it is embedded into the prompt template to form a structured question. This structured question is then processed by the router to perform domain classification.

TABLE 6.7: Prompt template for model-level routing.

| **Prompt template for model-level routing** |
|---|
| Classify the query based on the required expertise. Route the query to the appropriate model for a precise response. Only output the letter corresponding to the best category (A, B, C, . . . , F). |
| Query: {Insert the user's query here. } |
| Options: A) Instruct - For general guidance, explanations, or broad advice. B) Code - For programming-related queries, like debugging or coding. C) Math - For mathematical inquiries, such as problems or theories. D) Chinese Language Expert - For inquiries related to the Chinese language, including translation, grammar, and usage. . . . F) {Specify additional categories and their descriptions here.} |
| Response should be only 'A', 'B', 'C', . . . or 'F", with no additional text. |

### 6.9.2 More Performance Comparisons With Different Weight-only Quantization Methods

We present the detailed results of Figure 6.4 in Table 6.8. A comprehensive analysis is available in Section 6.7.

### 6.9.3 More Analysis on the Rank of Delta Weights

We show the cumulative energy of delta weights for MetaMath-Mistral-7B in Figure 6.9, using squared singular values to measure the "energy" of the projection matrix. The

TABLE 6.8: Performance comparisons with different weight-only quantization methods.

| Domain | Dataset | FP | 1-bit | AWQ | Random | Wanda | Magnitude | Ours |
|---|---|---|---|---|---|---|---|---|
| Instruct (%) ↑ | MMLU | 63.43 | 63.14 | 63.32 | 63.19 | 63.19 | 63.04 | **63.43** |
| Math (%) ↑ | GSM8K | 73.92 | 53.45 | 53.75 | 54.66 | 58.07 | 54.89 | 59.14 |
| | Math | 20.62 | 1.50 | 1.72 | 1.82 | 2.14 | 1.64 | 1.74 |
| | Avg. | 47.27 | 27.48 | 27.74 | 28.24 | 30.11 | 28.27 | **30.44** |
| Code (%) ↑ | HumanEval | 51.20 | 47.00 | 47.00 | 46.30 | 48.20 | 48.20 | 47.60 |
| | MBPP | 60.40 | 58.40 | 58.10 | 58.60 | 58.60 | 58.90 | 60.70 |
| | Avg. | 55.80 | 52.70 | 52.55 | 52.45 | 53.40 | 53.55 | **54.15** |
| Domain | Dataset | FP | 2-bit | AWQ | Random | Wanda | Magnitude | Ours |
| Instruct (%) ↑ | MMLU | 63.43 | 63.72 | 63.65 | 63.71 | 63.90 | 63.94 | **63.95** |
| Math (%) ↑ | GSM8K | 73.92 | 73.31 | 73.24 | 73.01 | 72.71 | 73.16 | 73.62 |
| | Math | 20.62 | 20.44 | 19.98 | 20.52 | 20.64 | 20.08 | 20.48 |
| | Avg. | 47.27 | 46.88 | 46.61 | 46.77 | 46.68 | 46.62 | **47.05** |
| Code (%) ↑ | HumanEval | 51.20 | 47.00 | 47.00 | 48.80 | 50.60 | 50.00 | 51.80 |
| | MBPP | 60.40 | 59.10 | 60.20 | 59.60 | 59.90 | 60.20 | 60.70 |
| | Avg. | 55.80 | 53.05 | 53.60 | 54.20 | 55.25 | 55.10 | **56.25** |

results show that all projection layers consistently exhibit a similar trend and possess a relatively high rank. Therefore, due to the absence of low-rank properties, LoRA cannot accurately approximate the delta weights of full fine-tuned models.

### 6.9.4 BERT *vs.* Small LLM for Model-level Routing

In addition to smaller LLMs, we can also use BERT for domain classification given user queries. Specifically, we employ DistillBERT [124] as the backbone and fine-tune it on our collected dataset (described in Section 6.4.2) with just the queries and corresponding domain labels. We show the router accuracy in Table 6.9 and the latency comparisons in Table 6.10. From the results, DistillBERT with a faster response time performs well across most datasets, except for MMLU where its limited capacity struggles with complex data. In contrast, our method consistently achieves good performance across all datasets, demonstrating its effectiveness even in complex scenarios. Moreover, our router's inference latency is just 17.60ms for sequence lengths of 128, which is less than 7% of the inference time for expert models. Therefore, we continue to leverage Qwen1.5-1.8B-Chat for its proven effectiveness in challenging scenarios.

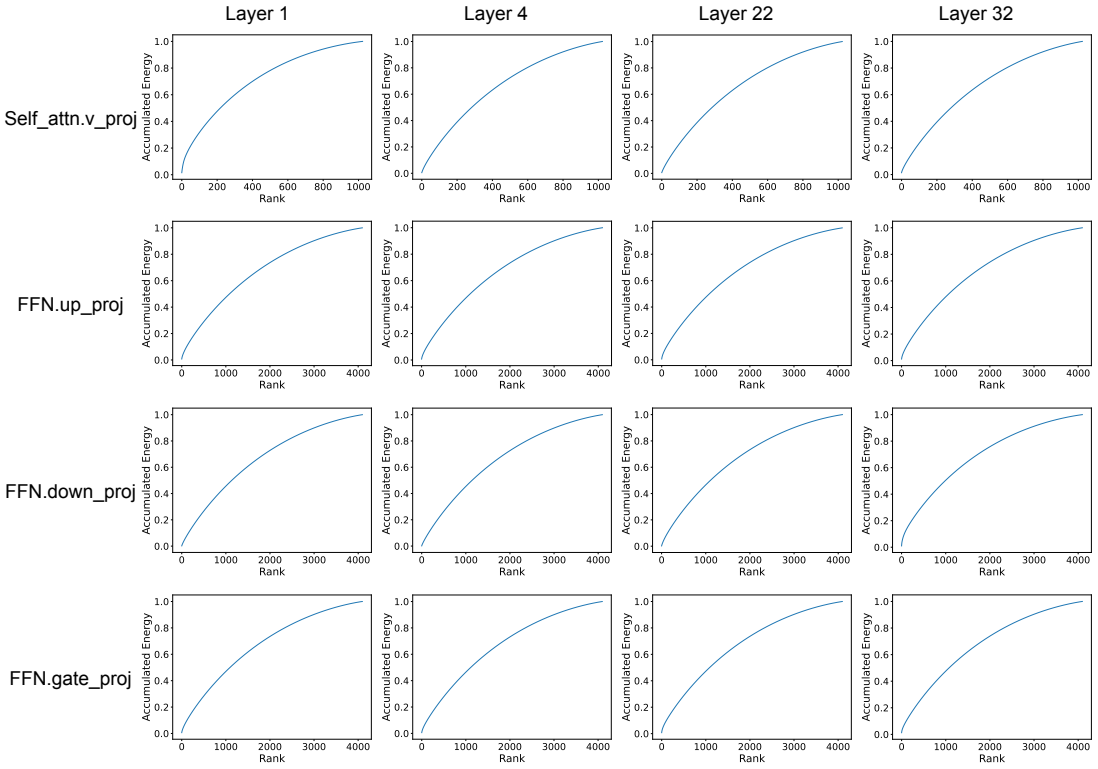



FIGURE 6.9: An illustration showing the cumulative energy of delta weights for MetaMath-Mistral-7B model derived through Singular Value Decomposition (SVD).

TABLE 6.9: Router performance comparisons.

| Model | MMLU | GSM8K | Math | HumanEval | MBPP | C-Eval | C-MMLU |
|---|---|---|---|---|---|---|---|
| DistillBERT | 73.29 | **100.00** | **99.80** | 96.34 | **100.00** | 99.97 | **100.00** |
| Qwen1.5-1.8B-Chat | **99.73** | 99.92 | 99.70 | **100.00** | **100.00** | **100.00** | 99.91 |

TABLE 6.10: Router latency (ms) comparisons.

| Sequence Length | 128 | 256 | 512 |
|---|---|---|---|
| DistillBERT | **3.70** | **3.80** | **4.90** |
| Qwen1.5-1.8B-Chat | 17.60 | 18.30 | 19.10 |

### 6.9.5 More Results on Latency Decomposition

We provide a detailed breakdown of decoding times for Mistral-7B model in Figure 6.10, using an input sequence length of 128 on a single NVIDIA A100 GPU. Since loading delta weights, dequantization, and matrix multiplication are integrated into our efficient Triton kernel, it is challenging to isolate and measure the latency of each individual component separately. Therefore, we measure the inference time for the operations $\mathbf{xW}$ and $\mathbf{x}\tilde{\Delta}$. Notably, as the number of models increases, the increased latency attributed

to $\mathbf{x}\tilde{\Delta}$ exceeds that of $\mathbf{xW}$, primarily due to the augmented loading time of the more compressed delta weights.

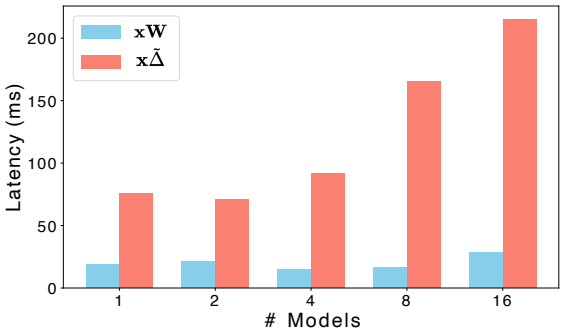



FIGURE 6.10: Latency decomposition of ME-Switch for Mistral-7B on a NVIDIA A100 80G GPU.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

In this thesis, we have introduced four novel methods aimed at mitigating the performance degradation caused by quantization. The first two approaches focus on addressing challenges in QAT, while the latter two tackle issues specific to PTQ. Specifically, we have addressed the following key challenges encountered in quantization:

- Low-precision models represent weights and activations using a limited set of values, which significantly restricts their expressive capacity. This limitation causes drastic loss fluctuations, as small changes in full-precision weights can lead to disproportionately large shifts in quantized weights due to the rounding operation, making the optimization process unstable. These fluctuations create a much sharper loss landscape, severely degrading model performance during training. To address this challenge, in Chapter 3, we have introduced Sharpness-Aware Quantization (SAQ), a method that combines quantization with loss landscape smoothing to improve generalization. SAQ utilizes Sharpness-Aware Minimization (SAM) to smooth the loss landscape and incorporates an efficient training strategy to minimize additional computational overhead, ultimately enhancing the performance of quantized models.

- For extremely low-bitwidth quantization, such as binary quantization, the primary focus is on minimizing quantization errors for individual tokens. However, this approach often overlooks the preservation of pairwise token similarity, which is essential

for the effectiveness of attention mechanisms, leading to performance degradation. Moreover, the computational cost of attention remains prohibitively high due to its quadratic complexity with respect to token length, presenting significant challenges in long-context scenarios. To address these issues, in Chapter 4, we have introduced a novel binarization method that maps queries and keys into compact binary codes using kernelized hashing. These kernelized hash functions are learned in a self-supervised manner to preserve the ground-truth token similarity relations derived from the attention map. Leveraging the associative property of matrix multiplication, the attention operation can then be approximated with linear complexity, significantly reducing computational overhead.

- For quantization in LLMs, we encounter challenges that differ from those in traditional CNNs and smaller Transformers. Specifically, activation outliers with significantly large magnitudes in certain channels make quantization less effective by expanding the quantization range of activations, leading to substantial performance degradation. Furthermore, due to the vast number of parameters in LLMs, even post-training quantization (PTQ) incurs prohibitively high computational costs. To address these issues, in Chapter 5, we have introduced a simple yet effective channel reassembly method that redistributes activation outliers across channels, making the activations more quantization-friendly. Additionally, we have proposed an efficient error correction method that introduces learnable low-rank parameters to mitigate quantization errors while significantly reducing training costs.

- Deploying multiple LLMs can impose a significant memory burden. To address this issue, one approach is to decompose model weights into pre-trained weights and delta weights introduced during fine-tuning. By quantizing these delta weights, it becomes possible to efficiently share the storage of a base model across various fine-tuned versions. However, differences in delta weight values between input and output channels can cause information loss during quantization, ultimately leading to a decline in model performance. To address these, in Chapter 6, we have introduced a memory-efficient framework called ME-Switch to address the memory constraints of deploying multiple LLMs. This framework has been designed to store only a single full-precision pre-trained model and dynamically load compressed delta model weights based on user queries. To achieve this, we have developed a mixed-precision quantization strategy, where the majority of non-salient weights are quantized to extremely low bit-widths,

while salient weights are retained in full precision. Furthermore, we have proposed a model-level routing mechanism that dynamically selects the most appropriate LLM for each query.

## 7.2 Future Work

We have focused on using quantization to enhance inference efficiency for Transformers in both vision and languages. While these efforts have demonstrated significant improvements, enhancing the efficiency of large foundation models remains a complex and ongoing challenge. There are still several promising directions for future exploration.

- Incorporating both QAT and PTQ techniques into future work could also offer a balanced approach to model efficiency and performance. While QAT provides higher accuracy by integrating quantization into the training process, it requires significant computational resources. On the other hand, PTQ offers a more resource-efficient solution, using only a small amount of data for calibration. However, PTQ still tends to suffer from performance degradation, particularly for more complex models or when quantizing to very low bit-widths. To mitigate this issue, one promising direction is to develop hybrid methods that combine the strengths of both QAT and PTQ.
- As multi-modal large foundation models gain popularity, applying quantization to these models remains an underexplored area. A key question is whether multi-modal models exhibit the same outlier patterns as unimodal models, such as vision Transformers or LLMs. Understanding the behavior of activation outliers and the effects of quantization across different modalities—such as vision, text, and audio—could pave the way for developing more effective quantization strategies tailored specifically for multi-modal models.
- Beyond improving inference efficiency, enhancing training efficiency for efficient learning presents another intriguing direction for future research. For instance, NVIDIA's latest GPU DGX B200, offers twice the FP4 throughput compared to FP8, highlighting the potential of using lower precision formats for faster and more efficient training. Exploring how such advancements can be leveraged for training deep learning models, particularly large foundation models, could reduce the computational burden and resource requirements while maintaining performance.

- Recently, training-free compression likes token merging [249, 250, 286] and KV cache merging [287] offer a promising direction for effective compression with minimal cost. However, these methods often experience significant performance degradation as the compression ratio increases. To mitigate this, combining quantization with training-free compression to achieve high compression ratios presents a promising solution. This approach enables the model to jointly consider the effects of both techniques within a unified objective, allowing it to automatically determine the optimal compression ratio for each method.

- Apart from quantization, lottery ticket-based pruning [288, 289], which focuses on identifying and retaining critical subnetworks during training, presents another effective approach for model size reduction. Integrating this approach during quantization training could offer additional compression benefits. Specifically, by identifying important subnetworks and applying quantization techniques directly during the training phase, we could simultaneously reduce both the number of weights (via lottery ticket pruning) and the precision of the remaining parameters (via quantization). This combined strategy could lead to more efficient models by focusing on optimizing the network's structure and precision simultaneously.

# Bibliography


[1] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Adv. Neural Inform. Process. Syst.*, 2018.

[2] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *Int. Conf. Mach. Learn.*, pages 38087–38099. PMLR, 2023.

[3] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In *Proc. Mach. Learn. Syst.*, volume 1, pages 348–359, 2019.

[4] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: hardware-aware automated quantization with mixed precision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8612–8620, 2019.

[5] Eunhyeok Park and Sungjoo Yoo. Profit: A novel training method for sub-4-bit mobilenet models. In *Eur. Conf. Comput. Vis.*, pages 430–446. Springer, 2020.

[6] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Comput. Archit. News*, 44(3):243–254, 2016.

[7] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *IEEE Int. Conf. Solid-State Circuits*, pages 10–14, 2014.

[8] Xiaocong Lian, Zhenyu Liu, Zhourui Song, Jiwu Dai, Wei Zhou, and Xiangyang Ji. High-performance fpga-based cnn accelerator with block-floating-point arithmetic. *IEEE Trans. Very Large Scale Integr.*, 27(8):1874–1885, 2019.

[9] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pvtv2: Improved baselines with pyramid vision transformer. *Comput. Vis. Media*, 8(3):415–424, 2022.

[10] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *Int. Symp. Comput. Archit.*, pages 764–775, 2018.

[11] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *Int. Conf. Learn. Represent.*, 2017.

[12] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, 1993.

[13] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1):5485–5551, 2020.

[14] D Jannach. *Recommender Systems: An Introduction*. Cambridge University Press, 2010.

[15] Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016.

[16] Jean-Paul Laumond et al. *Robot motion planning and control*, volume 229. Springer, 1998.

[17] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[18] Ravi Aggarwal, Viknesh Sounderajah, Guy Martin, Daniel SW Ting, Alan Karthikesalingam, Dominic King, Hutan Ashrafian, and Ara Darzi. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. *NPJ digit. med.*, 4(1):65, 2021.

[19] Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. Large language models in medicine. *Nat. med.*, 29(8):1930–1940, 2023.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[21] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[22] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, volume 30, 2017.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inform. Process. Syst.*, 25, 2012.

[25] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE J. Sel. Top. Signal Process.*, 13(2):206–219, 2019.

[26] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[27] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Adv. Neural Inform. Process. Syst.*, volume 33, pages 1877–1901, 2020.

[28] Vasudev Bhaskaran and Konstantinos Konstantinides. *Image and video compression standards - algorithms and architectures, Second Edition*, volume 408 of *The Kluwer international series in engineering and computer science*. Kluwer, 1997. ISBN 978-0-7923-9952-0.

[29] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *IEEE intell. veh. symp.*, pages 1013–1020. IEEE, 2018.

[30] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Commun. ACM*, 63(12):54–63, 2020.

[31] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[32] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2704–2713, 2018.

[33] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *Int. Conf. Learn. Represent.*, 2020.

[34] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: Data-free quantization aware training for large language models. In *Find. Assoc. Comput. Linguist.*, pages 467–484, August 2024.

[35] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *Int. Conf. Mach. Learn.*, pages 7197–7206. PMLR, 2020.

[36] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 17402–17414, 2022.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016.

[38] Sachin Mehta and Mohammad Rastegari. Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer. In *Int. Conf. Learn. Represent.*, 2022.

[39] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, 2023.

[40] Zechun Liu, Zhiqiang Shen, Shichao Li, Koen Helwegen, Dong Huang, and Kwang-Ting Cheng. How do adam and training strategies help bnns optimization. In *Int. Conf. Mach. Learn.*, volume 139, pages 6936–6946, 2021.

[41] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Eur. Conf. Comput. Vis.*, pages 525–542, 2016.

[42] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. Binarybert: Pushing the limit of bert quantization. In *Assoc. Comput. Linguist.*, pages 4334–4348, 2021.

[43] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3.int8(): 8-bit matrix multiplication for transformers at scale. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 30318–30332, 2022.

[44] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. Learn. Represent.*, 2021.

[46] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Int. Conf. Comput. Vis.*, pages 568–578, 2021.

[47] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Int. Conf. Comput. Vis.*, pages 10012–10022, 2021.

[48] Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. Scalable vision transformers with hierarchical pooling. In *Int. Conf. Comput. Vis.*, pages 377–386, 2021.

[49] Zizheng Pan, Bohan Zhuang, Haoyu He, Jing Liu, and Jianfei Cai. Less is more: Pay less attention in vision transformers. In *Assoc. Adv. Artif. Intell.*, volume 36, pages 2035–2043, 2022.

[50] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. In *Int. Conf. Comput. Vis.*, pages 6824–6835, 2021.

[51] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. In *Int. Conf. Comput. Vis.*, pages 22–31, 2021.

[52] Yawei Li, Kai Zhang, Jiezhang Cao, Radu Timofte, and Luc Van Gool. Localvit: Bringing locality to vision transformers. *arXiv preprint arXiv:2104.05707*, 2021.

[53] Yufei Xu, Qiming Zhang, Jing Zhang, and Dacheng Tao. Vitae: Vision transformer advanced by exploring intrinsic inductive bias. *Adv. Neural Inform. Process. Syst.*, 34, 2021.

[54] Kun Yuan, Shaopeng Guo, Ziwei Liu, Aojun Zhou, Fengwei Yu, and Wei Wu. Incorporating convolution designs into visual transformers. In *Int. Conf. Comput. Vis.*, pages 579–588, 2021.

[55] Stéphane d'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *Int. Conf. Mach. Learn.*, pages 2286–2296, 2021.

[56] Xuanhong Chen, Hang Wang, and Bingbing Ni. X-volution: on the unification of convolution and self-attention. *arXiv preprint arXiv:2106.02253*, 2021.

[57] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Int. Conf. Comput. Vis.*, pages 12259–12269, 2021.

[58] Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. Cmt: Convolutional neural networks meet vision transformers. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12175–12185, 2022.

[59] Haoyu He, Jianfei Cai, Jing Liu, Zizheng Pan, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Pruning self-attentions into convolutional layers in single path. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(5):3910–3922, 2024.

[60] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *N. Am. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol.*, pages 464–468, 2018.

[61] Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. Character-level language modeling with deeper self-attention. In *Assoc. Adv. Artif. Intell.*, volume 33, pages 3159–3166, 2019.

[62] Xiangxiang Chu, Zhi Tian, Bo Zhang, Xinlong Wang, and Chunhua Shen. Conditional positional encodings for vision transformers. In *Int. Conf. Learn. Represent.*, 2023.

[63] Kan Wu, Houwen Peng, Minghao Chen, Jianlong Fu, and Hongyang Chao. Rethinking and improving relative position encoding for vision transformer. In *Int. Conf. Comput. Vis.*, pages 10033–10041, 2021.

[64] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Int. Conf. Comput. Vis.*, pages 9650–9660, 2021.

[65] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Trans. Mach. Learn. Res.*, 2024. ISSN 2835-8856.

[66] S Hochreiter. Long short-term memory. *Neural Comput. MIT-Press*, 1997.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *N. Am. Chapter Assoc. Comput. Linguist.: Hum. Lang. Technol.*, pages 4171–4186, June 2019.

[68] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Conf. Empir. Methods Nat. Lang. Process.*, 2016.

[69] Wei Wang, Ming Yan, and Chen Wu. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *Assoc. Comput. Linguist.*, pages 1705–1714. Association for Computational Linguistics, July 2018.

[70] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. Conf. Nat. Lang. Learn. HLT-NAACL*, pages 142–147, 2003.

[71] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE Trans. Knowl. Data Eng.*, 34(1):50–70, 2020.

[72] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Assoc. Comput. Linguist.*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[73] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(120):1–39, 2022.

[74] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[75] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A

robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019. URL https://api.semanticscholar.org/CorpusID:198953378.

[76] Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. Multitask prompted training enables zero-shot task generalization. In *Int. Conf. Learn. Represent.*, 2022. URL https://openreview.net/forum?id=9Vrb9D0WI4.

[77] Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. What language model architecture and pretraining objective works best for zero-shot generalization? In *Int. Conf. Mach. Learn.*, pages 22964–22984. PMLR, 2022.

[78] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[79] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models. *ArXiv*, abs/2205.01068, 2022.

[80] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ili'c, Daniel Hesslow, Roman Castagn'e, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurenccon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi,

Aitor Soroa Etxabe, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris C. Emezue, Christopher Klamm, Colin Leong, Daniel Alexander van Strien, David Ifeoluwa Adelani, Dragomir R. Radev, Eduardo Gonz'alez Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Trung Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jorg Frohberg, Josephine Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro von Werra, Leon Weber, Long Phan, Loubna Ben Allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, Mar'ia Grandury, Mario vSavsko, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto L'opez, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, S. Longpre, Somaieh Nikpoor, S. Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-Shaibani, Matteo Manica, Nihal V. Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Févry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiang Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Y Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre Franccois Lavall'ee, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil,

Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure Ligozat, Arjun Subramonian, Aur'elie N'ev'eol, Charles Lovering, Daniel H Garrette, Deepak R. Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Xiangru Tang, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, S. Osher Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdenvek Kasner, Zdeněk Kasner, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ananda Santa Rosa Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ayoade Ajibade, Bharat Kumar Saxena, Carlos Muñoz Ferrandis, Danish Contractor, David M. Lansky, Davis David, Douwe Kiela, Duong Anh Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatim Tahirah Mirza, Frankline Ononiwu, Habib Rezanejad, H.A. Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jan Passmore, Joshua Seltzer, Julio Bonis Sanz, Karen Fort, Lívia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nourhan Fahmy, Olanrewaju Samuel, Ran An, R. P. Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas L. Wang, Sourav Roy, Sylvain Viguier, Thanh-Cong Le, Tobi Oyebade, Trieu Nguyen Hai Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Kumar Singh, Benjamin Beilharz, Bo Wang, Caio Matheus Fonseca de Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel Le'on Perin'an, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Iman I.B. Bello, Isha Dash, Ji Soo Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthi Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, María Andrea Castillo, Marianna Nezhurina, Mario Sanger, Matthias Samwald, Michael Cullan, Michael Weinberg, M Wolf,

Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patricia Haller, Patrick Haller, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Pratap Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yashasvi Bajaj, Y. Venkatraman, Yifan Xu, Ying Xu, Yu Xu, Zhee Xao Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. Bloom: A 176b-parameter open-access multilingual language model. *ArXiv*, abs/2211.05100, 2022.

[81] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[82] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[83] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[84] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[85] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Trans. Mach. Learn. Res.*, 2022. ISSN 2835-8856.

[86] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[87] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[88] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *Int. Conf. Learn. Represent.*, 2021.

[89] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[90] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[91] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 17283–17297, 2020.

[92] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *Int. Conf. Learn. Represent.*, 2020.

[93] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[94] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H Abdi, Dongsheng Li, Chin-Yew Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. *arXiv preprint arXiv:2407.02490*, 2024.

[95] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient LLM inference. In *Int.*

*Conf. Learn. Represent. Worksh.*, 2024. URL https://openreview.net/forum?id=Ue8EHzaFI4.

[96] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *Int. Conf. Learn. Represent.*, 2024.

[97] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *Int. Conf. Learn. Represent.*, 2024.

[98] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking attention with performers. In *Int. Conf. Learn. Represent.*, 2021.

[99] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Assoc. Adv. Artif. Intell.*, volume 35, page 14138, 2021.

[100] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. Soft: Softmax-free transformer with linear complexity. In *Adv. Neural Inform. Process. Syst.*, volume 34, 2021.

[101] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Int. Conf. Mach. Learn.*, pages 5156–5165, 2020.

[102] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. In *Int. Conf. Learn. Represent.*, 2022.

[103] Jing Liu, Zizheng Pan, Haoyu He, Jianfei Cai, and Bohan Zhuang. Ecoformer: Energy-saving attention with linear complexity. *Adv. Neural Inform. Process. Syst.*, 35:10295–10308, 2022.

[104] Haoran You, Yunyang Xiong, Xiaoliang Dai, Bichen Wu, Peizhao Zhang, Haoqi Fan, Peter Vajda, and Yingyan Celine Lin. Castling-vit: Compressing self-attention via switching towards linear-angular attention at vision transformer inference. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14431–14442, 2023.

[105] Haoran You, Yichao Fu, Zheng Wang, Amir Yazdanbakhsh, and Yingyan Celine Lin. When linear attention meets autoregressive decoding: Towards more effective and efficient linearized large language models. In *Int. Conf. Mach. Learn.*, 2024.

[106] Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1468–1477, 2020.

[107] Han Shu, Jiahao Wang, Hanting Chen, Lin Li, Yujiu Yang, and Yunhe Wang. Adder attention for vision transformer. In *Adv. Neural Inform. Process. Syst.*, volume 34, 2021.

[108] Haoran You, Huihong Shi, Yipin Guo, and Yingyan Lin. Shiftaddvit: Mixture of multiplication primitives towards efficient vision transformer. In *Adv. Neural Inform. Process. Syst.*, volume 36, pages 33319–33337, 2023.

[109] Haoran You, Baopu Li, Shi Huihong, Yonggan Fu, and Yingyan Lin. ShiftAddNAS: Hardware-inspired search for more accurate and efficient neural networks. In *Int. Conf. Mach. Learn.*, volume 162, pages 25566–25580, 17–23 Jul 2022.

[110] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Adv. Neural Inform. Process. Syst.*, volume 32. Curran Associates, Inc., 2019.

[111] Mingjian Zhu, Yehui Tang, and Kai Han. Vision transformer pruning. *arXiv preprint arXiv:2104.08500*, 2021.

[112] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *Adv. Neural Inform. Process. Syst.*, volume 34, pages 19974–19988, 2021.

[113] Shixing Yu, Tianlong Chen, Jiayi Shen, Huan Yuan, Jianchao Tan, Sen Yang, Ji Liu, and Zhangyang Wang. Unified visual transformer compression. In *Int. Conf. Learn. Represent.*, 2022.

[114] Zejiang Hou and Sun-Yuan Kung. Multi-dimensional model compression of vision transformer. In *Int. Conf. Multimedia and Expo*, pages 01–06. IEEE, 2022.

[115] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *Int. Conf. Mach. Learn.*, pages 22137–22176. PMLR, 2023.

[116] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. In *Adv. Neural Inform. Process. Syst.*, 2023.

[117] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. In *Int. Conf. Learn. Represent.*, 2024.

[118] Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. Dynamicvit: Efficient vision transformers with dynamic token sparsification. In *Adv. Neural Inform. Process. Syst.*, 2021.

[119] Bowen Pan, Rameswar Panda, Yifan Jiang, Zhangyang Wang, Rogerio Feris, and Aude Oliva. Ia-redˆ2: Interpretability-aware redundancy reduction for vision transformers. In *Adv. Neural Inform. Process. Syst.*, volume 34, pages 24898–24911. Curran Associates, Inc., 2021.

[120] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8607–8617, 2021.

[121] Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not all patches are what you need: Expediting vision transformers via token reorganizations. *arXiv preprint arXiv:2202.07800*, 2022.

[122] Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. LoRAPrune: Structured pruning meets low-rank

parameter-efficient fine-tuning. In *Find. Assoc. Comput. Linguist.*, pages 3013–3026, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.178. URL https://aclanthology.org/2024.findings-acl.178/.

[123] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Adv. Neural Inform. Process. Syst.*, 36:21702–21720, 2023.

[124] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

[125] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In *Find. Assoc. Comput. Linguist.: Conf, Empir, Methods Nat, Lang, Process,*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics.

[126] Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*, 2023.

[127] Chen Liang, Simiao Zuo, Qingru Zhang, Pengcheng He, Weizhu Chen, and Tuo Zhao. Less is more: Task-aware layer-wise distillation for language model compression. In *Int. Conf. Mach. Learn.*, pages 20852–20867. PMLR, 2023.

[128] Inar Timiryasov and Jean-Loup Tastet. Baby llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty. *arXiv preprint arXiv:2308.02019*, 2023.

[129] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In *Int. Conf. Learn. Represent.*, 2024.

[130] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Assoc. Adv. Artif. Intell.*, volume 34, pages 8815–8821, 2020.

[131] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Conf. Empir. Methods Nat. Lang. Process.*, pages 509–521, November 2020.

[132] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. In *Int. Conf. Mach. Learn.*, pages 5506–5518. PMLR, 2021.

[133] Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua YAN, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. BiBERT: Accurate fully binarized BERT. In *Int. Conf. Learn. Represent.*, 2022.

[134] Zechun Liu, Barlas Oguz, Aasish Pappu, Lin Xiao, Scott Yih, Meng Li, Raghuraman Krishnamoorthi, and Yashar Mehdad. Bit: Robustly binarized multi-distilled transformer. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 14303–14316. Curran Associates, Inc., 2022.

[135] Jing Liu, Ruihao Gong, Xiuying Wei, Zhiwei Dong, Jianfei Cai, and Bohan Zhuang. QLLM: Accurate and efficient low-bitwidth quantization for large language models. In *Int. Conf. Learn. Represent.*, 2024.

[136] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Eur. Conf. Comput. Vis.*, pages 373–390, 2018.

[137] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4350–4359, 2019.

[138] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. In *Int. Conf. Learn. Represent.*, 2021.

[139] Tiantian Han, Dong Li, Ji Liu, Lu Tian, and Yi Shan. Improving low-precision network quantization via bin regularization. In *Int. Conf. Comput. Vis.*, pages 5261–5270, 2021.

[140] Yefei He, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. EfficientDM: Efficient quantization-aware fine-tuning of low-bit diffusion models. In *Int. Conf. Learn. Represent.*, 2024.

[141] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Adv. Neural Inform. Process. Syst.*, volume 29, 2016.

[142] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Structured binary neural networks for accurate image classification and semantic segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 413–422, 2019.

[143] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Eur. Conf. Comput. Vis.*, pages 722–737, 2018.

[144] Haotong Qin, Mingyuan Zhang, Yifu Ding, Aoyu Li, Zhongang Cai, Ziwei Liu, Fisher Yu, and Xianglong Liu. BiBench: Benchmarking and analyzing network binarization. In *Int. Conf. Mach. Learn.*, volume 202, pages 28351–28388. PMLR, 23–29 Jul 2023.

[145] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Lifeng Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits. *ArXiv*, abs/2402.17764, 2024.

[146] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5406–5414, 2017.

[147] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, pages 2978–2985, 2020.

[148] Ruizhou Ding, Ting-Wu Chin, Zeye Liu, and Diana Marculescu. Regularizing activation distribution for training binarized deep networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11408–11417, 2019.

[149] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *Int. Conf. Learn. Represent.*, 2019.

[150] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized neural networks with a full-precision auxiliary module. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1485–1494, 2020.

[151] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *Int. Conf. Learn. Represent.*, 2020.

[152] Shoukai Xu, Haokun Li, Bohan Zhuang, Jing Liu, Jiezhang Cao, Chuangrun Liang, and Mingkui Tan. Generative low-bitwidth data free quantization. In *Eur. Conf. Comput. Vis.*, pages 1–17, 2020.

[153] Peng Chen, Jing Liu, Bohan Zhuang, Mingkui Tan, and Chunhua Shen. Aqd: Towards accurate quantized object detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[154] Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. In *Int. Conf. Learn. Represent.*, 2020.

[155] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *Int. Conf. Mach. Learn.*, pages 11875–11886, 2021.

[156] Weihan Chen, Peisong Wang, and Jian Cheng. Towards mixed-precision quantization of neural networks via constrained optimization. In *Int. Conf. Comput. Vis.*, pages 5350–5359, 2021.

[157] Ivan Koryakovskiy, Alexandra Yakovleva, Valentin Buchnev, Temur Isaev, and Gleb Odinokikh. One-shot model for mixed-precision quantization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7939–7949, June 2023.

[158] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Int. Conf. Comput. Vis.*, pages 1325–1334, 2019.

[159] Yury Nahshan, Brian Chmiel, Chaim Baskin, Evgenii Zheltonozhskii, Ron Banner, Alexander M. Bronstein, and Avi Mendelson. Loss aware post-training quantization. *Mach. Learn.*, 110:3245 – 3262, 2019.

[160] Di Wu, Qi Tang, Yongle Zhao, Ming Zhang, Ying Fu, and Debing Zhang. Easyquant: Post-training quantization via scale optimization. *arXiv preprint arXiv:2006.16669*, 2020.

[161] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

[162] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. In *Int. Conf. Learn. Represent.*, 2021.

[163] Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kai-Chuang Zhang, Yu Qiao, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models. *ArXiv*, abs/2407.11062, 2024.

[164] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. In *Int. Conf. Learn. Represent.*, 2024.

[165] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Adv. Neural Inform. Process. Syst.*, volume 32, 2019.

[166] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient inference. In *Workshop Energy Effic. Mach. Learn. Cogn. Comput.-NeurIPS Ed.*, pages 6–9. IEEE, 2019.

[167] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *Int. Conf. Comput. Vis. Worksh.*, pages 3009–3018. IEEE, 2019.

[168] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *Int. Conf. Mach. Learn.*, pages 7543–7552, 2019.

[169] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *Int. Conf. Learn. Represent.*, 2022.

[170] Gunho Park, Baeseong park, Minsub Kim, Sungjae Lee, Jeonghoon Kim, Beomseok Kwon, Se Jung Kwon, Byeongwook Kim, Youngjoo Lee, and Dongsoo Lee. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models. In *Int. Conf. Learn. Represent.*, 2024.

[171] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. In *Conf. Mach. Learn. Syst.*, 2024.

[172] Tim Dettmers, Ruslan A. Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *Int. Conf. Learn. Represent.*, 2024.

[173] Yuji Chai, John Gkountouras, Glenn G Ko, David Brooks, and Gu-Yeon Wei. Int2.1: Towards fine-tunable quantized large language models with error correction through low-rank adaptation. *arXiv preprint arXiv:2306.08162*, 2023.

[174] Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, and Kaokao Lv. Optimize weight rounding via signed gradient descent for the quantization of llms. *arXiv preprint arXiv:2309.05516*, 2023.

[175] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Adv. Neural Inform. Process. Syst.*, 2023.

[176] Yixiao Li, Yifan Yu, Chen Liang, Nikos Karampatziakis, Pengcheng He, Weizhu Chen, and Tuo Zhao. Loftq: LoRA-fine-tuning-aware quantization for large language models. In *Int. Conf. Learn. Represent.*, 2024.

[177] Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. In *Adv. Neural Inform. Process. Syst.*, volume 36, pages 36187–36207, 2023.

[178] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. In *Adv. Neural Inform. Process. Syst.*, volume 36, pages 4396–4429, 2023.

[179] Changhun Lee, Jungyu Jin, Taesu Kim, Hyungjun Kim, and Eunhyeok Park. Owq: Lessons learned from activation outliers for weight quantization in large language models. *arXiv preprint arXiv:2306.02272*, 2023.

[180] Xiuying Wei, Yunchen Zhang, Yuhang Li, Xiangguo Zhang, Ruihao Gong, Jinyang Guo, and Xianglong Liu. Outlier suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling. In *Conf. Empir. Methods Nat. Lang. Process.*, 2023.

[181] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 27168–27183, 2022.

[182] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. Exploring post-training quantization in llms from comprehensive study to low rank compensation. In *Assoc. Adv. Artif. Intell.*, 2024.

[183] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. Rptq: Reorder-based post-training quantization for large language models. *arXiv preprint arXiv:2304.01089*, 2023.

[184] Xiaoxia Wu, Zhewei Yao, and Yuxiong He. Zeroquant-fp: A leap forward in llms post-training w4a8 quantization using floating-point formats. *arXiv preprint arXiv:2307.09782*, 2023.

[185] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant–llm quantization with learned rotations. *arXiv preprint arXiv:2405.16406*, 2024.

[186] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Adv. Neural Inform. Process. Syst.*, volume 28, 2015.

[187] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Eur. Conf. Comput. Vis.*, pages 213–229. Springer, 2020.

[188] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Int. Conf. Learn. Represent.*, 2019.

[189] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Adv. Neural Inform. Process. Syst.*, 2017.

[190] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Adv. Neural Inform. Process. Syst.*, pages 529–536, 1995.

[191] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Int. Conf. Learn. Represent.*, 2017.

[192] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Conf. Uncertain. Artif. Intell.*, 2017.

[193] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *Int. Conf. Learn. Represent.*, 2020.

[194] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *Int. Conf. Learn. Represent.*, 2021.

[195] Jungmin Kwon, Jeongseop Kim, Hyun-Seok Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In *Int. Conf. Mach. Learn.*, 2021.

[196] Juntang Zhuang, Boqing Gong, Liangzhe Yuan, Yin Cui, Hartwig Adam, Nicha C Dvornek, James s Duncan, Ting Liu, et al. Surrogate gap minimization improves sharpness-aware training. In *Int. Conf. Learn. Represent.*, 2022.

[197] Minyoung Kim, Da Li, Shell X Hu, and Timothy Hospedales. Fisher sam: Information geometry and sharpness aware minimisation. In *Int. Conf. Mach. Learn.*, pages 11148–11161, 2022.

[198] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

[199] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards efficient and scalable sharpness-aware minimization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12360–12370, 2022.

[200] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 248–255, 2009.

[201] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Adv. Neural Inform. Process. Syst.*, volume 32, 2019.

[202] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4510–4520, 2018.

[203] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *Int. Conf. Learn. Represent.*, 2017.

[204] Zhexin Li, Tong Yang, Peisong Wang, and Jian Cheng. Q-vit: Fully differentiable quantization for vision transformer. *arXiv preprint arXiv:2201.07703*, 2022.

[205] Zhexin Li, Peisong Wang, Zhiyuan Wang, and Jian Cheng. Fixed-point quantization for vision transformer. In *Chin. Autom. Congr.*, pages 7282–7287. IEEE, 2021.

[206] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. Learn. Represent.*, 2019.

[207] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1–9, 2015.

[208] Zechun Liu, Kwang-Ting Cheng, Dong Huang, Eric P Xing, and Zhiqiang Shen. Nonuniform-to-uniform quantization: Towards accurate quantization via generalized straight-through estimation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4942–4952, 2022.

[209] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. Overcoming oscillations in quantization-aware training. In *Int. Conf. Mach. Learn.*, volume 162, pages 16318–16330, 2022.

[210] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. HAWQ: hessian aware quantization of neural networks with mixed-precision. In *Int. Conf. Comput. Vis.*, pages 293–302, 2019.

[211] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Int. Conf. Comput. Vis.*, pages 4852–4861, 2019.

[212] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized neural networks with a full-precision auxiliary module. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1488–1497, 2020.

[213] Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. In *Int. Conf. Learn. Represent.*, 2020.

[214] Dohyung Kim, Junghyup Lee, and Bumsub Ham. Distance-aware quantization. In *Int. Conf. Comput. Vis.*, pages 5271–5280, 2021.

[215] Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6448–6457, 2021.

[216] Longguang Wang, Xiaoyu Dong, Yingqian Wang, Li Liu, Wei An, and Yulan Guo. Learnable lookup table for neural network quantization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12423–12433, 2022.

[217] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards effective low-bitwidth convolutional neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7920–7928, 2018.

[218] Kohei Yamamoto. Learnable companding quantization for accurate low-bit neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5029–5038, 2021.

[219] Xiangning Chen, Cho-Jui Hsieh, and Boqing Gong. When vision transformers outperform resnets without pre-training or strong data augmentations. In *Int. Conf. Learn. Represent.*, 2022.

[220] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

[221] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3498–3505. IEEE, 2012.

[222] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conf. Comput. Vis. Graph. Image Process.*, pages 722–729. IEEE, 2008.

[223] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *Int. Conf. Learn. Represent.*, pages 1–23, 2019.

[224] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *Int. Conf. Mach. Learn.*, pages 10347–10357, 2021.

[225] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *Int. Conf. Mach. Learn.*, pages 4651–4664, 2021.

[226] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Adv. Neural Inform. Process. Syst.*, volume 28, pages 3123–3131, 2015.

[227] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. In *IEEE/ACM Int. Symp. Microarchitecture*, pages 811–824, 2020.

[228] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18(1):6869–6898, 2017.

[229] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *Adv. Neural Inform. Process. Syst. Worksh.*, pages 36–39, 2019.

[230] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Adv. Neural Inform. Process. Syst.*, volume 28, pages 1225–1233, 2015.

[231] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. In *Int. Conf. Learn. Represent.*, pages 1–19, 2021.

[232] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2074–2081, 2012.

[233] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Adv. Neural Inform. Process. Syst.*, volume 20, pages 1177–1184, 2007.

[234] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *Int. Conf. Learn. Represent.*, pages 1–11, 2020.

[235] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Adv. Neural Inform. Process. Syst.*, pages 344–352, 2017.

[236] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.

[237] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton Van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1963–1970, 2014.

[238] Xiangxiang Chu, Zhi Tian, Yuqing Wang, Bo Zhang, Haibing Ren, Xiaolin Wei, Huaxia Xia, and Chunhua Shen. Twins: Revisiting the design of spatial attention in vision transformers. In *Adv. Neural Inform. Process. Syst.*, pages 9355–9366, 2021.

[239] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *Int. Conf. Learn. Represent.*, pages 1–19, 2021.

[240] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. Long-short transformer: Efficient transformers for language and vision. In *Adv. Neural Inform. Process. Syst.*, volume 34, pages 17723–17736, 2021.

[241] Hongyu Ren, Hanjun Dai, Zihang Dai, Mengjiao Yang, Jure Leskovec, Dale Schuurmans, and Bo Dai. Combiner: Full attention transformer with sparse computation cost. In *Adv. Neural Inform. Process. Syst.*, volume 34, pages 22470–22482, 2021.

[242] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symp. Comput. Geom.*, pages 253–262, 2004.

[243] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(6):1092–1104, 2011.

[244] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *Int. Conf. Learn. Represent.*, pages 1–17, 2018.

[245] Lu Hou and James T Kwok. Loss-aware weight quantization of deep networks. In *Int. Conf. Learn. Represent.*, pages 1–16, 2018.

[246] Yu Bai, Yu-Xiang Wang, and Edo Liberty. Proxquant: Quantized neural networks via proximal operators. In *Int. Conf. Learn. Represent.*, pages 1–19, 2019.

[247] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *Int. Conf. Learn. Represent.*, 2022.

[248] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Understanding and overcoming the challenges of efficient transformer quantization. In *Conf. Empir. Methods Nat. Lang. Process.*, pages 7947–7969, 2021.

[249] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *Int. Conf. Learn. Represent.*, 2023.

[250] Daniel Bolya and Judy Hoffman. Token merging for fast stable diffusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4598–4602, 2023.

[251] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Int. Conf. Comput. Vis.*, pages 1389–1397, 2017.

[252] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *MAPL*, pages 10–19, 2019.

[253] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Assoc. Adv. Artif. Intell.*, volume 34, pages 7432–7439, 2020.

[254] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[255] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Assoc. Comput. Linguist.*, pages 4791–4800, 2019.

[256] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Commun. ACM*, 64 (9):99–106, 2021.

[257] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

[258] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Conf. Empir. Methods Nat. Lang. Process.*, 2023. URL https://openreview.net/forum?id=hmOwOZWzYE.

[259] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023. URL https://lmsys.org/blog/2023-03-30-vicuna/.

[260] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Neural Inf. Process. Syst. Datasets Benchmarks Track*, 2023.

[261] Saleh Ashkboos, Ilia Markov, Elias Frantar, Tingxuan Zhong, Xincheng Wang, Jie Ren, Torsten Hoefler, and Dan Alistarh. Towards end-to-end 4-bit inference on generative large language models. *arXiv preprint arXiv:2310.09259*, 2023.

[262] Guibin Wang, YiSong Lin, and Wei Yi. Kernel fusion: An effective method for better power efficiency on multithreaded gpu. In *IEEE/ACM Int'l Conf. Int'l Conf. Cyber Phys. Soc. Comput. (CPSCom) Green Comput. Commun. (GreenCom)*, pages 344–350, 2010.

[263] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *Eur. Conf. Comput. Vis.*, pages 544–560, 2020.

[264] Ying Wang, Yadong Lu, and Tijmen Blankevoort. Differentiable joint pruning and quantization for hardware efficiency. In *Eur. Conf. Comput. Vis.*, pages 259–277, 2020.

[265] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[266] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *Int. Conf. Learn. Represent.*, 2024.

[267] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

[268] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

[269] James Liu, Guangxuan Xiao, Kai Li, Jason D Lee, Song Han, Tri Dao, and Tianle Cai. Bitdelta: Your fine-tune may only be worth one bit. *arXiv preprint arXiv:2402.10193*, 2024.

[270] Xiaozhe Yao and Ana Klimovic. Deltazip: Multi-tenant language model serving via delta compression. *arXiv preprint arXiv:2312.05215*, 2023.

[271] Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReloRA: High-rank training through low-rank updates. In *Int. Conf. Learn. Represent.*, 2024.

[272] Yongchang Hao, Yanshuai Cao, and Lili Mou. Flora: Low-rank adapters are secretly gradient compressors. In *Int. Conf. Mach. Learn.*, 2024.

[273] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *Int. Conf. Mach. Learn.*, pages 31094–31116, 2023.

[274] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[275] Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Roziere, Jacob Kahn, Shang-Wen Li, Wen tau Yih, Jason E Weston, and Xian Li. Branch-train-mix: Mixing expert LLMs into a mixture-of-experts LLM. In *First Conf. Lang. Model.*, 2024.

[276] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[277] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

[278] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

[279] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Adv. Neural Inform. Process. Syst.*, 2021.

[280] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

[281] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In *Adv. Neural Inform. Process. Syst.*, 2023.

[282] Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. CMMLU: Measuring massive multitask language understanding in Chinese. In *Find. Assoc. Comput. Linguist.*, pages 11260–11285, August 2024.

[283] OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023.

[284] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[285] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Adv. Neural Inform. Process. Syst.*, volume 36, pages 21558–21572. Curran Associates, Inc., 2023.

[286] Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Winter Conf. Appl. Comput. Vis.*, pages 1383–1392, 2024.

[287] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: KV cache compression in depth dimension for large language models. In *Adv. Neural Inform. Process. Syst.*, 2024.

[288] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Int. Conf. Learn. Represent.*, 2019.

[289] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *Int. Conf. Mach. Learn.*, pages 6682–6691. PMLR, 2020.