ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

ARTIFICIAL INTELLIGENCE

**MASTER THESIS**

IN

ARCHITECTURES AND PLATFORMS FOR ARTIFICIAL INTELLIGENCE

# Optimizing Small Language Models: An Experimental Investigation in Compressing Distilled LLaMA Architectures

**Supervisor:**
**Prof. Francesco Conti**

**Candidate:**
**Angelo Galavotti**

**Co-supervisor:**
**Luca Bompani**

**Session I**
**Academic Year 2024/2025**

*Alla migliore madre del mondo,*
*al miglior padre del mondo,*
*e al miglior fratello del mondo.*

# Abstract

Large Language Models have transformed our interaction with technology, yet their massive scale introduces major drawbacks. Their substantial energy consumption and water usage create environmental pressures, while cloud-based architectures limit user autonomy and enable pervasive data collection. Additionally, their resource-intensive nature leads to entirely different challenges for edge deployment, where memory constraints and computational limitations make standard LLMs impractical.

This thesis presents a comprehensive compression pipeline that systematically applies multiple optimization strategies to the compact LLaMA 3.2 1B Instruct model: depth pruning removes redundant transformer layers based on importance metrics, width pruning applies structured sparsity to attention matrices using the WANDA algorithm, Low-Rank Adaptation recovers performance while enabling task-specific fine-tuning, 4-bit quantization further reduces memory footprint through GPTQ, and Eigenspace Low-Rank Approximation provides training-free accuracy recovery.

The experimental evaluation demonstrates that integrating multiple optimization techniques can significantly decrease memory footprint while preserving core language functionality. LoRA adaptation proves particularly effective at recovering capabilities lost during aggressive pruning. However, compression beyond specific thresholds leads to rapid performance deterioration that recovery procedures cannot fully mitigate, revealing fundamental architectural limits in transformer compression.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

It is no secret that, in the last three years, *Large Language Models* (LLMs) have fundamentally transformed our relationship with technology. Their impact rivals the most significant innovations of the past century, such as the internet and the smartphone. When people contemplate *Artificial Intelligence* (AI) today, they immediately think of ChatGPT or Claude, which have seamlessly integrated into our daily routines. Yet these powerful tools come with significant concerns. Their development and operation consume vast amounts of energy and water resource: modern data centers supporting these models require extensive cooling systems and electricity consumption that can rival small cities.

The computational complexity of these systems necessitates cloud-based deployment, which not only amplifies their environmental footprint but also fundamentally restricts user autonomy. This cloud dependency creates a concerning power dynamic where users have limited control over their tools, while simultaneously enabling extensive data collection practices and potential surveillance mechanisms that would be impossible with local, user-controlled alternatives.

In this more conversational opening chapter we will briefly examine the environmental and social impact of LLMs while highlighting the growing imperative for efficient, locally-deployable models that democratize access without depleting our planet's resources. Afterwards, we will outline the scope of this project, which aims to explore the potential of compression techniques to make LLMs more efficient.

The future of AI depends not just on what these models can do, but how sustainably they can do it.

## 1.1 The Social Impact of LLMs

The widespread adoption of LLMs has created ripple effects across virtually every sector of society, fundamentally altering how we work, learn, and create. In education, these tools have sparked heated debates about academic integrity while simultaneously offering new possibilities for personalized learning and accessibility for students with disabilities [1]. The workplace has experienced perhaps the most dramatic shifts, with entire professions grappling with automation anxiety while others discover unprecedented productivity gains. Creative industries find themselves in particularly complex territory: writers, artists, and content creators must navigate between leveraging AI as a collaborative tool and protecting their intellectual property from being absorbed into training datasets without consent or compensation [2].

Perhaps what is most striking is how these models have democratized access to sophisticated capabilities that were once the exclusive domain of experts. A small business owner can now generate marketing copy that rivals professional agencies, students can receive tutoring in subjects where human expertise might be scarce, and non-programmers can write functional code with natural language instructions. Yet this democratization comes with a troubling caveat: it's entirely dependent on maintaining access to centralized, corporate-controlled systems. When OpenAI experiences an outage, millions of users worldwide suddenly lose access to tools they've integrated into their daily workflows. When pricing models change, entire business models built around AI assistance can become unsustainable overnight.

This dependency becomes even more concerning when we consider the data these systems collect. Every interaction, every query, every creative prompt may potentially become part of these companies' datasets, raising questions about privacy and intellectual property. As such, the need for transparency and user control over these systems has never been more urgent, and many believe that the future of AI must prioritize local, user-deployable models that empower individuals rather than centralizing power in the hands of a few corporations.

## 1.2 The Environmental Impact of LLMs

The computational demands of modern LLMs create an environmental footprint that grows exponentially with model size and usage. Training GPT-3, for instance, consumed an estimated 1,287 MWh of electricity, which is enough to power an average American home for over a century [3]. However, training represents only the tip of the iceberg; the real environmental cost lies in inference, where billions of daily queries across millions of users create a continuous drain on global energy resources. A recent study by Jegham et al. [5] estimates that OpenAI's GPT-4.5 requires 6.7 Wh of energy for a medium sized query (i.e. 100 input tokens, and outputting 300 tokens) to be processed. This figure grows to 20.5 Wh for a larger query (i.e. 1000 input tokens, and outputting 1000 tokens). To put this into perspective, this is equal to charging an average 40 Wh laptop battery to 50%. A graph comparing the energy consumption of different LLMs with different prompt sizes is shown in Figure 1.1.

Data centers housing these models consume approximately 1-2% of global electricity, a figure that's projected to reach 8% by 2030 if current trends continue [4]. The infrastructure supporting a single large-scale LLM requires thousands of high-performance GPUs running 24/7, each consuming as much power as several households.

Water consumption presents an equally pressing concern that receives far less attention. Modern data centers require extensive cooling systems, with some facilities consuming millions of gallons daily. According to their sustainability report [6], Google's water usage increased by 20% between 2021 and 2022, then by 17% from 2022 to 2023, and is largely attributed to AI inference operations [7]. In regions already facing water scarcity, this additional demand creates direct competition with human needs and agricultural requirements.

On the other hand, one has to keep in mind the embedded carbon cost of the hardware itself. Each GPU cluster supporting LLM operations represents significant emissions from manufacturing, shipping, and eventual disposal. The rapid pace of AI advancement drives frequent hardware upgrades, creating electronic waste streams that the recycling industry may struggle to process effectively.

These environmental costs scale directly with model size and usage frequency, leading to a fundamental tension between AI capabilities and sustainability.

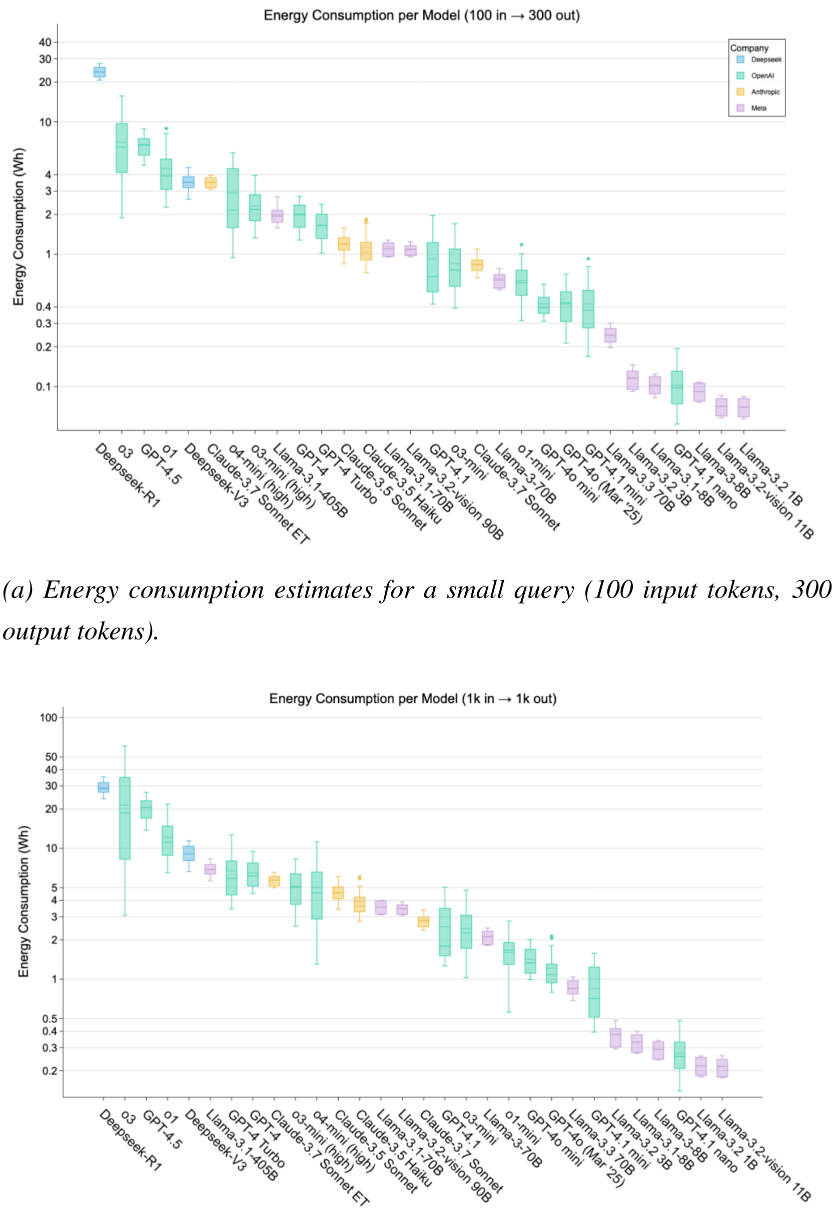
## 1.3 Scope

While we've examined several critical challenges facing LLMs, it's worth emphasizing that these models hold significant potential for positive impact. As such, what was outlined in the previous sections points instead toward an urgent need for alternatives to the current paradigm of massive, centralized LLMs. A promising solution lies in compression techniques that can dramatically reduce model size while preserving core functionality. Through compression, billion-parameter server-sized models can be scaled down to more manageable ones that run much more efficiently.

### 1.3.1 The Main Objective

Given the context, the objective of this project is to push the boundaries of memory efficiency for small-scale LLMs by applying a targeted suite of compression techniques. We begin with a distilled model, a 1B parameter variant of LLaMA 3 [28], which already represents a significantly reduced footprint compared to full-scale LLMs. From there, we explore and implement further optimizations, including depth and width pruning, *Low-Rank Adaptation* (LoRA), and quantization.

Pruning allows us to remove redundant layers or neurons from the network architecture, trimming excess capacity without substantial loss of capability. Quantization reduces the bit-width of model weights and activations, decreasing both



*(a) Energy consumption estimates for a small query (100 input tokens, 300 output tokens).*

*(b) Energy consumption estimates for a large query (1000 input tokens, 1000 output tokens).*

*Figure 1.1: Comparison of the estimated energy consumption of different LLMs for small and large queries. The data highlights the significant increase in energy requirements as query size grows. These graphs have been sourced from [5].*

memory usage and compute requirements. Meanwhile, LoRA introduces a lightweight, parameter-efficient training mechanism that reduces the cost of fine-tuning and adaptation without (necessarily) modifying the base model weights. In this way, the model can be fine-tuned on new tasks or domains with minimal overhead.

Memory efficiency represents the primary constraint in our target deployment scenario, which consists of low-power RISC-V edge devices with severely limited, albeit exceptionally fast, memory resources (further details on representative target hardware can be found in Section 2.4). While specialized hardware addresses these constraints through dedicated neural processors and optimized memory hierarchies, this thesis pursues a complementary algorithmic approach by optimizing the models themselves. Crucially, reducing memory footprint simultaneously decreases computational complexity, providing dual benefits for resource-constrained deployment.

### 1.3.2 How Can Optimization Techniques Help?

Optimization techniques increase the efficiency of models, reducing per-query energy consumption and tackling the environmental, social, and infrastructural problems that come with large-scale LLMs. Smaller models can run on edge devices and other energy-efficient hardware, cutting down on electricity usage substantially. This efficiency boost also extends the useful life of older devices: hardware that might otherwise be considered obsolete can suddenly run modern LLMs, which helps reduce electronic waste and makes better use of existing resources.

There's also the autonomy angle. Compact models that run locally allows users to be independent from centralized servers when using AI. People can run LLMs right on their own devices without any internet connection, which means no data gets sent to third parties and there's no risk of surveillance. This approach supports decentralization and opens up AI access to areas with poor connectivity or limited economic resources.

In other words, optimization is a pathway toward environmentally sustainable, privacy-respecting, and widely accessible AI.

## 1.4 Document Structure

This thesis is organized into five chapters that progressively build from foundational concepts to practical implementation and evaluation.

Chapter 1 establishes the motivation for LLM compression by examining the environmental and social impacts of large-scale language models. It essentially outlines the main objective of the work presented in this thesis.

Chapter 2 provides the technical background, tracing the evolution from RNN-based models to Transformer architectures with focus on the attention mechanism. It introduces the LLaMA model family, examines representative target hardware constraints, and reviews related work in optimization techniques.

Chapter 3 presents the core methodology of the five-stage compression pipeline: depth pruning, width pruning using WANDA, LoRA adaptation, quantization through GPTQ, and EoRA compensation. The evaluation framework using WikiText-2 and TriviaQA benchmarks is also described.

Chapter 4 analyzes experimental results across different model configurations. Key findings include the effectiveness of recovery methods and identification of performance thresholds.

Chapter 5 concludes the work by summarizing the key insights from the experiments while outlining future directions, including knowledge distillation integration and broader architectural compatibility.

# Chapter 2

# Background and Related Work

Understanding the methodology behind this thesis requires first examining how Transformer architectures and Language Models have evolved. This chapter explores that evolution and briefly introduces the LLaMA family as it will be relevant to this project (see Chapter 3). The discussion then turns to the hardware limitations which influenced design decisions in this context. Finally, the chapter reviews research and literature on LLM optimization techniques relevant to this work.

## 2.1 A Brief Overview of Early Language Models

Before the emergence of the Transformer architecture, language models predominantly relied on *Recurrent Neural Networks* (RNNs). These networks extend the *Multi-Layer Perceptron* (MLP) by incorporating cyclic connections within their architecture to create recurrent circuits. Such design enables the model to maintain a form of memory, allowing it to consider previous inputs when generating predictions and thereby capturing long-range dependencies inherent in sequential data such as text.

Despite their theoretical advantages, RNNs suffer from a critical limitation known as the vanishing gradient problem [10]. During backpropagation, gradients from earlier time steps decay exponentially as they flow backward through the network layers. This degradation severely hampers the network's ability to be trained effectively, as the influence of distant past information becomes negligible in the parameter updates.

To address these limitations, Hochreiter and Schmidhuber introduced *Long Short-Term Memory* (LSTM) networks [11] [12], which revolutionized sequence modeling through their sophisticated gating mechanisms. LSTMs employ specialized memory cells designed to selectively retain, update, and output information across extended sequences. The architecture incorporates three fundamental gate types that regulate information flow:

- The *input gate* determines the extent to which new information from the current input should be incorporated into the cell state. It evaluates the relevance of incoming data and controls its integration with existing memory.

- The *forget gate* governs the retention of information from previous time steps, deciding which aspects of the historical cell state remain relevant and should be preserved.

- Finally, the *output gate* regulates how much of the current cell state should be exposed to subsequent layers, effectively controlling what information propagates forward in the network.

All of these gates can be clearly observed in Figure 2.1. This gating mechanism allows LSTMs to maintain gradient flow over much longer sequences compared to vanilla RNNs. Thus, they can capture dependencies spanning hundreds of time steps, making them particularly effective for tasks requiring context understanding such as language modeling, machine translation, and text summarization.

While LSTMs significantly improved the ability to model sequential data, they still faced challenges in terms of parallelization, long-term context understanding, and training efficiency especially when compared to newer architectures such as Transformers [15] (see Section 2.2). Nevertheless, LSTMs are still widely used for various *natural language processing* (NLP) tasks, including text generation [13].

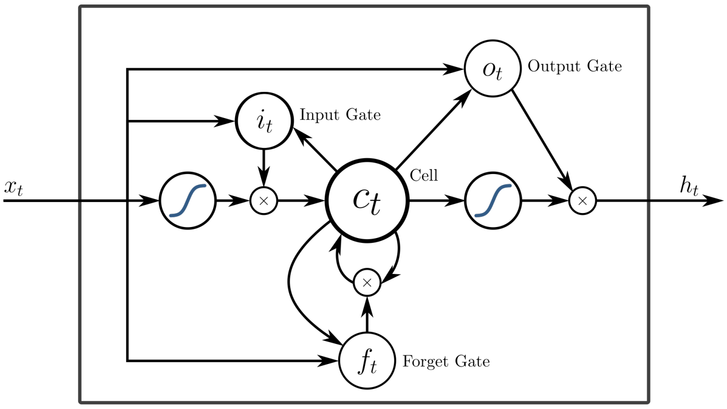


*Figure 2.1: Architecture of an LSTM cell showing the three gating mechanisms. The input gate (left) controls information incorporation, the forget gate (lower center) manages memory retention, and the output gate (upper right) regulates information flow to subsequent layers. The cell state (horizontal line) maintains long-term memory throughout the sequence. The image was retrieved from [12].*

### 2.1.1 Tokenization

Neural networks operate exclusively with numerical data, thus textual input needs to be converted into vector representations before being processed. This is achieved through tokenization, which segments raw text into discrete units called tokens that are then mapped to high-dimensional vector embeddings.

The tokenization process typically involves breaking text into subword units rather than complete words, which enables models to handle both common vocabulary and previously unseen terms through compositional understanding. Modern language models predominantly employ algorithms such as *Byte Pair Encoding* (BPE) [14], which iteratively merges frequently occurring character pairs to create an optimal vocabulary. Each resulting token corresponds to a learned vector embedding that captures semantic and syntactic properties, enabling the neural network to process linguistic information through mathematical operations.

## 2.2 The Transformer Architecture

The Transformer architecture, introduced by Vaswani et al. [15], fundamentally changed how sequence modeling is approached by entirely abandoning traditional explicit recurrent connections in favor of the attention mechanism, allowing to capture long-range dependencies. Moreover, instead of that processing sequences step-by-step like LSTMs, the Transformer can examine all elements in a sequence simultaneously, and thus it can be easily parallelized during training.

### 2.2.1 The Attention Mechanism

Attention allows a model to dynamically focus on different parts of the input sequence when generating each output token, similar to how humans selectively concentrate on relevant information when reading a book. This is achieved by directly comparing and relating any two positions in a sequence, regardless of their distance (i.e. self-attention). The behavior of the attention mechanism is illustrated in Figure 2.2.

As such, the information bottleneck created when sequence models compress an entire input sequence into a single fixed-size vector is partially dealt with. This bottleneck becomes particularly problematic for long sequences, where important information can be lost or diluted.

### 2.2.2 Scaled Dot-Product Attention

The attention mechanism is computed using three matrices derived from the input:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Each component serves a specific purpose:

- **Queries (Q)**: Generated by $Q = XW_Q$ where $X$ is the input and $W_Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$. Each query vector can be seen as asking “what information am I looking for?”.

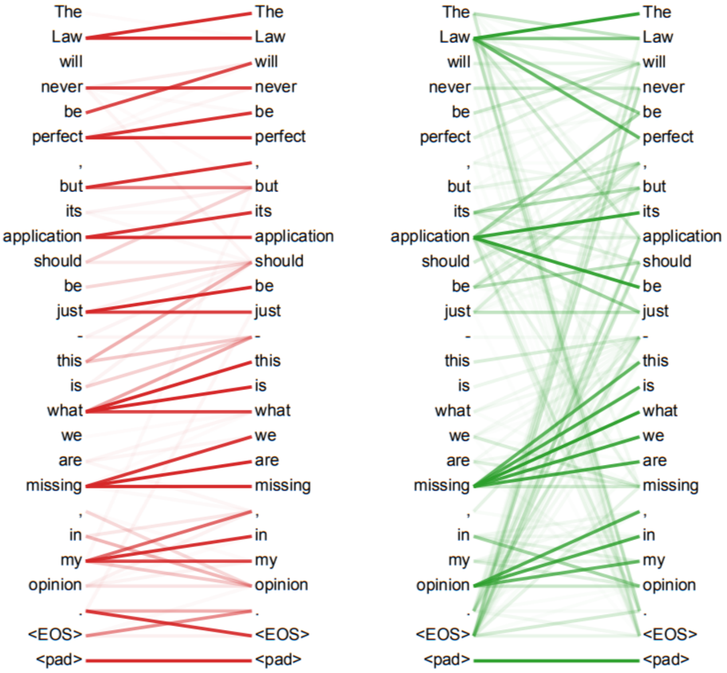


*Figure 2.2: Attention patterns from two different heads in a layer of a Transformer. The green visualization (right) shows distributed attention across multiple word pairs, while the red visualization (left) displays more focused, localized connections. Each line represents the attention weight between tokens, with darker lines indicating stronger attention, thus showing how individual attention heads specialize in capturing different linguistic relationships within the same sequence. The image was sourced from the appendix of [15].*

- **Keys (K)**: Generated by $K = XW_K$ where $W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$. Keys act as indexed labels for each position's content.
- **Values (V)**: Generated by $V = XW_V$ where $W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$. Values contain the actual information to be retrieved.

The attention computation proceeds in four steps:

1. **Similarity Computation**: $QK^T$ produces a $n \times n$ attention matrix where entry $(i, j)$ represents how much each token at position $i$ is correlated to a token at position $j$. The dot product naturally measures similarity between query and key vectors.
2. **Scaling**: Division by $\sqrt{d_k}$ (where $d_k$ is the dimensionality of the keys) prevents the dot products from becoming too large. Without scaling, large dot products push the softmax function into regions with extremely small gradients. For example, if $d_k = 512$, dot products could reach magnitudes of $\pm 20$ or more, causing softmax to output distributions close to one-hot vectors.
3. **Normalization**: The softmax function converts similarity scores into a probability distribution: softmax$(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$. This ensures attention weights sum to 1 and creates a differentiable selection mechanism.
4. **Weighted Aggregation**: The attention weights are applied to the value vectors, producing a weighted sum that represents the attended information for each position.

### 2.2.3 Multi-Head Attention

Rather than applying attention once to the full representation, the Transformer divides the input representation (i.e. $Q, K, V$) into $h$ blocks and applies the Scaled Dot Product Attention independently to each of them. Each of these independent blocks is called an attention head, and they capture different types of relationships simultaneously (some heads might track syntax, others semantics, positions, etc.), as shown in Figure 2.2. To achieve this, the input matrices $Q, K, V$ are projected $h$ times using different learned linear projections:

$$(QW_Q^i, KW_K^i, VW_V^i) \tag{2.2}$$

Given this, each head is computed as:

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \tag{2.3}$$

Each head uses different projection matrices $W_Q^i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_K^i \in \mathbb{R}^{d_{\text{model}} \times d_k}$, and $W_V^i \in \mathbb{R}^{d_{\text{model}} \times d_v}$, where typically $d_k = d_v = d_{\text{model}}/h$ for $h$ heads.

As such, the final equation for multi-head attention is:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O \tag{2.4}$$

where $W_O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ is the output projection matrix that transforms the concatenated multi-head output back to the original model dimension. This final projection is necessary because the concatenated heads have dimension $h \times d_v$, and must be projected back to $d_{\text{model}}$ to maintain dimensional consistency for subsequent layers.

Multi-head attention essentially gives the model multiple different perspective of the same input, and it is a vital system that contributes to make this architecture so powerful.

### 2.2.4 Positional Encodings

Since the attention mechanism computes similarity scores between every pair of tokens based solely on their content representations (i.e. it is permutation invariant), it treats the input as an unordered set rather than a sequence. This means that rearranging the input tokens would produce identical attention weights, making it impossible for the model to distinguish between different orderings of the same words. To address this limitation, the Transformer requires explicit positional information.

The original work uses sinusoidal positional encodings:

$$PE_{(\mathrm{pos},2i)} = \sin(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}) \tag{2.5}$$

$$PE_{(\mathrm{pos},2i+1)} = \cos(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}) \tag{2.6}$$

These encodings create unique representations for each position that enable the model to learn the relative location of tokens through linear combinations of the sinusoidal functions.

### 2.2.5 Encoder-Decoder Transformers

Transformer-based models generate text by predicting the next token in a sequence, conditioned either on both an external input and previously generated tokens (encoder-decoder models) or solely on the sequence generated so far (decoder-only models). The former is what the original Transformer architecture [15] follows.

- The **encoder** processes the full input sequence in parallel, producing a series of contextual embeddings that capture the meaning and structure of the input tokens.
- The **decoder** generates the output sequence token by token, using the embeddings from the encoder.

This design enables the model to generate outputs that are grounded in the input sequence, rather than generating new text from scratch. Thus, such design is well-suited for sequence-to-sequence tasks such as machine translation.

### 2.2.6 Decoder-Only Transformers

In decoder-only Transformers, there is no separate encoder module: the model predicts each token based only on the previously generated tokens in an auto-regressive manner. This means that the model outputs one token at a time, appends it to the input, and repeats until a stopping condition is met.

These architectures are simpler and more scalable, and they are employed in large language models such as GPT [8] and, of course, LLaMA [16].

Decoder-only Transformers are ideal for open-ended text generation and other tasks where the output is not conditioned on a separate input sequence. Figure 2.3 shows a comparison of encoder-decoder and decoder-only Transformer architectures.

## 2.3 The LLaMA Family of Models

The *Large Language Model Meta AI* (LLaMA) family [16], developed by Meta AI, comprises a series of Transformer-based auto-regressive language models designed as competitors to OpenAI's GPT model family. The original LLaMA models were introduced in early 2023, followed by LLaMA 2 [17] later that year and LLaMA 3 [18] in 2024.

The models vary in the number of parameters and capabilities, with the newer versions offering improved performance over the older ones. In particular, with LLaMA 3, Meta introduced models ranging from 8B to 405B parameters with major upgrades in both training scale and performance over LLaMA 2, while approaching results to OpenAI's GPT-4 [9] [18].

The biggest strength of LLaMA, however, lies in its open-weight release strategy. Unlike most proprietary models, Meta provides access to the model weights under a research-friendly license, enabling the broader research community and industry practitioners to fine-tune, deploy and experiment with large-scale language models without relying on closed systems. This openness has led to a proliferation of derivative models, and thus to the project central to this thesis.

## 2.4 Representative Target Hardware

The hardware constraints of an ideal target deployment scenario provide an additional compelling motivation for this compression work, beyond the environmental and social concerns outlined in Chapter 1. To illustrate the severity of

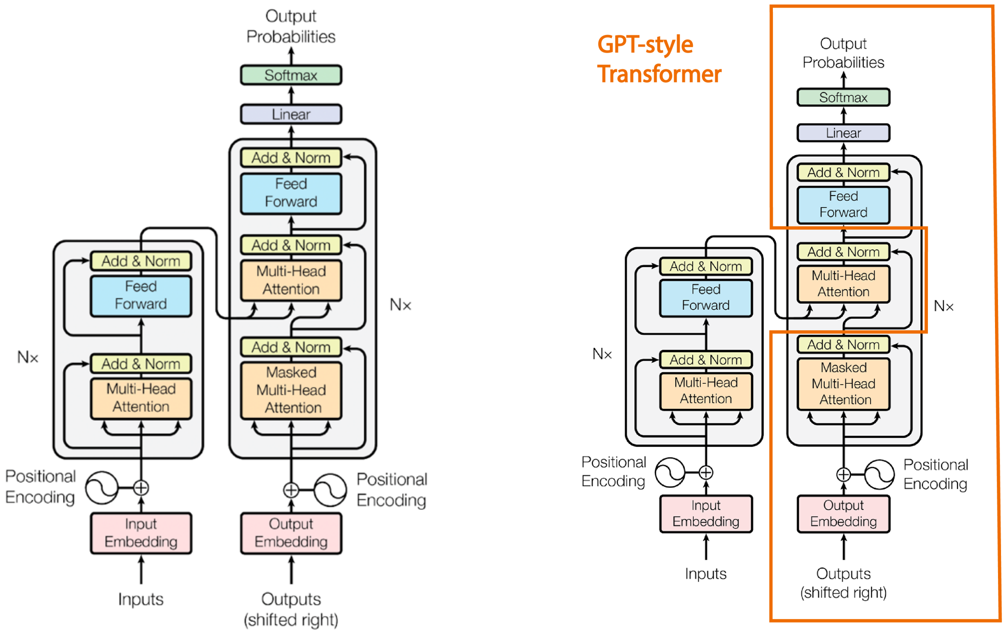


*Figure 2.3: On the left, a standard Transformer architecture, which consists of an encoder and a decoder. On the right, we can observe the same architecture, while highlighting only the layers used by a GPT-style model (i.e. a decoder-only Transformer). The main difference is that the decoder does not attend to the encoder's output, allowing for auto-regressive generation. The image is a modified version of the one found in Vaswani et al. research paper [15].*

these constraints, Siracusa [19] serves as a representative example of the type of deployment platform this compression pipeline aims to support. This heterogeneous RISC-V *System-on-Chip* (SoC) fabricated in 16 nm CMOS technology exemplifies the stringent resource limitations typical of edge devices designed for *Extended Reality* (XR) applications and similar embedded AI scenarios. While this project does not involve actual deployment or testing on this specific hardware, the severe memory limitations and computational constraints characteristic of such platforms make LLM compression not merely an environmental imperative, but a fundamental requirement for any practical edge deployment. A visual representation of Siracusa's architecture can be found in Figure 2.4.

Siracusa's memory hierarchy presents the primary constraint for the optimization efforts. The system features a three-tier memory organization: 256 KiB of L1 *Tightly-Coupled Data Memory* (TCDM) SRAM organized into 16 word-interleaved banks, 4 MiB of SRAM tile memory, and 4 MiB of *magnetoresistive memory* (MRAM) for weight storage, totaling 8.5 MiB. The memory hierarchy uses specialized allocation: MRAM stores weights while tile memory handles activations. However, it may be obvious to the reader that this amount of storage is rather limiting: exceeding the 4 MiB weight memory capacity forces reliance on slower off-chip transfers that can increase inference latency by orders of magnitude and considerably raise energy consumption for memory access.

The computational architecture features an octa-core RISC-V cluster paired with the N-EUREKA neural processing engine, which provides substantial parallel processing capability with peak performance reaching 1.95 TOp/s. However, this performance is only achievable when models fit entirely within the on-chip memory hierarchy. The 92 Gbit/s bandwidth between MRAM and the neural engine enables efficient weight streaming, but only when the model's memory footprint aligns with the available storage capacity.

These specifications illustrate the type of resource constraints that drive the need for aggressive compression techniques in edge AI deployment scenarios, and the compression pipeline developed in this work represents a promising approach toward achieving the level of optimization required for practical deployment on

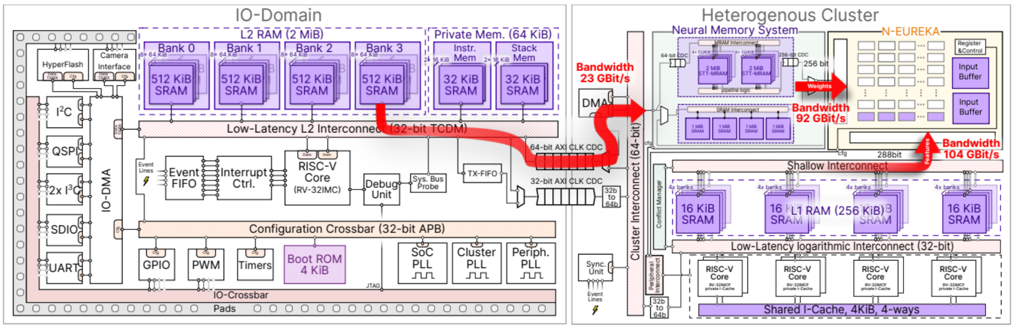


*Figure 2.4: A visual representation of Siracusa's architecture. On the upper right we can see the N-EUREKA accelerator, while on the lower right the RISC-V cluster is visible. This image was sourced from the original paper [19].*

such platforms.

## 2.5 Related Work

This section examines some techniques that have proven effective for compressing large language models, providing an overview of the current available research and inspiration for novel methods that could extend the project's scope in the future (further discussed in Section 5.1).

### 2.5.1 Knowledge Distillation

Knowledge distillation [20] has emerged as a fundamental technique for compressing pre-trained language models by transferring knowledge from a large "teacher" model to a smaller "student" model. In particular, the core principle involves training the student to match not only the ground truth labels but also the soft predictions of the teacher model, in order to capture richer information about the decision boundaries learned by the larger network.

Traditional distillation approaches can be categorized into task-specific and task-agnostic variants. Task-specific distillation first fine-tunes the teacher model on a particular downstream task before distillation, while task-agnostic distillation maintains a general-purpose student that can be efficiently adapted to various tasks.

A significant challenge in conventional distillation arises from the substantial capacity gap between teacher and student models. When the student model has significantly fewer parameters than the teacher, the prediction discrepancy can become too large for effective knowledge transfer, particularly over massive amounts of open-domain training data. This limitation has motivated more sophisticated initialization and training strategies.

*Homotopic Distillation* (HomoDistil) [21] addresses this challenge through an innovative approach that combines iterative pruning with distillation. Rather than initializing a small student model from scratch, HomoDistil begins with the full teacher model and gradually prunes neurons while simultaneously distilling knowledge. This maintains a small prediction discrepancy throughout training, ensuring effective knowledge transfer even at high compression ratios.

The key insight of HomoDistil is that starting from the teacher model and iteratively removing the least important components prevents the dramatic performance drops typically seen with aggressive compression. At each training iteration, the method computes importance scores for individual neurons and removes those with minimal impact on the loss function. This pruning strategy is very similar to one adopted in this document for removing whole layers (i.e. depth pruning), as shown in Section 3.3.1. The gradual reduction maintains network connectivity and allows the distillation loss to effectively guide the compression process.

### 2.5.2 Pruning Approaches

Pruning is an optimization technique that reduces model complexity by systematically removing network components. The approach operates through two primary strategies: width pruning, which removes individual neurons or layers within Transformer blocks, and depth pruning, which eliminates entire submodules. Given their central importance to this work, these techniques are examined in detail in Section 3.3.

Nevertheless, it is worth reviewing relevant work in this area to position our contribution within the broader research landscape. *Two-Stage Structured Pruning* (2SSP) [22] combines width and depth pruning optimally using a two stage approach. This method applies width pruning to feed-forward networks in the first stage, followed by depth pruning of attention modules in the second stage.

For width pruning, importance scores are computed based on the magnitude of neuron activations across calibration samples, which are then used to remove entire neurons. On the other hand, the second stage targets attention modules which are iteratively removed based on their impact on model perplexity. According to the paper [22], this coarser-grained approach proves effective because attention modules often exhibit high redundancy.

A critical innovation in 2SSP is the automatic balancing of sparsity between the two stages. The method uses an adaptive formula that considers the relative sizes of feed-forward and attention components to determine optimal allocation of parameter reduction across stages. This ensures that neither component becomes a bottleneck while maintaining overall performance.

While some similarities can be found, the work of this thesis differs from the pruning approach adopted in 2SSP in two fundamental ways.

- While 2SSP focuses on feed-forward network components, the approach presented in this thesis also targets the attention weight matrices.

- 2SSP removes complete rows and columns by slicing weight matrices rather than using ratio-based structural pruning (see Section 3.3.2).

Another promising approach to structured weight removal is demonstrated by *Sheared LLaMA* [23], which takes a fundamentally different perspective by viewing pruning as an initialization step rather than a final compression technique. This method employs targeted structured pruning to compress LLaMA2-7B down to 1.3B and 2.7B parameter models, followed by substantial continued pre-training to recover performance.

In particular, Sheared LLaMA combines two techniques: first, a constrained optimization approach that prunes models to precisely match target architectures (derived from existing well-optimized models), and second, dynamic batch loading that adjusts training data composition based on domain-specific loss recovery rates. This addresses a critical observation that pruned models retain different levels of knowledge across domains (e.g. maintaining better performance on code repositories while losing more capability on natural language text).

# Chapter 3

# Methodology and Implementation

Having covered the importance of LLM optimization and the technical architecture of Transformers in previous chapters, we now turn to the practical challenge of model compression.

The following sections begin with details on the preliminary experiments that revealed crucial insights about architectural redundancy in Transformer models and shaped the main compression strategy. It then presents what is possibly the heart of this research: a multi-stage pipeline that combines several optimization techniques in a structured sequence. Each stage is examined with both theoretical foundations and key implementation considerations. Finally, the evaluation framework is outlined along with the chosen metrics and benchmarks.

## 3.1 Preliminary Research: Franken-LLaMA

Before embarking on systematic compression techniques for our target 1B parameter LLaMA model, preliminary research was conducted to understand the behavior of Transformer architectures under structural changes. This exploratory phase consisted on the "Franken-LLaMA" project [24], which involved experimenting with architectural modifications on the larger LLaMA2-7B-Chat model [17] to gain a first insight into which components of the Transformer architecture are most critical for maintaining model performance.

The approach centered on modifying the standard Transformer execution flow by selectively including, excluding, or repeating attention blocks within the 32-layer architecture. Implementation was carried out using PyTorch [25], with modifications to Hugging Face's transformers library [26] to enable dynamic layer skipping and repetition at runtime. The repetition strategy was particularly attractive as it could theoretically reduce memory footprint by reusing the same layer weights multiple times rather than storing distinct parameters for each position, and aligned directly with the target hardware constraints outlined in Section 2.4.

25 different layer configurations were tested, each of which was evaluated through qualitative text generation tasks and quantitative assessment on the HellaSwag dataset [27]. The results revealed that conservative modifications often maintained reasonable performance while reducing computational overhead. In particular, skipping layers towards the middle-end of the model rather than the beginning or final layers resulted in lower output quality degradation. For instance, the configuration that skipped layers 23 to 27 achieved a HellaSwag score of 0.38 compared to the baseline's 0.34 (the baseline consisted on the unmodified LLaMA 2 model), suggesting that certain middle layers may contribute less to final performance than expected.

However, more aggressive modifications typically led to a much more severe degradation in output quality. Configurations involving extensive layer repetition or using only sparse layer selections often produced incoherent text with non-ASCII characters and semantic breakdown. This behavior indicated that while some redundancy exists in the Transformer architecture, there are clear limits to how extensively the model can be modified before fundamental language capabilities deteriorate.

Results from this exploratory phase informed the subsequent approach to systematic compression, as they revealed that strategic layer removal could maintain acceptable quality levels while reducing computational overhead, suggesting that pruning techniques might offer promising avenues for optimization. At the same time, Franken-LLaMA proved that layer repetition was not a viable strategy and caused heavy performance degradation.

## 3.2 An Overview of the Pipeline

The core result of this research is the compression pipeline, which implements a sequential approach that combines multiple optimization techniques in a carefully orchestrated manner. As previously mentioned, the target model for these optimizations is LLaMA 3.2 1B, a distilled version of the larger LLaMA 3.2 model that has already undergone some level of pruning during its creation, according to the model documentation on Hugging Face [28].

The choice of this particular model as the starting point is both strategic and practical. While incorporating distillation directly into the pipeline would have been ideal given its effectiveness as a compression technique [21], the computational requirements make it rather prohibitive: in terms of complexity, distillation is essentially equivalent to training a model from scratch, thus requiring extensive GPU resources that far exceed the computational budget available for the project. In light of this, an existing pre-distilled model was utilized instead which already provides a rather compact foundation.

Following the approach established in the preliminary experiments (Section 3.1), the pipeline is implemented using PyTorch [25] for model modifications and Hugging Face's transformers library [26] for model loading and inference. The majority of configurations were developed using the Instruct version of LLaMA 3.2 1B [29], as these instruction-tuned models are specifically optimized for dialogue and question-answering tasks that constitute the primary evaluation benchmarks in this work (see Section 3.6), thereby minimizing the need for additional prompt engineering during testing phases. Nevertheless, the compatibility with the original model is kept, and the framework can produce configurations built upon the vanilla LLaMA 3.2 1B variant [28] by specifying a flag before execution.

The compression pipeline follows a five-stage progression, with each stage building upon the previous one. This particular ordering was chosen based on the complementary nature of these techniques and their relative impact on model structure.

1. The first stage implements **depth-wise pruning** (Section 3.3.1), where entire Transformer layers are removed based on importance metrics computed during a calibration phase. This coarse-grained approach eliminates redundant blocks, and ultimately redefines the skeleton of the architecture.

2. **Width-wise pruning** follows as the second stage (Section 3.3.2), applying the WANDA algorithm [31] to remove less critical weights within the remaining layers. This is a finer-grained approach compared to depth pruning, as it operates at the parameter level.

3. The third stage introduces ***Low-Rank Adaptation*** (LoRA) [33] (Section 3.4) to recover performance lost during the pruning phases, while also allowing for fine-tuning on specific downstream tasks.

4. The fourth stage applies 4-bit **quantization** using GPTQ [36] (Section 3.5), reducing the memory footprint of individual parameters.

5. The final stage produces and applies an ***Eigenspace Low-Rank Approximation*** (EoRA) [40] adapter (Section 3.5.1) step to improve the performance of the quantized model.

Detailed descriptions of each stage are provided in the following sections. Each step yields an intermediate model suitable for independent evaluation, except when the user specifies otherwise. Additionally, the pipeline features a comprehensive set of tuning options, such as the ability to skip specific stages or experiment with different parameter configurations without modifying the core implementation. This allows to conduct ablation studies more easily and adapt the framework to specific hardware constraints.

## 3.3 Pruning Techniques

Pruning reduces neural network complexity by removing components according to importance criteria. The technique operates through two distinct main approaches that differ in their granularity and organization:

- **Unstructured pruning** eliminates individual weights throughout the network randomly or based on an importance criteria such as magnitude.

- **Structured pruning** removes entire architectural units such as groups of neurons, attention heads, or complete layers.

Additionally, they can be further categorized based on the spatial dimension of the operation. In particular:

- **Width pruning** targets components within layers, such as individual neurons inside an attention block.

- **Depth pruning** eliminates entire layers or Transformer blocks from the architecture. It is generally regarded as a form of structured pruning.

A visual comparison between these two pruning techniques is shown in Figure 3.1.

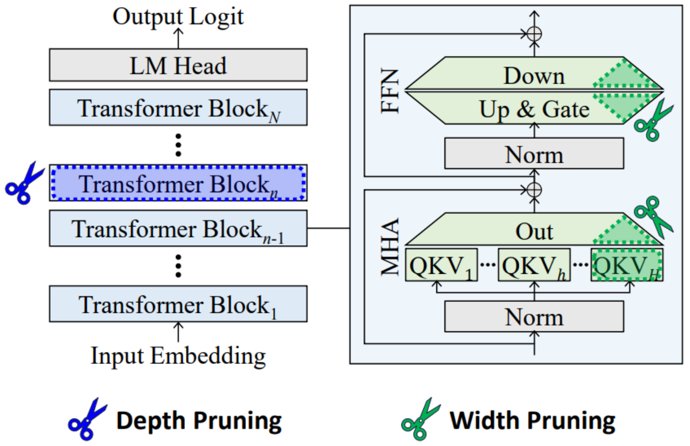


*Figure 3.1: Depth pruning removes the single Transformer layers (left), while width pruning removes single neurons from the weight matrices (right). The image was retrieved from [30].*

### 3.3.1 Depth Pruning

Regarding depth pruning, the approach used in this project eliminates entire attention blocks following the methodology established by Kim et al. in their Shortened LLaMA work [30], in which they define three metrics to determine the importance of a Transformer block:

- **Magnitude-based importance**: computes L1 norms of weights within each block, assuming that layers with larger weight magnitudes contribute more significantly to model output.
- **Taylor expansion importance**: leverages gradient information to estimate removal impact through the approximation

  $$L(W = 0) \approx L(W) + \nabla L \cdot (-W) \tag{3.1}$$

  where the gradient-weight product indicates layer significance based on how loss would change if the layer were removed.
- **Perplexity-based importance**: temporarily removes each layer and measures performance degradation on calibration data through perplexity [42], providing a direct empirical assessment of layer contribution to model quality (details on this metric are further explained in Section 3.6).

While the pipeline implementation incorporates all three importance ranking methods, in practice, perplexity served as the main importance metric during testing due to its direct correlation with model performance degradation. In view of this, all layers are ranked according to their computed importance scores, with layers receiving lower rankings considered less critical and prioritized for removal during the pruning process.

Following insights from prior work in Section 3.1 and Shortened-LLaMA [30], the implementation protects the first four and last two layers from removal, as these positions prove critical for maintaining performance.

The pruning implementation creates a new model architecture with reduced depth by systematically copying weights from the original model while skipping the least important layers. The process begins by modifying the model configuration to reflect the reduced number of layers, then establishes a mapping between original and pruned layer indices. This mapping accounts for the gaps created by removed layers, ensuring that retained layers maintain their relative positioning and connectivity.

The resulting pruned model maintains the same computational pattern as the original but with fewer sequential operations, directly reducing inference latency and, most importantly, memory requirements.

### 3.3.2 Width Pruning

The width pruning approach in this pipeline exclusively employs structured pruning techniques to maintain compatibility with modern hardware accelerators. Specifically, the implementation targets structured sparsity patterns supported by NVIDIA's sparse Tensor Cores in Ampere and Hopper architectures [32]. This sparsity pattern enables theoretical $2\times$ compute throughput compared to dense matrix multiplication while maintaining efficient memory access. In particular, unlike unstructured pruning that creates irregular sparse matrices, structured sparsity preserves regular patterns that can be compressed when loaded into VRAM, storing only non-zero values and compact index metadata while mapping directly to accelerator capabilities without additional software overhead. A visual representation of the advantage is shown in Figure 3.2.

*Weight AND Activation-based pruning* (WANDA) [31] serves as the core algorithm for implementing this structured approach. Originally developed for unstructured weight removal, WANDA's official implementation includes extensions for structured pruning patterns that align with the hardware requirements. The method leverages the observation that large language models develop outlier activations, which are emergent features with magnitudes significantly larger than typical hidden state values and are crucial for performance. Consequently, the WANDA importance metric for each weight $W_{ij}$ is defined as

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2 \tag{3.2}$$

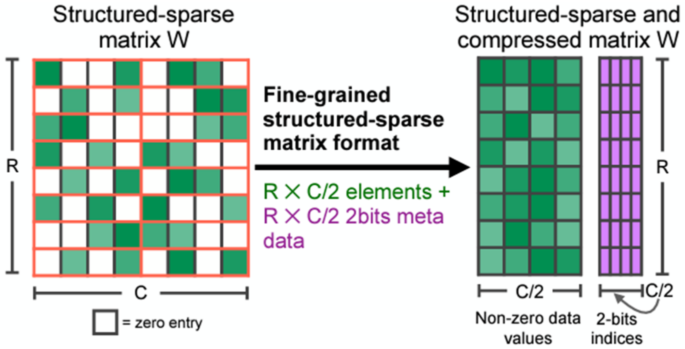

Structured-sparse matrix W
Structured-sparse and compressed matrix W
Fine-grained structured-sparse matrix format
R X C/2 elements +
R X C/2 2bits meta data
R
C
□ = zero entry
C/2
C/2
Non-zero data values
2-bits indices

*Figure 3.2: Visual representation of how the structured-sparse matrices are processed in NVIDIA's Ampere and Hopper GPUs. The image was sourced from [32].*

where $|W_{ij}|$ represents the absolute weight magnitude and $\|X_j\|_2$ evaluates the L2 norm of the $j$-th input feature across calibration samples. This formulation addresses a key limitation of traditional magnitude-based importance, which is typically used by width pruning approaches, by accounting for input activations, which play an equally important role in determining neuron outputs.

The structured pruning process each layer by organizing weights into groups of $M$ consecutive elements (i.e. consecutive columns of the same row of the weight matrix) and removing the $N$ weights with lowest importance scores while retaining the rest. This is applied comprehensively to all linear layers in the Transformer architecture, including both the feed-forward networks and the attention-related projection matrices $W_Q$, $W_K$, $W_V$, and $W_O$ (see Section 2.2.2). The activation statistics are collected by forward hooks during calibration, with each linear layer wrapped to compute the L2 norm of input activations across samples. After computing importance scores for each weight group, the implementation uses PyTorch's scatter operations to efficiently mark low-importance weights for removal. The algorithm completes pruning in a single forward pass using the C4 corpus [39] as calibration dataset.

## 3.4 Low-Rank Adaptation (LoRA)

*Low-Rank Adaptation* (LoRA) [33] represents a parameter-efficient fine-tuning technique that addresses the computational and memory constraints associated with full model retraining. Rather than updating all model parameters during adaptation, LoRA introduces trainable low-rank decomposition matrices that capture the essential changes needed for task-specific performance. Figure 3.3 provides an overview of this process.

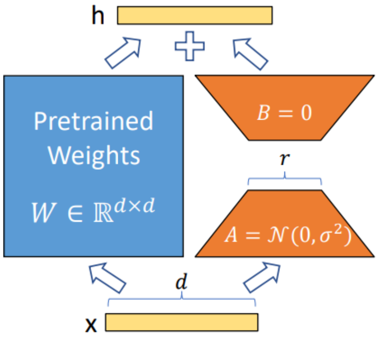

h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
$r$
$A = \mathcal{N}(0, \sigma^2)$
$d$
x

*Figure 3.3: A summary of the LoRA training process. We can see how the original pre-trained weights and the low-rank adapter are summed when the output activation is computed. The image was retrieved from the original LoRA paper [33].*

The fundamental insight behind LoRA stems from the hypothesis that weight updates during adaptation have a low intrinsic rank, even when the full rank of the weight matrices is very high. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d \times k}$, LoRA constrains the update by representing it through a low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA \tag{3.3}$$

where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$. During training, $W_0$ remains frozen while only $A$ and $B$ contain trainable parameters.

Consequently, the modified forward pass becomes:

$$h = W_0 X + \Delta W X = W_0 X + BAX \tag{3.4}$$

The matrices are initialized such that $A$ follows a random Gaussian distribution while $B$ is initialized to zero, ensuring that $\Delta W = BA$ starts at zero so that the model begins with its original pre-trained behavior.

LoRA offers substantial reductions in trainable parameters while maintaining comparable performance to full fine-tuning. Its linear design allows to merge the low-rank matrices with frozen weights during deployment, introducing no additional inference latency. According to the original paper [33], in Transformer architectures LoRA is most effective when applied to attention weights, particularly the query and value projection matrices ($W_q$ and $W_v$), with very low ranks ($r = 1$ or $r = 2$) often proving sufficient for downstream tasks.

Within the compression pipeline, LoRA serves as the third stage, following the pruning phases. This positioning is strategic: LoRA helps recover performance degraded by the aggressive structural modifications while simultaneously adapting the model for specific downstream tasks.

The implementation utilizes Hugging Face's *Parameter-Efficient Fine-Tuning* (PEFT) library [34], which provides optimized LoRA integration with Hugging Face's transformer models. The configuration specifically targets the four attention projection matrices: query ($W_Q$), key ($W_K$), value ($W_V$), and output ($W_O$) projections (see Section 2.2.2) with a rank of 8 and dropout rate of 0.1. Training employs mixed-precision optimization with gradient accumulation to maintain computational efficiency while adapting to the reduced parameter space.

## 3.5 Quantization

Quantization represents a fundamental compression technique that maps continuous floating-point values to a discrete, finite set of representations, typically using low precision formats such as 8-bit or 4-bit integers. This approach is widely used on architectures such as Convolutional Neural Networks [35], as these models are often deployed on edge devices and quantization offers a substantial memory reduction with controlled impact performance.

The quantization process follows the mapping:

$$\text{quant}(x) = \text{round}\left(\frac{x - z}{s}\right) \tag{3.5}$$

where $x$ represents the original floating-point value, $s$ denotes the scale factor and $z$ is the zero-point offset. Subtracting the offset and dividing by the scale normalizes the floating-point value to the quantization grid, so that it is accurately represented within the discrete range of the target precision format.

However, this straightforward round-to-nearest approach often results in significant accuracy degradation. The GPTQ technique proposed by Frantar et al. [36] addresses these limitations through a sophisticated post-training quantization algorithm that leverages second-order information from the Hessian matrix of the layer's reconstruction error to minimize quantization impact. Building upon the *Optimal Brain Quantization* (OBQ) framework [37], GPTQ formulates quantization as a layer-wise optimization problem where for each linear layer with weight matrix $W$ and calibration inputs $X$, the algorithm seeks quantized weights $\hat{W}$ that minimize the reconstruction error:

$$\arg\min_{\hat{W}} ||WX - \hat{W}X||_2^2 \tag{3.6}$$

The core innovation of GPTQ lies in its three-step algorithmic approach that enables scalability to billion-parameter models.

The first step is based on the idea that arbitrary quantization order suffices. Unlike OBQ, which greedily selects weights based on quantization error, GPTQ demonstrates that quantizing weights in any fixed order yields comparable results for large, heavily-parametrized layers. This insight allows all rows of the weight matrix to be quantized in the same column order, reducing the overall computational complexity.

The second step consists on the implementation of lazy batch updates. To address memory bandwidth bottlenecks, GPTQ processes weights in blocks of $B = 128$ columns simultaneously. The algorithm exploits the observation that quantization decisions for column $i$ are only affected by updates to that specific column, enabling "lazy batching" where updates are accumulated within blocks before global application:

$$\delta_F = -(w_Q - \text{quant}(w_Q))([H_F^{-1}]_{QQ})^{-1}(H_F^{-1})_{:,Q} \tag{3.7}$$

$$H_{-Q}^{-1} = \left[H^{-1} - H^{-1}_{:,Q}([H^{-1}]_{QQ})^{-1}H^{-1}_{Q,:}\right]_{-Q} \tag{3.8}$$

where $Q$ denotes the set of quantized weight indices within the current block, $F$ represents remaining full-precision weights, $w_q$ denotes a weight $\in Q$, $\delta_F$ is the compensation update applied to unquantized weights to minimize reconstruction error, and $H_{-Q}^{-1}$ is the updated inverse Hessian matrix with rows and columns corresponding to quantized weights $Q$ removed.

The final step is the Cholesky reformulation. For numerical stability at scale, GPTQ precomputes Hessian inverse information using Cholesky decomposition rather than iterative matrix updates. The Hessian matrix is computed as:

$$H = 2XX^T + \lambda I \tag{3.9}$$

where $\lambda$ represents a damping factor ($1\%$ of average diagonal value) for numerical stability. The Cholesky decomposition $H^{-1} = LL^T$ provides numerically stable access to required matrix rows while avoiding the accumulation of errors from repeated matrix inversions that plagued the older approaches when applied to large models.

The quantization procedure processes each layer independently, iterating through weight columns while maintaining compensation updates to remaining unquantized weights, and is applied to the attention matrices and linear layers of the Transformer.

The compression pipeline places quantization as the fourth stage, applied after pruning and LoRA adaptation phases, to strategically leverage the reduced parameter count from pruning. For the practical implementation, the GPTQ-Model toolkit [38] was employed, which provides optimized implementations of the GPTQ algorithm with support for various model architectures and quantization configurations. This toolkit handles the complex numerical computations and memory management required for large-scale quantization, allowing focus on the integration aspects within the broader compression pipeline. The quantization process utilizes 4-bit precision with a group size of 128, which means that groups of 128 consecutive weights share the same quantization parameters (scale and zero-point).

### 3.5.1 Eigenspace Low-Rank Approximation (EoRA)

Following quantization, *Eigenspace Low-Rank Approximation* (EoRA) [40] can be optionally applied to recover accuracy lost during compression. EoRA is a rather novel technique which provides a training-free method to enhance performance of compressed models through low-rank matrix compensation by projecting compression errors into task-specific eigenspaces. Figure 3.4 summarises the process in a schematic way.

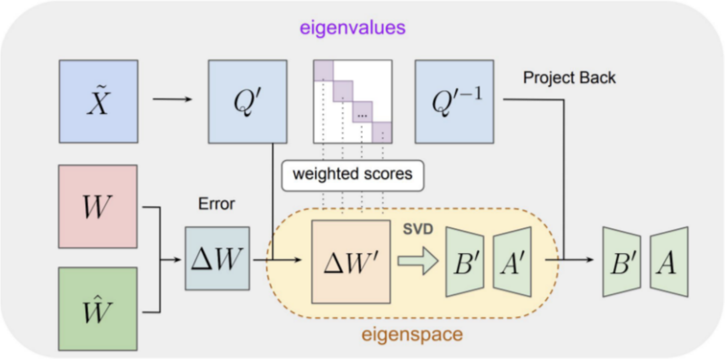


*Figure 3.4: The image summarizes the EoRA process in a schematic way, courtesy of [40].*

Despite their similar names, EoRA differs fundamentally from Low-Rank Adaptation. While LoRA adapts pre-trained models to new tasks through gradient-based optimization of low-rank matrices, EoRA focuses on compensating compression-induced errors without training. Nevertheless, both methods employ additive low-rank corrections to weight matrices which are formulated as adapters.

As noted earlier, EoRA projects compression error into the eigenspace of layer-wise input activations before applying singular value decomposition. For a layer with original weights $W$ and compressed weights $\hat{W}$, compression error is $\Delta W = W - \hat{W}$. Rather than directly decomposing this error, EoRA first computes the eigendecomposition $\tilde{X}\tilde{X}^T = Q\Lambda Q^T$, where $\tilde{X}$ represents average input activations over the calibration dataset. This is done in order to derive the eigenspace projection matrix $Q$, whose columns are the eigenvectors, and $\Lambda$, which is a diagonal matrix containing the corresponding eigenvalues.

The projection of the compression error into the eigenspace is then computed using the transformation matrix $Q' = Q\sqrt{\Lambda}$:

$$\Delta W' = \Delta W \cdot Q' \tag{3.10}$$

This eigenspace projection amplifies compression errors along directions of high activation variance (i.e. high eigenvalues) while diminishing those along low-variance directions, causing the subsequent SVD operation to naturally allocate more of its limited representational capacity to approximating the amplified, task-relevant error components.

Finally, the projected error $\Delta W'$ is decomposed using SVD to obtain $\Delta W' \approx B'A'$, where $B'$ and $A'$ are low-rank matrices. The final compensation matrices are obtained by back-projecting to the original space: $A = A'(Q')^{-1}$.

Similarly to LoRA, the forward pass on the compensated model becomes:

$$h = \hat{W}X + \Delta W'(Q')^{-1}X = \hat{W}X + B'A'(Q')^{-1}X = \hat{W}X + B'AX \tag{3.11}$$

Implementation utilizes the GPTQModel toolkit [38], which integrates EoRA compensation with existing quantized models. In the default configuration of the pipeline, the EoRA calibration step uses 64 samples from the C4 dataset [39]. This minimal calibration requirement represents one of the key advantages of this approach. Additionally, the adapter is generated using a rank of 32. Although the original paper employed a rank of 128 [40], the GPTQModel documentation indicates that larger ranks can cause overfitting while obviously consuming more memory [38]. This issue may be particularly pronounced for smaller 1B models such as the one adopted in this work.

## 3.6 Evaluation

The evaluation system employs two distinct benchmarking datasets to assess model performance across different dimensions of language understanding.

WikiText-2 [41] serves as the primary benchmark for measuring language modeling capability through perplexity computation [42], while TriviaQA [43] evaluates both reading comprehension and factual knowledge through question-answering tasks.

The implementation leverages a custom evaluation framework which automatically detects model types and applies appropriate loading procedures.

### 3.6.1 WikiText-2 Perplexity Evaluation

WikiText-2 [41] represents a collection of high-quality Wikipedia articles that serves as a standard benchmark for language modeling tasks. The dataset consists of over 2 million tokens extracted from verified Wikipedia articles, providing a diverse corpus of well-structured text that spans multiple domains and writing styles.

Perplexity serves as the fundamental metric for language model evaluation, measuring how well a model predicts the next token in a sequence. Mathematically, perplexity is defined as the exponential of the cross-entropy loss:

$$\text{Perplexity} = \exp\left(\frac{1}{N}\sum_{i=1}^{N} -\log P(w_i|w_1, w_2, \ldots, w_{i-1})\right) \tag{3.12}$$

where $N$ represents the total number of tokens, $w_i$ denotes the $i$-th token, and $P(w_i|w_1, w_2, \ldots, w_{i-1})$ represents the model's predicted probability for the correct next token given the preceding context. Lower perplexity values indicate better language modeling performance, with the metric providing intuitive interpretation: a perplexity of $X$ means the model is, on average, as confused as if it had to choose uniformly among $X$ possibilities for each token.

The evaluation protocol processes the whole WikiText-2 test split (trimmed to 512 sequence length) by accumulating negative log-likelihoods across all valid tokens before computing the final exponential transformation. The implementation iterates through sequences, computing cross-entropy loss for each token while applying attention masks to exclude padding tokens from the calculation, ensuring that only meaningful content contributes to the perplexity measurement.

### 3.6.2 TriviaQA Question-Answering Evaluation

TriviaQA [43] constitutes a comprehensive reading comprehension dataset containing over 650,000 question-answer pairs sourced from trivia and quiz-bowl competitions. The dataset's distinctive characteristic lies in its dual-mode evaluation framework that enables assessment of both knowledge retrieval and reading comprehension capabilities through closed-book and open-book testing scenarios.

The closed-book evaluation presents models with questions in isolation, requiring them to rely entirely on knowledge encoded in their parameters during training. This mode directly tests the factual knowledge retention of compressed models and reveals whether compression techniques have degraded the model's ability to recall specific information. Questions use a simple prompt structure: "Question: [question text] Answer:".

Conversely, the open-book evaluation provides relevant context passages alongside each question, which simulates a reading comprehension scenario where models must extract answers from provided text. The context is derived from TriviaQA's associated search results, with the implementation selecting the shortest available context passage to minimize computational overhead while preserving essential information. Consequently, the prompt format becomes: "Context: [context passage] Question: [question text] Answer:".

Answer correctness evaluation employs a three-tier verification system designed to accommodate the variability inherent in natural language responses. The evaluation process begins with testing the exact matching of the answer, where the generated text is compared directly against all provided correct answer aliases after normalization. Normalization involves converting text to lowercase, removing punctuation, and standardizing whitespace to ensure fair comparison despite minor formatting differences.

When exact matching fails, the system applies substring matching, checking whether any correct answer appears as a contiguous substring within the generated response. This approach accounts for models that provide correct answers embedded within longer explanations or additional context, which is common behavior in instruction-tuned models.

The final verification stage implements fuzzy matching for cases where neither exact nor substring matching succeeds. For answers containing multiple words, the system computes the intersection between the set of words in the generated answer and each correct answer, considering a response correct if at least 80% of the words in the target answer appear in the generated text:

$$\text{Fuzzy Match} = \frac{|\text{Words}_{\text{generated}} \cap \text{Words}_{\text{correct}}|}{|\text{Words}_{\text{correct}}|} \geq 0.8 \tag{3.13}$$

Generation parameters are carefully set up to ensure consistent evaluation conditions. Models generate between 1 and 50 tokens per answer using deterministic decoding (with temperature set to 1.0) to mitigate randomness that could affect reproducibility. The evaluation framework processes 1200 samples of TriviaQA's validation split (alternatively, the quantity of samples can be tuned by the user), and the accuracy is computed separately for these two tasks.

# Chapter 4

# Experimental Results and Analysis

This chapter presents the empirical evaluation of the pipeline that was described in Chapter 3 when applied to the LLaMA 3.2 1B Instruct model. The experiments reveal systematic patterns in how different optimization techniques affect performance across multiple evaluation metrics.

Subsequent sections examine compression efficiency, quality degradation, and the interaction effects between sequential optimization stages. Results demonstrate that certain technique combinations achieve substantial parameter reduction while maintaining acceptable capability levels, whereas others exhibit rapid degradation beyond critical compression ratios.

The tables presented in this chapter display a representative subset of the experimental data to highlight key findings and trends. The complete results from all configurations that were tested are available in Appendix A, along with examples of text generated by selected models.

## 4.1 Pruning Analysis

The initial phase of experimentation examined the isolated effects of pruning techniques without quantization or recovery mechanisms. A total of 42 pruning-only configurations were evaluated, including 29 combined ones that applied both depth and width pruning simultaneously to assess the compounding effects of multi-dimensional compression.

Table 4.1 shows the results from a subset of the complete experimental data, with size measurements reflecting GPU VRAM consumption for model loading (without inference) in float16 precision on a single GPU of our testing cluster (which is comprised of 4× GTX 1080 Ti). Width-pruned variants show no memory reduction despite significant parameter reduction because the inference framework does not exploit structured sparsity patterns for memory optimization, requiring full dense tensor allocation regardless of zero weights. Implementing hardware-accelerated structured sparsity support would address this limitation and is discussed in Section 5.1.7.

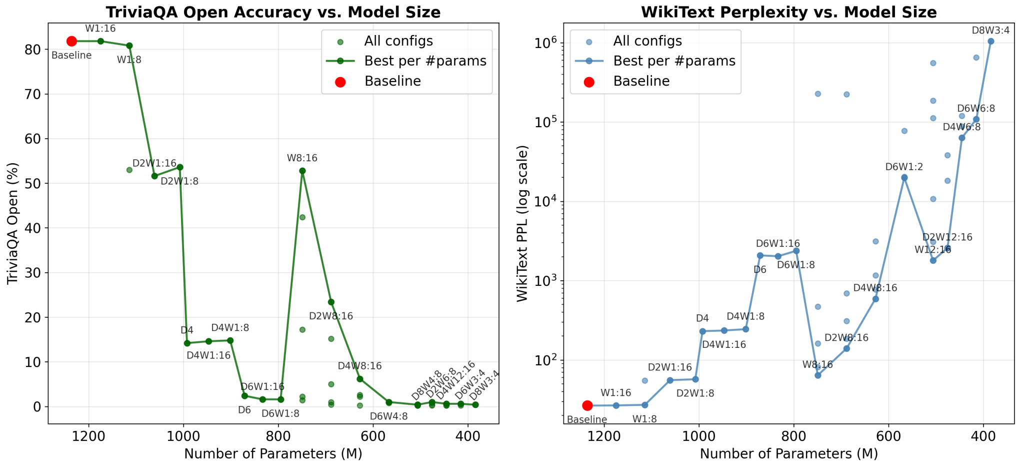


*Figure 4.1: The graph compares performance across parameter counts for models that were pruned without any additional transformations, using TriviaQA Open accuracy and WikiText perplexity as evaluation metrics. The darker line represents the Pareto front, highlighting which pruned models deliver the best results for their given parameter budget without being dominated by smaller, more efficient alternatives.*

As evidenced by Figure 4.1, the results demonstrate a steep degradation curve where even modest layer removal causes substantial quality deterioration. The most striking observation is the precipitous decline beyond removing two layers. While Depth 2, the variant with 2 layers removed, maintains reasonable functionality with 53.0% accuracy on TriviaQA's open-book benchmark (i.e. TriviaQA

*Table 4.1: Results for models which have undergone pruning without quantization or performance recovery. This table presents a subset of results from the complete dataset available in Appendix A.*

| Config | TriviaQA (%) ↑ | | WikiText ↓ | #Params | Size |
|---|---|---|---|---|---|
| | Closed | Open | PPL | | (GB) |
| *Baseline Instruct* | *50.6* | *81.8* | *26.61* | *1235.8M* | *2.30* |
| Depth 2 | 20.0 | 53.0 | 55.24 | 1114.2M | 2.08 |
| Depth 4 | 5.6 | 14.2 | 230.77 | 992.5M | 1.85 |
| Depth 6 | 2.0 | 2.4 | 2081.04 | 870.9M | 1.62 |
| Depth 8 | 1.2 | 1.4 | 227734.90 | 749.2M | **1.40** |
| Width 1:2 | 1.8 | 2.2 | 471.66 | 749.3M | 2.30 |
| Width 2:4 | 7.0 | 17.2 | 161.10 | 749.3M | 2.30 |
| Width 1:8 | 47.8 | 80.8 | 27.19 | 1114.2M | 2.30 |
| Width 4:8 | 12.6 | 42.4 | 81.32 | 749.3M | 2.30 |
| Width 1:16 | **49.2** | **81.8** | **26.77** | 1175.0M | 2.30 |
| Width 8:16 | 16.2 | 52.8 | 64.08 | 749.3M | 2.30 |
| Width 12:16 | 0.2 | 0.2 | 1793.97 | 506.0M | 2.30 |
| Depth 2 + Width 1:2 | 0.4 | 0.4 | 694.34 | 688.4M | 2.08 |
| Depth 2 + Width 2:4 | 3.8 | 5.0 | 309.32 | 688.4M | 2.08 |
| Depth 2 + Width 1:8 | 20.2 | 53.6 | 57.22 | 1007.7M | 2.08 |
| Depth 2 + Width 4:8 | 6.4 | 15.2 | 185.43 | 688.4M | 2.08 |
| Depth 2 + Width 1:16 | 19.6 | 51.6 | 55.68 | 1061.0M | 2.08 |
| Depth 2 + Width 8:16 | 7.6 | 23.4 | 139.97 | 688.4M | 2.08 |
| Depth 4 + Width 1:8 | 5.2 | 14.8 | 244.69 | 901.3M | 1.85 |
| Depth 4 + Width 4:8 | 3.0 | 2.6 | 774.60 | 627.6M | 1.85 |
| Depth 4 + Width 1:16 | 4.6 | 14.6 | 235.32 | 946.9M | 1.85 |
| Depth 4 + Width 8:16 | 2.6 | 6.2 | 591.59 | 627.6M | 1.85 |
| Depth 6 + Width 1:8 | 2.0 | 1.6 | 2376.63 | 794.9M | 1.62 |
| Depth 6 + Width 4:8 | 0.8 | 1.0 | 77482.90 | 566.8M | 1.62 |
| Depth 6 + Width 1:16 | 1.8 | 1.6 | 2032.84 | 832.9M | 1.62 |
| Depth 8 + Width 4:8 | 0.8 | 0.4 | 554910.71 | 506.0M | **1.40** |
| Depth 8 + Width 3:4 | 0.4 | 0.4 | 1044840.71 | **384.3M** | **1.40** |

Open) compared to the baseline's 81.8%, removing four layers causes performance to collapse to just 14.2%. This suggests that architectural depth in Transformer models exhibits critical thresholds beyond which fundamental language capabilities deteriorate rapidly.

The perplexity measurements on WikiText-2 reinforce this pattern, with Depth 2 showing a manageable increase to 55.24 compared to the baseline's 26.61, while deeper pruning leads to catastrophic degradation. Removing eight layers (Depth 8 configuration) causes perplexity to exceed beyond 200,000, indicating near-complete loss of coherent language modeling capability.

On the other hand, width pruning shows markedly different characteristics. The Width 1:8 variant emerges as particularly noteworthy, maintaining 80.8% accuracy on TriviaQA Open while reducing parameters from 1235.8M to 1114.2M. This represents only a minimal performance drop of 1.0 percentage point while achieving a 10% reduction in model size, with parameter count comparable to the Depth 2 configuration. The perplexity increase is equally modest, rising from 26.61 to 27.19. This finding suggests that Transformer architectures contain significant redundancy in their width dimension when weights are removed judiciously, with the WANDA algorithm's ability to identify and remove less critical weights while preserving essential model capacity proving effective for moderate compression ratios, thus proving that depth pruning is significantly more destructive than width pruning even when removing fewer parameters.

However, more aggressive width pruning follows the same catastrophic degradation pattern observed when extensively removing entire layers. The Width 1:2 configuration reduces TriviaQA Open accuracy to just 2.2%, thus proving that there are clear limits to how extensively models can be compressed through width reduction alone. Moreover, the granularity of width pruning significantly influences these results, as evidenced by the progression from Width 1:2 (2.2% TriviaQA Open) to Width 2:4 (17.2%) to Width 4:8 (42.4%), showing how different sparsity patterns at the same compression ratio yield vastly different outcomes.

Ultimately, performance of models with combined configurations, without recovery mechanisms, proves unacceptable for practical deployment unless weight removal remains minimal. The only viable exception is Depth 2 + Width 1:8, which maintains 53.6% accuracy on TriviaQA Open while achieving an 18% parameter reduction, representing the practical threshold for structural compression without additional optimization techniques.

## 4.2 Complete Pipeline Performance

Of the 42 pruned-only configurations, 27 were selected to undergo LoRA fine-tuning and quantization. In particular, training was conducted separately on both WikiText and TriviaQA train datasets using 8192 samples divided into batches of 8. Additionally, 12 of these variants also underwent EoRA performance recovery to assess its effectiveness in mitigating quantization-induced degradation. Table 4.2 presents key configurations that showcase different aspects of the optimization process.

LoRA adaptation reveals significant effectiveness in recovering capabilities lost during pruning across all tested configurations. The Depth 2 variant exemplifies this recovery potential, with TriviaQA fine-tuning improving open-book performance from the pruned baseline of 53.0% to 60.2% and closed-book accuracy from 20.0% to 27.8%. The perplexity improvements are equally dramatic, with WikiText fine-tuning reducing perplexity from 55.24 to 21.34, bringing the compressed model's language modeling capabilities significantly closer to the original baseline. The Width 1:8 configuration demonstrates even more impressive recovery, maintaining 80.8% TriviaQA Open performance while achieving remarkable perplexity reduction from 27.19 to 15.70 with WikiText fine-tuning. Figure 4.2 provides a visual comparison of a select set of configurations.

The recovery potential of LoRA becomes even more striking when dealing with heavily compressed models. For the aggressively pruned Depth 6 + Width 4:8 configuration, LoRA fine-tuning dramatically reduces perplexity from 554910 to 55.46, demonstrating that even severely degraded models can achieve substantial recovery. When quantization is applied to this setup, the model still maintains a perplexity of 58.80, which represents a reasonable score considering the extreme

*Table 4.2: Results for models which have undergone the complete optimization pipeline or selected steps in addition to pruning. This table presents a subset of results from the complete dataset available in Appendix A.*

| **Config** | **LoRA** | **Quant** | **TriviaQA (%) ↑** | | **WikiText ↓** | **Size** |
|---|---|---|---|---|---|---|
| | **Type** | | **Closed** | **Open** | **PPL** | **(GB)** |
| *Baseline Instruct* | *None* | *No* | *50.6* | *81.8* | *26.61* | *2.30* |
| Depth 2 | TriviaQA | No | 27.8 | 60.2 | 51.09 | 2.08 |
| Depth 2 | TriviaQA | Yes | 23.9 | 54.6 | 55.03 | 0.92 |
| Depth 2 | TriviaQA | EoRA | 24.4 | 58.4 | 54.60 | 0.92 |
| Width 1:2 | WikiText | No | 10.0 | 32.0 | 40.26 | 2.30 |
| Width 1:2 | WikiText | Yes | 9.1 | 27.3 | 42.19 | 0.98 |
| Width 1:2 | WikiText | EoRA | 8.8 | 28.3 | 41.78 | 0.98 |
| Width 1:2 | TriviaQA | No | 12.4 | 29.3 | 145.54 | 2.30 |
| Width 1:2 | TriviaQA | Yes | 11.3 | 26.5 | 154.33 | 0.98 |
| Width 1:2 | TriviaQA | EoRA | 11.7 | 25.7 | 152.53 | 0.98 |
| Width 1:8 | WikiText | No | **47.8** | **80.8** | **15.70** | 2.30 |
| Width 1:8 | WikiText | Yes | 39.0 | 74.8 | 16.85 | 0.98 |
| Width 1:8 | TriviaQA | EoRA | 42.0 | 77.2 | 32.86 | 0.98 |
| Width 4:8 | WikiText | No | 17.0 | 51.1 | 26.04 | 2.30 |
| Width 4:8 | WikiText | Yes | 15.7 | 45.3 | 27.40 | 0.98 |
| Width 8:16 | TriviaQA | Yes | 20.8 | 55.7 | 60.67 | 0.98 |
| Width 8:16 | WikiText | Yes | 17.5 | 52.1 | 25.39 | 0.98 |
| Width 8:16 | WikiText | EoRA | 17.8 | 54.9 | 25.09 | 0.98 |
| Depth 2 + Width 1:8 | WikiText | EoRA | 22.3 | 59.6 | 22.98 | 0.92 |
| Depth 2 + Width 1:8 | TriviaQA | No | 27.4 | 60.4 | 51.89 | 2.08 |
| Depth 2 + Width 1:16 | TriviaQA | Yes | 25.2 | 56.3 | 55.46 | 0.92 |
| Depth 2 + Width 2:4 | WikiText | Yes | 10.6 | 30.8 | 41.54 | 0.92 |
| Depth 2 + Width 2:4 | WikiText | EoRA | 10.4 | 29.8 | 40.73 | 0.92 |
| Depth 4 + Width 1:16 | WikiText | No | 18.1 | 53.2 | 29.51 | 1.85 |
| Depth 4 + Width 1:16 | WikiText | Yes | 15.2 | 48.6 | 31.66 | 0.86 |
| Depth 4 + Width 1:16 | TriviaQA | No | 18.7 | 47.3 | 89.32 | 1.85 |
| Depth 4 + Width 2:4 | WikiText | Yes | 7.2 | 25.8 | 52.20 | 0.86 |
| Depth 4 + Width 2:4 | WikiText | EoRA | 7.6 | 26.3 | 51.19 | 0.86 |
| Depth 4 + Width 4:8 | WikiText | Yes | 8.2 | 29.6 | 46.34 | 0.86 |
| Depth 4 + Width 4:8 | WikiText | EoRA | 8.3 | 31.1 | 45.53 | 0.86 |
| Depth 4 + Width 4:8 | TriviaQA | No | 10.5 | 20.8 | 210.94 | 1.85 |
| Depth 6 + Width 1:16 | WikiText | No | 11.5 | 25.6 | 37.82 | 1.62 |
| Depth 6 + Width 1:16 | TriviaQA | No | 16.8 | 27.5 | 310.53 | 1.62 |
| Depth 6 + Width 4:8 | WikiText | No | 0.9 | 9.6 | 55.46 | 1.62 |
| Depth 6 + Width 4:8 | WikiText | Yes | 2.1 | 10.7 | 58.80 | 0.80 |
| Depth 8 + Width 3:4 | TriviaQA | No | 0.3 | 0.3 | 7263.56 | 1.40 |
| Depth 8 + Width 3:4 | WikiText | Yes | 0.5 | 1.3 | 289.45 | **0.74** |
| Depth 8 + Width 3:4 | TriviaQA | No | 0.3 | 0.3 | 7263.56 | 1.40 |
| Depth 8 + Width 3:4 | TriviaQA | Yes | 0.8 | 0.2 | 9326.59 | **0.74** |

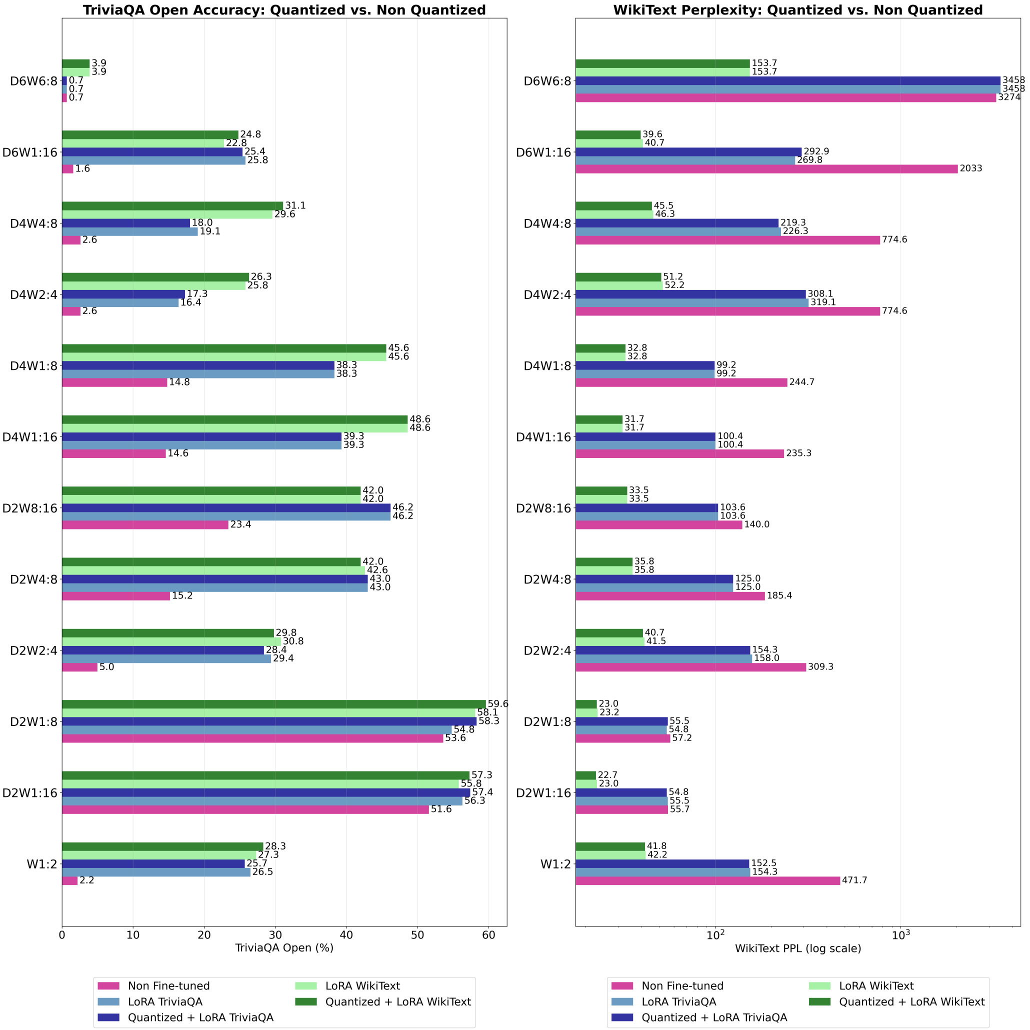


*Figure 4.2: This figure shows a performance comparison of various pruned model configurations when LoRA fine-tuning and quantization are applied to them. Each configuration is evaluated under five conditions: the baseline pruned model, LoRA fine-tuning on WikiText, LoRA fine-tuning on TriviaQA, and both LoRA methods combined with quantization. Performance is measured using TriviaQA Open accuracy and WikiText perplexity metrics.*

compression applied.

Interestingly, WikiText fine-tuning frequently outperforms TriviaQA fine-tuning across configurations, even when evaluated on TriviaQA benchmarks. For instance, the Width 1:2 variant with 4-bit quantization exhibits this pattern: WikiText LoRA achieves comparable TriviaQA open-book accuracy (27.3% vs. 26.5%) while delivering dramatically better perplexity (42.19 vs. 154.33) compared to TriviaQA LoRA. This pattern holds consistently across all optimization stages for this configuration, with WikiText variants achieving superior TriviaQA Open performance in the non-quantized (32.0% vs. 29.3%) and EoRA-compensated (28.3% vs. 25.7%) versions. This suggests that WikiText's broader linguistic diversity provides more effective adaptation for general language modeling capabilities, which translates to better overall results even on task-specific evaluations.

The addition of 4-bit quantization introduces a predictable trade-off between memory efficiency and quality. Across configurations, quantization typically causes 3-6 percentage point degradation on TriviaQA tasks, though this cost remains manageable given the substantial memory savings achieved through 4-bit representation. The Width 1:8 variant exemplifies this trade-off, dropping from 80.8% to 74.8% TriviaQA Open accuracy with quantization while maintaining excellent perplexity scores.

In contrast, EoRA compensation provides modest but consistent improvements over standard quantization. The technique typically recovers 1-2 percentage points of lost performance with minimal computational overhead. For the Depth 4 + Width 4:8 setting, EoRA improves accuracy from 29.6% to 31.1% TriviaQA Open while slightly reducing perplexity from 46.34 to 45.53. These gains, while incremental, show that EoRA provides only marginal quality improvements given its rather low computational overhead. A visualization of the effect of EoRA on the quantized models is illustrated in Figure 4.3.

Nevertheless, even when recovery techniques are applied to the most compressed models, their performance degradation remains unsalvageable. The Depth 8 + Width 3:4 configuration demonstrates this limitation, where despite TriviaQA fine-tuning, the model yields essentially nil accuracy of only 0.3% on both Triv-

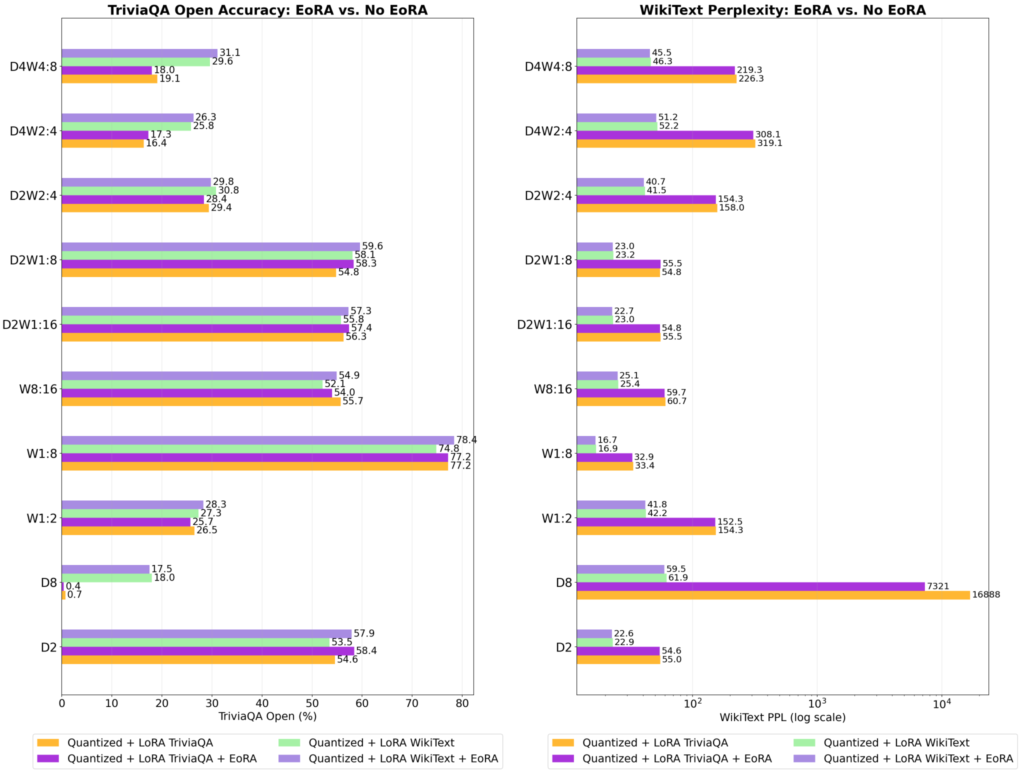


*Figure 4.3: This figure shows the effect of EoRA on a subset of fine-tuned quantized model configurations. Each configuration is evaluated under four conditions: quantized LoRA fine-tuning on TriviaQA, quantized LoRA fine-tuning on WikiText, and both methods enhanced with EoRA performance recovery. Performance is measured using TriviaQA Open accuracy and WikiText perplexity metrics.*

iaQA metrics with a perplexity of 7263.56, rendering it effectively unusable for practical applications.

# Chapter 5

# Conclusion and Future Work

The compression pipeline developed throughout this thesis represents a comprehensive approach to reducing the computational and memory requirements of large language models while maintaining acceptable performance levels. By combining multiple optimization techniques in a carefully orchestrated sequence, the work demonstrates that aggressive model compression remains viable even for already-distilled architectures like LLaMA 3.2 1B.

The sequential application of depth pruning, width pruning, LoRA adaptation, quantization, and EoRA compensation creates a systematic pathway from the original 1B parameter model to significantly more compact variants suitable for deployment on resource-constrained hardware.

The evaluation framework reveals that these techniques, when properly integrated, can achieve substantial compression ratios while preserving core language modeling capabilities. The WikiText-2 perplexity results demonstrate that compressed models retain their fundamental ability to predict token sequences, while TriviaQA evaluations confirm that both factual knowledge retention and reading comprehension abilities survive the compression process with manageable degradation for moderate compression ratios, and can often be restored using a LoRA adapter in configurations where heavy performance loss occurs.

However, the work also highlights several limitations that constrain its immediate applicability. The most significant challenge lies in the substantial deterioration observed under aggressive compression ratios, where even refinement techniques such as LoRA adaptation and EoRA compensation cannot fully recover the capabilities lost during extensive pruning, resulting in models that struggle to maintain coherent language generation and factual accuracy for the most memory-constrained edge devices. Additionally, the sequential nature of the pipeline may not capture optimal interactions between different compression stages. The evaluation framework also presents limitations, as assessments concentrated primarily on English language tasks and, despite using relatively large prompts for reading comprehension tasks, focused on relatively short-context scenarios compared to the extended sequences that modern LLMs are increasingly expected to handle, leaving uncertainty about compression behavior on longer sequences and multilingual applications.

Nevertheless, this work establishes a solid foundation for practical model compression and demonstrates that multi-stage optimization approaches leveraging pruning techniques and quantization represent a viable pathway toward developing both environmentally sustainable LLMs and making advanced AI capabilities accessible across low-powered, resource-constrained devices.

## 5.1 Future Work

While the models produced by the compression pipeline developed in this thesis demonstrate promising results across multiple evaluation benchmarks, numerous avenues remain unexplored that could significantly enhance both the quality of compressed models and the scope of the framework itself. These directions encompass methodological refinements, algorithmic innovations, and broader architectural compatibility considerations that could substantially improve the project's capabilities and applicability.

### 5.1.1 Knowledge Distillation Integration

One of the most significant limitations of the current approach lies in its reliance on pre-distilled models rather than incorporating distillation directly into the compression pipeline. As outlined in Section 2.5.1, knowledge distillation represents a rather effective compression technique, yet computational constraints prevented its integration into this work. Future research should prioritize developing distillation workflows that can operate within reasonable computational budgets.

In this context, an interesting direction could involve adopting approaches such as *Homotopic Distillation* (HomoDistil) [21], which combines iterative pruning with knowledge distillation in a unified framework. HomoDistil (Section 2.5.1) addresses the capacity gap problem by starting with the full teacher model and gradually removing neurons while simultaneously distilling knowledge, maintaining small prediction discrepancies throughout the process. This approach could be particularly well-suited to the current pipeline, as it directly addresses the performance degradation observed under aggressive pruning by providing continuous guidance during the compression process rather than attempting recovery afterward.

### 5.1.2 Additional Quantization Methodologies

While GPTQ proved effective in the current pipeline, the quantization landscape continues evolving rapidly with new algorithms which address different aspects of the precision-accuracy trade-off. *Activation-aware Weight Quantization* (AWQ) [45] represents a particularly promising alternative that identifies salient weight channels based on activation distributions rather than weight magnitudes alone. AWQ's key insight that protecting only 1% of the most important weights can dramatically reduce quantization error suggests potential for even more aggressive compression than achieved with GPTQ. Unlike GPTQ's reliance on Hessian information, AWQ employs mathematically equivalent transformations to scale salient channels, avoiding hardware-inefficient mixed-precision schemes while maintaining better generalization across domains.

*Quality Quattuor-bit Quantization* (QQQ) [44] offers another compelling approach that addresses the performance-speed trade-off through W4A8 quantization (4-bit weights, 8-bit activations). The method combines two key innovations:

adaptive smoothing that selectively targets activation channels with significant outliers while preserving other channels, and Hessian-based compensation that employs the same mathematical framework as GPTQ for iterative weight adjustments to minimize quantization losses. This dual approach enables QQQ to handle both weight and activation quantization simultaneously, potentially achieving better compression ratios than weight-only methods while maintaining acceptable results. Notably, QQQ is natively supported by the GPTQModel toolkit [38], making integration into the existing pipeline straightforward and potentially offering superior performance recovery compared to current weight-only quantization approaches.

### 5.1.3 Advanced Investigations of Interdependencies

The current pipeline implements compression techniques sequentially with minimal consideration of their interactions, yet understanding how different stages influence each other could provide insights into significantly improved compression strategies. Future investigations should systematically examine adaptive methodologies that adjust later techniques based on earlier results, following a similar approach to 2SSP [22] (Section 2.5.2); for instance, the ratios and thresholds for width pruning could be dynamically determined based on the specific layers removed during depth pruning, potentially targeting different sparsity levels in regions where architectural changes have already occurred.

Likewise, applying consistent methodologies across compression stages warrants exploration, such as using perplexity-based importance for both depth and width pruning rather than the current mixed approach of perplexity for layers and magnitude-based metrics for weights. The layer preservation employed in depth pruning, which safeguards the first four and last two layers from removal, could also be extended to width pruning operations. This would protect critical architectural components across both compression dimensions while enabling more aggressive optimization in intermediate regions.

The unification of recovery methods represents another possible improvement. Rather than applying LoRA and EoRA as separate sequential steps, these techniques could potentially be unified into a single adapter mechanism, in order to simplify the recovery process by eliminating intermediate steps.

Otherwise, EoRA can be leveraged as superior starting point for LoRA fine-tuning. The original EoRA paper [40] demonstrates that using EoRA matrices to initialize LoRA substantially enhances accuracy recovery compared to standard methods. This strategy could prove particularly effective for heavily compressed models, where EoRA's eigenspace projection provides a more informed foundation for subsequent gradient-based optimization.

Alternative sequencing configurations also warrant systematic investigation. Rather than the current Depth→Width→LoRA progression, arrangements such as Depth→LoRA→Width could allow for performance recovery immediately after the most aggressive structural changes, potentially preserving more model capabilities during subsequent fine-grained pruning[1].

### 5.1.4 Extended Model Compatibility

The current pipeline's focus on LLaMA architectures, while strategic for this work, limits its broader applicability. Extending support to popular open-weight model families like DeepSeek [46] and Qwen [47] would significantly increase the framework's utility and enable broader comparative studies across different architectural paradigms.

While some components of the pipeline already support these architectures through underlying libraries like GPTQModel, which provides native support for both DeepSeek and Qwen models [38], other aspects of the framework may not accommodate these changes seamlessly. The compression pipeline's layer importance calculations, pruning strategies, and evaluation protocols were specifically designed around LLaMA's architectural characteristics and may require substantial modifications for different model families. Additionally, availability constraints affect the practicality of extending to certain architectures: DeepSeek models are not available in small language model configurations comparable to

[1] Preliminary experiments on this Depth→LoRA→Width configuration were conducted, with results presented in Appendix B.

LLaMA 3.2 1B, limiting their suitability for the target deployment scenarios. Conversely, Qwen offers promising alternatives with 1B and even 0.5B parameter variants in the Qwen2 series [48], potentially enabling exploration of even more aggressive compression ratios than achievable with LLaMA.

Beyond expanding to existing architectures, the pipeline design should anticipate future model developments. Creating modular, architecture-agnostic compression components that can be easily adapted to new Transformer variants would ensure the framework's longevity.

Finally, one must consider that different architectures may exhibit varying sensitivities to compression techniques, requiring architecture-specific tuning of pruning criteria, quantization parameters, and LoRA configurations. Systematic studies of how compression techniques interact with different architectural choices could inform more effective optimization strategies and reveal whether the current pipeline's assumptions hold across diverse model families.

### 5.1.5 Evaluation Framework Improvements

The current evaluation methodology provides solid insights into compression effects on language model performance, yet several expansions can applied to increase the scope and depth of future assessments.

Extending the linguistic scope presents an opportunity to explore compression behavior across diverse language families. Different idioms exhibit varying degrees of morphological complexity that could interact differently with the pipeline, particularly when comparing agglutinative systems to isolating ones. Large-scale multilingual datasets like HPLT [49], covering 75 distinct tongues, offer promising avenues for comprehensive cross-linguistic evaluation and could reveal language-specific vulnerabilities or robustness patterns. Additionally, it would be interesting to experiment with how LoRA adaptation on different languages from HPLT would influence performance on the WikiText-2 and TriviaQA evaluation methods previously adopted in this work.

Incorporating longer context evaluation represents another valuable enhancement. The focus on relatively short-context scenarios fails to capture compression behavior on larger sequences that increasingly define modern LLM applications. Thus, understanding how compression affects performance on sequences spanning thousands of tokens becomes essential. Extended context evaluation would be particularly valuable for compressed models, as the interaction between reduced parameter counts and long-range dependencies may create failure modes not fully visible in shorter evaluation scenarios.

The diversity of task assessments could be broadened. The current focus on perplexity and reading comprehension does not address the broad spectrum of capabilities that modern LLMs possess. Code generation represents a particularly valuable extension, as programming tasks require precise logical reasoning and structured output generation that may be disproportionately affected by parameter reduction. Recent proposals like CodeJudge [50] show that LLM-based evaluation can assess semantic correctness of generated code without requiring traditional test cases, and thus could be incorporated more easily into the evaluation framework.

### 5.1.6 KV Cache Compression

Key-value cache compression represents a critical bottleneck for auto-regressive generation that the current pipeline does not address. During inference, the KV cache stores attention keys and values for all previous tokens to prevent re-computation, but its size grows linearly with sequence length, creating substantial memory pressure that often exceeds hardware limitations [51].

Numerous techniques exist for KV cache compression, including quantization methods that reduce the precision of stored key-value pairs and pruning approaches that selectively remove components based on their significance [52]. Similar to how the current pipeline applies importance-based pruning to model parameters, KV cache optimizations can exploit fine-grained differences in significance across multiple dimensions (e.g. the differing computational impact of keys versus values in attention mechanisms, or the varying importance of individual tokens based on their contribution to subsequent predictions). Rather than applying uniform compression to all cache components, these approaches can selectively preserve the most critical elements while aggressively compressing or removing less important ones.

Recent frameworks like LeanKV [52] demonstrate that such differentiated approaches can achieve substantial compression ratios while maintaining near-lossless accuracy on complex reasoning tasks. Integrating similar KV cache optimization techniques into the pipeline could further optimize memory consumption beyond the current parameter reduction focus, and result in a more comprehensive approach that addresses both static model size and dynamic inference memory requirements.

### 5.1.7 Engineering Improvements

Several implementation enhancements could improve the practical performance of the compression pipeline without requiring algorithmic innovations. Flash Attention [53] and its variants represent a direct optimization opportunity, as these techniques reorganize attention computation to minimize memory transfers between GPU memory hierarchies while maintaining mathematical equivalence to standard attention. Since Flash Attention operates independently of model compression, integrating it into the inference workflow would complement the existing compression techniques by reducing runtime memory pressure during the pipeline execution and model assessment phases without affecting model weights or architecture. This could be implemented through Hugging Face's BetterTransformer [54], which provides optimized attention kernels including Flash Attention for supported architectures. However, BetterTransformer compatibility remains limited to specific model families, potentially creating challenges when extending the pipeline to Transformer variants beyond LLaMA 2 and 3, where such optimizations may not be readily available.

Supporting hardware-accelerated structured sparsity represents another feature that could significantly improve the pipeline's evaluation performance for models pruned in a structured fashion. As described in Section 3.3.2, NVIDIA's Ampere and Hopper GPU architectures provide native support for structured sparse matrix operations that can achieve theoretical $2\times$ speedups over dense computations [32]. While the current pipeline produces models with structured sparsity patterns, the inference implementation does not exploit these hardware capabilities. Adapting the framework to leverage these acceleration features would require substantial compatibility efforts, as structured sparsity support remains in PyTorch's nightly builds rather than stable releases at the time of writing [55]. In addition, implementation would likely necessitate redefining the Transformer architecture using the TorchAO optimization library [56], which provides the necessary primitives for structured sparse operations but requires careful integration with existing inference workflows.

Both improvements focus on optimizing the development and evaluation workflow rather than advancing compression methodologies themselves. While they could substantially enhance the practical utility and efficiency of the pipeline's operation, they do not necessarily translate to other embedded architectures where the models may be deployed, instead requiring specific hardware-dependent implementations that may not be available across all target platforms.

# Bibliography

[1] Mike Perkins, *Academic Integrity considerations of AI Large Language Models in the post-pandemic era: ChatGPT and beyond*, 2023. DOI: 10.53761/1.20.02.07. URL: https://doi.org/10.53761/1.20.02.07

[2] Daniel Mügge, *AI Is Threatening More Than Just Creative Jobs—It's Undermining Our Humanity*, Social Europe, 2024. URL: https://www.socialeurope.eu/ai-is-threatening-more-than-just-creative-jobs-its-undermining-our-humanity

[3] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, Jeff Dean, *Carbon Emissions and Large Neural Network Training*, 2021. arXiv: 2104.10350 [cs.LG]. URL: https://arxiv.org/abs/2104.10350

[4] Thomas Spencer, Siddharth Singh, *What the data centre and AI boom could mean for the energy sector*, IEA, 2024. URL: https://www.iea.org/commentaries/what-the-data-centre-and-ai-boom-could-mean-for-the-energy-sector

[5] Nidhal Jegham, Marwen Abdelatti, Lassad Elmoubarki, Abdeltawab Hendawi, *How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference*, 2025. arXiv: 2505.09598v1 [cs.CY]. URL: https://arxiv.org/abs/2505.09598v1

[6] Google, *Google Environmental Report 2023*, 2023. URL: https://sustainability.google/reports/google-2023-environmental-report-executive-summary/

[7] Pengfei Li, Jianyi Yang, Mohammad A. Islam, Shaolei Ren, *Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models*, 2023. arXiv: 2304.03271 [cs.LG]. URL: https://arxiv.org/abs/2304.03271

[8] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, *Improving Language Understanding by Generative Pre-Training*, OpenAI, 2018. URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[9] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman et al., *GPT-4 Technical Report*, 2023. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774

[10] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, *On the difficulty of training Recurrent Neural Networks*, 2012. arXiv:1211.5063 [cs.LG]. URL: https://arxiv.org/abs/1211.5063

[11] S. Hochreiter, J. Schmidhuber, *Long Short-Term Memory*, 1997. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735

[12] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton, *Speech Recognition with Deep Recurrent Neural Networks*, 2013. arXiv: 1303.5778 [cs.NE]. URL: https://arxiv.org/abs/1303.5778

[13] Mustafa Abbas Hussein Hussein, Serkan Savaş, *LSTM-Based Text Generation: A Study on Historical Datasets*, 2024. arXiv: 2403.07087 [cs.CL]. URL: https://arxiv.org/abs/2403.07087

[14] Philip Gage, *A new algorithm for data compression*, 1994. DOI: 10.5555/177910.177914. URL: `https://dl.acm.org/doi/10.5555/177910.177914`

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, *Attention is All You Need*, 2017. arXiv: 1706.03762 [cs.CL]. URL: `https://arxiv.org/abs/1706.03762`

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro et al., *LLaMA: Open and Efficient Foundation Language Models*, 2023. arXiv: 2302.13971 [cs.CL]. URL: `https://arxiv.org/abs/2302.13971`

[17] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava et al., *Llama 2: Open Foundation and Fine-Tuned Chat Models*, 2023. arXiv: 2307.09288 [cs.CL]. URL: `https://arxiv.org/abs/2307.09288`

[18] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten et al., *The Llama 3 Herd of Models*, 2024. arXiv: 2407.21783 [cs.AI]. URL: `https://arxiv.org/abs/2407.21783`

[19] Arpan Suravi Prasad, Moritz Scherer, Francesco Conti, Davide Rossi, Alfio Di Mauro, Manuel Eggimann, Jorge Tómas Gómez, Ziyun Li, Syed Shakib Sarwar, Zhao Wang, Barbara De Salvo, Luca Benini, *Siracusa: A 16 nm Heterogenous RISC-V SoC for Extended Reality with At-MRAM Neural Engine*, 2023. arXiv: 2312.14750 [cs.AR]. URL: `https://arxiv.org/abs/2312.14750`

[20] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, *Distilling the Knowledge in a Neural Network*, 2015. arXiv: 1503.02531 [stat.ML]. URL: `https://arxiv.org/abs/1503.02531`

[21] Chen Liang, Haoming Jiang, Zheng Li, Xianfeng Tang, Bin Yin, Tuo Zhao, *HomoDistil: Homotopic Task-Agnostic Distillation of Pre-trained Transformers*, 2023. arXiv: 2302.09632 [cs.CL]. URL: `https://arxiv.org/abs/2302.09632`

[22] Fabrizio Sandri, Elia Cunegatti, Giovanni Iacca, *2SSP: A Two-Stage Framework for Structured Pruning of LLMs*, 2025. arXiv: 2501.17771 [cs.CL]. URL: `https://arxiv.org/abs/2501.17771`

[23] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, Danqi Chen, *Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning*, 2023. arXiv: 2310.06694 [cs.CL]. URL: `https://arxiv.org/abs/2310.06694`

[24] Angelo Galavotti, *FRANKEN-LLAMA*, GitHub Repository, Last accessed July 2025. URL: `https://github.com/AngeloGalav/franken-llama`

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019. arXiv: 1912.01703 [cs.LG]. URL: `https://arxiv.org/abs/1912.01703`

[26] Hugging Face, *Transformers*, Library Documentation, Last accessed July 2025. URL: `https://huggingface.co/docs/transformers/index`

[27] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, Yejin Choi, *HellaSwag: Can a Machine Really Finish Your Sentence?*, 2019. arXiv: 1905.07830 [cs.CL]. URL: `https://arxiv.org/abs/1905.07830`

[28] Llama Team, AI @ Meta, *Llama-3.2-1B*, Hugging Face Model, 2024. URL: `https://huggingface.co/meta-llama/Llama-3.2-1B`

[29] Llama Team, AI @ Meta, *Llama-3.2-1B-Instruct*, Hugging Face Model, 2024. URL: `https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct`

[30] Bo-Kyeong Kim, Geonmin Kim, Tae-Ho Kim, Thibault Castells, Shinkook Choi, Junho Shin, Hyoung-Kyu Song, *Shortened LLaMA: Depth Pruning for Large Language Models with Comparison of Retraining Methods*, 2024. arXiv: 2402.02834 [cs.LG]. URL: `https://arxiv.org/abs/2402.02834`

[31] Mingjie Sun, Zhuang Liu, Anna Bair, J. Zico Kolter, *A Simple and Effective Pruning Approach for Large Language Models*, 2023. arXiv: 2306.11695 [cs.CL]. URL: `https://arxiv.org/abs/2306.11695`

[32] Hongxiao Bai, Yun Li, *Structured Sparsity in the NVIDIA Ampere Architecture and Applications in Search Engines*, NVIDIA Developer Blog, 2023. URL: `https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/`

[33] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, 2021. arXiv: 2106.09685 [cs.CL]. URL: `https://arxiv.org/abs/2106.09685`

[34] Hugging Face, *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning*, GitHub Repository, Last accessed July 2025. URL: `https://github.com/huggingface/peft`

[35] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, Jian Cheng, *Quantized Convolutional Neural Networks for Mobile Devices*, 2015. arXiv: 1512.06473 [cs.CV]. URL: `https://arxiv.org/abs/1512.06473`

[36] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, Dan Alistarh, *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*,

2022. arXiv: 2210.17323 [cs.LG]. URL: `https://arxiv.org/abs/2210.17323`

[37] Elias Frantar, Sidak Pal Singh, Dan Alistarh, *Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning*, 2022. arXiv: 2208.11580 [cs.LG]. URL: `https://arxiv.org/abs/2208.11580`

[38] ModelCloud, *GPTQModel*, GitHub Repository, Last accessed July 2025. URL: `https://github.com/ModelCloud/GPTQModel`

[39] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, 2019. arXiv: 1910.10683 [cs.LG]. URL: `https://arxiv.org/abs/1910.10683`

[40] Shih-Yang Liu, Maksim Khadkevich, Nai Chit Fung, Charbel Sakr, Chao-Han Huck Yang, Chien-Yi Wang, Saurav Muralidharan, Hongxu Yin, Kwang-Ting Cheng et al., *EoRA: Fine-tuning-free Compensation for Compressed LLM with Eigenspace Low-Rank Approximation*, 2024. arXiv: 2410.21271 [cs.CL]. URL: `https://arxiv.org/abs/2410.21271`

[41] Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher, *Pointer Sentinel Mixture Models*, 2016. arXiv: 1609.07843 [cs.CL]. URL: `https://arxiv.org/abs/1609.07843`

[42] F. Jelinek, R. L. Mercer, L. R. Bahl, J. K. Baker, *Perplexity—a measure of the difficulty of speech recognition tasks*, 1977. DOI: 10.1121/1.2016299. URL: `https://doi.org/10.1121/1.2016299`

[43] Mandar Joshi, Eunsol Choi, Daniel S. Weld, Luke Zettlemoyer, *TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension*, 2017. arXiv: 1705.03551 [cs.CL]. URL: `https://arxiv.org/abs/1705.03551`

[44] Ying Zhang, Peng Zhang, Mincong Huang, Jingyang Xiang, Yujie Wang, Chao Wang, Yineng Zhang, Lei Yu, Chuan Liu, Wei Lin, *QQQ: Quality Quattuor-Bit Quantization for Large Language Models*, 2024. arXiv: 2406.09904 [cs.LG]. URL: `https://arxiv.org/abs/2406.09904`

[45] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, Song Han, *AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration*, 2023. arXiv: 2306.00978 [cs.CL]. URL: `https://arxiv.org/abs/2306.00978`

[46] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang et al., *DeepSeek-V3 Technical Report*, 2024. arXiv: 2412.19437 [cs.CL]. URL: `https://arxiv.org/abs/2412.19437`

[47] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji et al., *Qwen Technical Report*, 2023. arXiv: 2309.16609 [cs.CL]. URL: `https://arxiv.org/abs/2309.16609`

[48] Hugging Face, *Qwen's Collections*, Hugging Face Model Collection, Last accessed July 2025. URL: `https://huggingface.co/collections/Qwen/qwen2-6659360b33528ced941e557f`

[49] Ona de Gibert, Graeme Nail, Nikolay Arefyev, Marta Bañón, Jelmer van der Linde, Shaoxiong Ji, Jaume Zaragoza-Bernabeu, Mikko Aulamo et al., *A New Massive Multilingual Dataset for High-Performance Language Technologies*, 2024. arXiv: 2403.14009 [cs.CL]. URL: `https://arxiv.org/abs/2403.14009`

[50] Weixi Tong, Tianyi Zhang, *CodeJudge: Evaluating Code Generation with Large Language Models*, 2024. arXiv: 2410.02184 [cs.LG]. URL: `https://arxiv.org/abs/2410.02184`

[51] Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhuo Xu et al., *KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches*, 2024. arXiv: 2407.01527 [cs.CL]. URL: `https://arxiv.org/abs/2407.01527`

[52] Yanqi Zhang, Yuwei Hu, Runyuan Zhao, John C.S. Lui, Haibo Chen, *Unifying KV Cache Compression for Large Language Models with LeanKV*, 2024. arXiv: 2412.03131v2 [cs.LG]. URL: `https://arxiv.org/abs/2412.03131v2`

[53] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, Christopher Ré, *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*, 2022. arXiv: 2205.14135 [cs.LG]. URL: `https://arxiv.org/abs/2205.14135`

[54] HuggingFace, *BetterTransformer*, Library Documentation, Last accessed July 2025. URL: `https://huggingface.co/docs/optimum/bettertransformer/overview`

[55] Jesse Cai, Daniel Haziza, Supriya Rao, *Accelerating Neural Network Training with Semi-Structured (2:4) Sparsity*, PyTorch, 2024. URL: `https://pytorch.org/blog/accelerating-neural-network-training/`

[56] PyTorch, *ao: PyTorch native quantization and sparsity for training and inference*, GitHub Repository, Last accessed July 2025. URL: `https://github.com/pytorch/ao`

# Appendix A

# Detailed Experimental Data

This appendix provides comprehensive experimental results across all tested configurations, including intermediate compression stages and generation examples that illustrate qualitative performance changes.

## A.1 Pruning-Only Configurations

This section examines models that have undergone only pruning operations (depth and width pruning) without additional optimization techniques.

### A.1.1 Benchmark Results

*Table A.1: Performance results of pruned models.*

| **Config** | **TriviaQA (%) ↑** | | **WikiText ↓** | **#Params** | **Size** |
|---|---|---|---|---|---|
| | **Closed** | **Open** | **PPL** | | **(GB)** |
| *Baseline Instruct* | *50.6* | *81.8* | *26.61* | *1235.8M* | *2.30* |
| Baseline Vanilla | 40.2 | 77.8 | 20.05 | 1235.8M | 2.30 |
| Depth 2 | 20.0 | 53.0 | 55.24 | 1114.2M | 2.08 |
| Depth 4 | 5.6 | 14.2 | 230.77 | 992.5M | 1.85 |
| Depth 6 | 2.0 | 2.4 | 2081.04 | 870.9M | 1.62 |
| Depth 8 | 1.2 | 1.4 | 227734.90 | 749.2M | **1.40** |
| Width 1:2 | 1.8 | 2.2 | 471.66 | 749.3M | 2.30 |

Continued on next page

Table A.1 – continued from previous page

| Config | TriviaQA (%) ↑ | | WikiText ↓ | #Params | Size |
|---|---|---|---|---|---|
| | Closed | Open | PPL | | (GB) |
| Width 2:4 | 7.0 | 17.2 | 161.10 | 749.3M | 2.30 |
| Width 3:4 | 0.4 | 0.2 | 10734.84 | 506.0M | 2.30 |
| Width 1:8 | 47.8 | 80.8 | 27.19 | 1114.2M | 2.30 |
| Width 4:8 | 12.6 | 42.4 | 81.32 | 749.3M | 2.30 |
| Width 6:8 | 0.2 | 0.2 | 3075.58 | 506.0M | 2.30 |
| Width 1:16 | **49.2** | **81.8** | **26.77** | 1175.0M | 2.30 |
| Width 8:16 | 16.2 | 52.8 | 64.08 | 749.3M | 2.30 |
| Width 12:16 | 0.2 | 0.2 | 1793.97 | 506.0M | 2.30 |
| Depth 2 + Width 1:2 | 0.4 | 0.4 | 694.34 | 688.4M | 2.08 |
| Depth 2 + Width 2:4 | 3.8 | 5.0 | 309.32 | 688.4M | 2.08 |
| Depth 2 + Width 3:4 | 0.2 | 0.2 | 38058.32 | 475.6M | 2.08 |
| Depth 2 + Width 1:8 | 20.2 | 53.6 | 57.22 | 1007.7M | 2.08 |
| Depth 2 + Width 4:8 | 6.4 | 15.2 | 185.43 | 688.4M | 2.08 |
| Depth 2 + Width 6:8 | 0.6 | 1.0 | 18260.58 | 475.6M | 2.08 |
| Depth 2 + Width 1:16 | 19.6 | 51.6 | 55.68 | 1061.0M | 2.08 |
| Depth 2 + Width 8:16 | 7.6 | 23.4 | 139.97 | 688.4M | 2.08 |
| Depth 2 + Width 12:16 | 0.2 | 0.4 | 2589.90 | 475.6M | 2.08 |
| Depth 4 + Width 1:2 | 0.6 | 0.2 | 3124.02 | 627.6M | 1.85 |
| Depth 4 + Width 2:4 | 1.0 | 2.2 | 1167.36 | 627.6M | 1.85 |
| Depth 4 + Width 3:4 | 0.8 | 0.2 | 87799.62 | 445.2M | 1.85 |
| Depth 4 + Width 1:8 | 5.2 | 14.8 | 244.69 | 901.3M | 1.85 |
| Depth 4 + Width 4:8 | 3.0 | 2.6 | 774.60 | 627.6M | 1.85 |
| Depth 4 + Width 6:8 | 0.2 | 0.2 | 63239.70 | 445.2M | 1.85 |
| Depth 4 + Width 1:16 | 4.6 | 14.6 | 235.32 | 946.9M | 1.85 |
| Depth 4 + Width 8:16 | 2.6 | 6.2 | 591.59 | 627.6M | 1.85 |
| Depth 4 + Width 12:16 | 1.0 | 0.6 | 119073.95 | 445.2M | 1.85 |
| Depth 6 + Width 1:2 | 0.6 | 0.8 | 19899.25 | 566.8M | 1.62 |
| Depth 6 + Width 2:4 | 1.0 | 1.0 | 20371.15 | 566.8M | 1.62 |
| Depth 6 + Width 3:4 | 0.4 | 0.6 | 648756.35 | 414.8M | 1.62 |
| Depth 6 + Width 1:8 | 2.0 | 1.6 | 2376.63 | 794.9M | 1.62 |
| Depth 6 + Width 4:8 | 0.8 | 1.0 | 77482.90 | 566.8M | 1.62 |
| Depth 6 + Width 6:8 | 0.2 | 0.2 | 108418.06 | 414.8M | 1.62 |
| Depth 6 + Width 1:16 | 1.8 | 1.6 | 2032.84 | 832.9M | 1.62 |
| Depth 8 + Width 1:2 | 0.4 | 0.2 | 111859.62 | 506.0M | **1.40** |

Continued on next page

**Table A.1 – continued from previous page**

| Config | TriviaQA (%) ↑ | | WikiText ↓ | #Params | Size |
|---|---|---|---|---|---|
| | Closed | Open | PPL | | (GB) |
| Depth 8 + Width 2:4 | 0.6 | 0.4 | 185871.81 | 506.0M | **1.40** |
| Depth 8 + Width 3:4 | 0.4 | 0.4 | 1044840.71 | **384.3M** | **1.40** |
| Depth 8 + Width 1:8 | 1.4 | 1.0 | 224204.19 | 688.4M | **1.40** |
| Depth 8 + Width 4:8 | 0.8 | 0.4 | 554910.71 | 506.0M | **1.40** |

# A.2 Optimized Configurations

This section presents the quantitative results for the configurations that have undergone stages of the compression pipeline after pruning, including LoRA adaptation, quantization, and EoRA compensation, as well as those that have completed the full pipeline.

## A.2.1 Benchmark Results

*Table A.2: Evaluation results of the configurations subset to which LoRA, Quantization and Eora were applied.*

| Config | LoRA | Quant | TriviaQA (%) ↑ | | WikiText ↓ | Size |
|---|---|---|---|---|---|---|
| | Type | | Closed | Open | PPL | (GB) |
| *Baseline Instruct* | *None* | *No* | *50.6* | *81.8* | *26.61* | *2.30* |
| Depth 2 | WikiText | No | 28.8 | 59.3 | 21.34 | 2.08 |
| Depth 2 | WikiText | Yes | 23.2 | 53.5 | 22.89 | 0.92 |
| Depth 2 | WikiText | EoRA | 24.7 | 57.9 | 22.63 | 0.92 |
| Depth 2 | TriviaQA | No | 27.8 | 60.2 | 51.09 | 2.08 |
| Depth 2 | TriviaQA | Yes | 23.9 | 54.6 | 55.03 | 0.92 |
| Depth 2 | TriviaQA | EoRA | 24.4 | 58.4 | 54.60 | 0.92 |
| Depth 4 | WikiText | No | 18.8 | 54.1 | 29.45 | 1.85 |
| Depth 4 | WikiText | Yes | 15.3 | 49.3 | 31.60 | 0.86 |
| Depth 4 | TriviaQA | No | 19.8 | 49.4 | 91.43 | 1.85 |
| Depth 4 | TriviaQA | Yes | 17.3 | 41.4 | 103.61 | 0.86 |
| Depth 8 | WikiText | No | 5.8 | 18.0 | 57.44 | 1.40 |
| Depth 8 | WikiText | Yes | 4.7 | 18.0 | 61.87 | **0.74** |
| Depth 8 | WikiText | EoRA | 5.0 | 17.5 | 59.50 | **0.74** |
| Depth 8 | TriviaQA | No | 9.5 | 0.5 | 10005.97 | 1.40 |
| Depth 8 | TriviaQA | Yes | 7.8 | 0.7 | 16888.28 | **0.74** |
| Depth 8 | TriviaQA | EoRA | 9.5 | 0.4 | 7320.53 | **0.74** |

Continued on next page

**Table A.2 – continued from previous page**

| Config | LoRA | Quant | TriviaQA (%) ↑ | | WikiText ↓ | Size |
|---|---|---|---|---|---|---|
| | Type | | Closed | Open | PPL | (GB) |
| Width 1:2 | WikiText | No | 10.0 | 32.0 | 40.26 | 2.30 |
| Width 1:2 | WikiText | Yes | 9.1 | 27.3 | 42.19 | 0.98 |
| Width 1:2 | WikiText | EoRA | 8.8 | 28.3 | 41.78 | 0.98 |
| Width 1:2 | TriviaQA | No | 12.4 | 29.3 | 145.54 | 2.30 |
| Width 1:2 | TriviaQA | Yes | 11.3 | 26.5 | 154.33 | 0.98 |
| Width 1:2 | TriviaQA | EoRA | 11.7 | 25.7 | 152.53 | 0.98 |
| Width 2:4 | WikiText | No | 12.6 | 45.3 | 30.63 | 2.30 |
| Width 2:4 | WikiText | Yes | 12.3 | 44.4 | 32.29 | 0.98 |
| Width 2:4 | TriviaQA | No | 15.8 | 48.6 | 90.72 | 2.30 |
| Width 2:4 | TriviaQA | Yes | 14.8 | 48.2 | 97.33 | 0.98 |
| Width 1:8 | WikiText | No | **47.8** | **80.8** | **15.70** | 2.30 |
| Width 1:8 | WikiText | Yes | 39.0 | 74.8 | 16.85 | 0.98 |
| Width 1:8 | WikiText | EoRA | 40.0 | 78.4 | 16.68 | 0.98 |
| Width 1:8 | TriviaQA | No | 44.9 | 79.4 | 30.81 | 2.30 |
| Width 1:8 | TriviaQA | Yes | 41.3 | 77.2 | 33.38 | 0.98 |
| Width 1:8 | TriviaQA | EoRA | 42.0 | 77.2 | 32.86 | 0.98 |
| Width 4:8 | WikiText | No | 17.0 | 51.1 | 26.04 | 2.30 |
| Width 4:8 | WikiText | Yes | 15.7 | 45.3 | 27.40 | 0.98 |
| Width 4:8 | TriviaQA | No | 19.0 | 56.8 | 64.33 | 2.30 |
| Width 4:8 | TriviaQA | Yes | 17.7 | 57.1 | 67.95 | 0.98 |
| Width 8:16 | WikiText | No | 19.4 | 56.3 | 24.04 | 2.30 |
| Width 8:16 | WikiText | Yes | 17.5 | 52.1 | 25.39 | 0.98 |
| Width 8:16 | WikiText | EoRA | 17.8 | 54.9 | 25.09 | 0.98 |
| Width 8:16 | TriviaQA | No | 20.6 | 57.6 | 56.11 | 2.30 |
| Width 8:16 | TriviaQA | Yes | 20.8 | 55.7 | 60.67 | 0.98 |
| Width 8:16 | TriviaQA | EoRA | 21.3 | 54.0 | 59.73 | 0.98 |
| Depth 2 + Width 1:16 | WikiText | No | 28.1 | 61.8 | 21.42 | 2.08 |
| Depth 2 + Width 1:16 | WikiText | Yes | 22.9 | 55.8 | 23.03 | 0.92 |
| Depth 2 + Width 1:16 | WikiText | EoRA | 23.3 | 57.3 | 22.72 | 0.92 |
| Depth 2 + Width 1:16 | TriviaQA | No | 28.0 | 59.5 | 50.99 | 2.08 |
| Depth 2 + Width 1:16 | TriviaQA | Yes | 25.2 | 56.3 | 55.46 | 0.92 |
| Depth 2 + Width 1:16 | TriviaQA | EoRA | 24.8 | 57.4 | 54.81 | 0.92 |
| Depth 2 + Width 1:8 | WikiText | No | 26.5 | 63.8 | 21.76 | 2.08 |
| Depth 2 + Width 1:8 | WikiText | Yes | 24.8 | 58.1 | 23.21 | 0.92 |
| Depth 2 + Width 1:8 | WikiText | EoRA | 22.3 | 59.6 | 22.98 | 0.92 |
| Depth 2 + Width 1:8 | TriviaQA | No | 27.4 | 60.4 | 51.89 | 2.08 |
| Depth 2 + Width 1:8 | TriviaQA | Yes | 24.0 | 54.8 | 54.81 | 0.92 |
| Depth 2 + Width 1:8 | TriviaQA | EoRA | 24.3 | 58.3 | 55.46 | 0.92 |
| Depth 2 + Width 2:4 | WikiText | No | 11.5 | 31.3 | 39.33 | 2.08 |
| Depth 2 + Width 2:4 | WikiText | Yes | 10.6 | 30.8 | 41.54 | 0.92 |
| Depth 2 + Width 2:4 | WikiText | EoRA | 10.4 | 29.8 | 40.73 | 0.92 |
| Depth 2 + Width 2:4 | TriviaQA | No | 11.1 | 29.6 | 146.68 | 2.08 |
| Depth 2 + Width 2:4 | TriviaQA | Yes | 11.6 | 29.4 | 157.98 | 0.92 |
| Depth 2 + Width 2:4 | TriviaQA | EoRA | 11.2 | 28.4 | 154.33 | 0.92 |

Continued on next page

**Table A.2 – continued from previous page**

| Config | LoRA | Quant | TriviaQA (%) ↑ | | WikiText ↓ | Size |
|---|---|---|---|---|---|---|
| | Type | | Closed | Open | PPL | (GB) |
| Depth 2 + Width 3:4 | WikiText | No | 1.2 | 5.3 | 134.60 | 2.08 |
| Depth 2 + Width 3:4 | WikiText | Yes | 1.3 | 5.8 | 139.97 | 0.92 |
| Depth 2 + Width 3:4 | WikiText | EoRA | 0.8 | 5.2 | 136.72 | 0.92 |
| Depth 2 + Width 3:4 | TriviaQA | No | 2.1 | 0.5 | 3148.52 | 2.08 |
| Depth 2 + Width 3:4 | TriviaQA | Yes | 2.3 | 0.7 | 3980.10 | 0.92 |
| Depth 2 + Width 3:4 | TriviaQA | EoRA | 2.1 | 0.7 | 3431.06 | 0.92 |
| Depth 2 + Width 4:8 | WikiText | No | 11.8 | 44.0 | 34.10 | 2.08 |
| Depth 2 + Width 4:8 | WikiText | Yes | 10.6 | 42.6 | 35.81 | 0.92 |
| Depth 2 + Width 4:8 | TriviaQA | No | 14.7 | 45.0 | 116.95 | 2.08 |
| Depth 2 + Width 4:8 | TriviaQA | Yes | 13.8 | 43.0 | 124.98 | 0.92 |
| Depth 2 + Width 8:16 | WikiText | No | 12.8 | 41.0 | 31.91 | 2.08 |
| Depth 2 + Width 8:16 | WikiText | Yes | 12.3 | 42.0 | 33.51 | 0.92 |
| Depth 2 + Width 8:16 | TriviaQA | No | 16.6 | 48.8 | 96.95 | 2.08 |
| Depth 2 + Width 8:16 | TriviaQA | Yes | 15.3 | 46.2 | 103.61 | 0.92 |
| Depth 4 + Width 1:16 | WikiText | No | 18.1 | 53.2 | 29.51 | 1.85 |
| Depth 4 + Width 1:16 | WikiText | Yes | 15.2 | 48.6 | 31.66 | 0.86 |
| Depth 4 + Width 1:16 | TriviaQA | No | 18.7 | 47.3 | 89.32 | 1.85 |
| Depth 4 + Width 1:16 | TriviaQA | Yes | 16.8 | 39.3 | 100.42 | 0.86 |
| Depth 4 + Width 1:8 | WikiText | No | 17.9 | 52.3 | 29.98 | 1.85 |
| Depth 4 + Width 1:8 | WikiText | Yes | 14.5 | 45.6 | 32.79 | 0.86 |
| Depth 4 + Width 1:8 | TriviaQA | No | 19.1 | 47.5 | 90.72 | 1.85 |
| Depth 4 + Width 1:8 | TriviaQA | Yes | 17.1 | 38.3 | 99.25 | 0.86 |
| Depth 4 + Width 2:4 | WikiText | No | 7.7 | 27.0 | 49.71 | 1.85 |
| Depth 4 + Width 2:4 | WikiText | Yes | 7.2 | 25.8 | 52.20 | 0.86 |
| Depth 4 + Width 2:4 | WikiText | EoRA | 7.6 | 26.3 | 51.19 | 0.86 |
| Depth 4 + Width 2:4 | TriviaQA | No | 9.6 | 15.9 | 298.63 | 1.85 |
| Depth 4 + Width 2:4 | TriviaQA | Yes | 8.5 | 16.4 | 319.14 | 0.86 |
| Depth 4 + Width 2:4 | TriviaQA | EoRA | 8.6 | 17.3 | 308.11 | 0.86 |
| Depth 4 + Width 3:4 | WikiText | No | 0.8 | 4.3 | 156.76 | 1.85 |
| Depth 4 + Width 3:4 | WikiText | Yes | 1.1 | 4.6 | 164.28 | 0.86 |
| Depth 4 + Width 3:4 | TriviaQA | No | 1.7 | 0.5 | 3652.35 | 1.85 |
| Depth 4 + Width 3:4 | TriviaQA | Yes | 1.9 | 0.3 | 4545.42 | 0.86 |
| Depth 4 + Width 4:8 | WikiText | No | 8.9 | 33.2 | 44.04 | 1.85 |
| Depth 4 + Width 4:8 | WikiText | Yes | 8.2 | 29.6 | 46.34 | 0.86 |
| Depth 4 + Width 4:8 | WikiText | EoRA | 8.3 | 31.1 | 45.53 | 0.86 |
| Depth 4 + Width 4:8 | TriviaQA | No | 10.5 | 20.8 | 210.94 | 1.85 |
| Depth 4 + Width 4:8 | TriviaQA | Yes | 11.1 | 19.1 | 226.30 | 0.86 |
| Depth 4 + Width 4:8 | TriviaQA | EoRA | 11.6 | 18.0 | 219.34 | 0.86 |
| Depth 6 + Width 1:16 | WikiText | No | 11.5 | 25.6 | 37.82 | 1.62 |
| Depth 6 + Width 1:16 | WikiText | Yes | 9.6 | 22.8 | 40.73 | 0.80 |
| Depth 6 + Width 1:16 | WikiText | EoRA | 9.4 | 24.8 | 39.63 | 0.80 |
| Depth 6 + Width 1:16 | TriviaQA | No | 16.8 | 27.5 | 310.53 | 1.62 |
| Depth 6 + Width 1:16 | TriviaQA | Yes | 14.6 | 25.8 | 269.79 | 0.80 |
| Depth 6 + Width 1:16 | TriviaQA | EoRA | 15.8 | 25.4 | 292.86 | 0.80 |

Continued on next page

**Table A.2 – continued from previous page**

| Config | LoRA | Quant | TriviaQA (%) ↑ | | WikiText ↓ | Size |
|---|---|---|---|---|---|---|
| | Type | | Closed | Open | PPL | (GB) |
| Depth 6 + Width 4:8 | WikiText | No | 0.9 | 9.6 | 55.46 | 1.62 |
| Depth 6 + Width 4:8 | WikiText | Yes | 2.1 | 10.7 | 58.80 | 0.80 |
| Depth 6 + Width 4:8 | TriviaQA | No | 8.9 | 16.2 | 568.93 | 1.62 |
| Depth 6 + Width 4:8 | TriviaQA | Yes | 7.5 | 14.0 | 515.99 | 0.80 |
| Depth 6 + Width 6:8 | WikiText | No | 1.7 | 5.8 | 139.97 | 1.62 |
| Depth 6 + Width 6:8 | WikiText | Yes | 1.3 | 3.9 | 153.72 | 0.80 |
| Depth 6 + Width 6:8 | TriviaQA | No | 2.2 | 0.7 | 3273.94 | 1.62 |
| Depth 6 + Width 6:8 | TriviaQA | Yes | 2.2 | 0.7 | 3457.97 | 0.80 |
| Depth 8 + Width 1:8 | WikiText | No | 5.8 | 16.8 | 58.58 | 1.40 |
| Depth 8 + Width 1:8 | WikiText | Yes | 5.3 | 14.9 | 63.33 | **0.74** |
| Depth 8 + Width 1:8 | TriviaQA | No | 8.7 | 1.3 | 132836.59 | 1.40 |
| Depth 8 + Width 1:8 | TriviaQA | Yes | 3.8 | 0.3 | 126753.56 | **0.74** |
| Depth 8 + Width 3:4 | WikiText | No | 0.2 | 1.3 | 278.36 | 1.40 |
| Depth 8 + Width 3:4 | WikiText | Yes | 0.5 | 1.3 | 289.45 | **0.74** |
| Depth 8 + Width 3:4 | TriviaQA | No | 0.3 | 0.3 | 7263.56 | 1.40 |
| Depth 8 + Width 3:4 | TriviaQA | Yes | 0.8 | 0.2 | 9326.59 | **0.74** |

## A.2.2 Examples of Generated Text

To illustrate the qualitative differences between configurations, Table A.3 contains several examples of text generated by some of these configurations. Each model has generated the text based on the prompt “`Question: Who was the man behind The Chipmunks?\nAnswer:`”.

The configurations are labeled using the initial letter of the name of the procedure (i.e. D[L] for depth pruning of L layers, W[N]:[M] for width pruning with N:M ratio, W and T for WikiText and TriviaQA LoRA fine-tuning respectively, Q for quantization and E for EoRA).

*Table A.3: Sample answers generated by selected model configurations.*

| Config | Generated Text |
|---|---|
| Baseline Instruct | Ross Bagdasarian Sr. was an American singer, songwriter, and record producer who was known for his hit song ”The Chipmunk Song [..] |
| W1:2 + T | John Paul (disambiguation) - The Chipmunk’s Hat (disambiguation) - The Chipmunk’s Hat (disambiguation) - The Chipmunk[..] |
| W1:2 + T + Q | The Chipmunk (disambiguation) is the English title of which film? |
| W1:2 + T + Q + E | John Paul (disambiguation) - The Chipmunks - The Chipmunks (disambiguation) - The Chipmunks - The Chipmunks - The Chi[..] |

Continued on next page

**Table A.3 – continued from previous page**

| Config | Generated Text |
|---|---|
| W1:2 + W | The Chipmunks are the only group of the world's most popular and influential rock bands. The group is known for its u[..] |
| W1:2 + W + Q | The Chipmunks are a group of children who are born in the early 20th century, who are the descendants of the family o[..] |
| W1:2 + W + Q + E | The Chipmunks are the only group of children who are known to be the only ones who have been born in the same way as [..] |
| W1:8 + T | Ross Bagdasarian Jr. (disambiguation) - film producer and singer. He was born Ross Bagdasarian, Jr. in 1936 and died [..] |
| W1:8 + W | Ross Bagdasarian Jr., also known as David Seville, is the voice behind the Chipmunks. He was born on May 24, 1943, in[..] |
| W1:8 + W + Q | The Chipmunk Man, also known as The Chipmunk Man, is a 2005 American comedy film directed by Chris Weigman and writte[..] |
| W4:8 + T | John Brown (disambiguation) (film) character. (disambiguation) John Brown (disambiguation) (film) character. (disambi[..] |
| W4:8 + T + Q | John Brown (disambiguation) - Wikipedia disambiguation page is for John Brown. He was the main character in the Chipm[..] |
| W4:8 + W | The Chipmunks were created by the British cartoonist and writer Arie Bellosi. |
| W8:16 + T + Q | Dave Barry (disambiguation) and Neil Richardson (disambiguation) are both the founders of the company, but Dave Barry[..] |
| W8:16 + W + Q | The Chipmunks were created by the British animation studio Aardust Productions. The studio was founded in 1991 by Pet[..] |
| D2 + T + Q | Don Quibberta of the Chipmunks. Chipmunk song. Chipmunk song. [..] |
| D2 + W | The Chipmunks were created by Tom Pollock, a British animator and cartoonist. Pollock was inspired by the 1950s carto[..] |
| D2 + W1:16 + T + Q + E | Richard E. Young (disambiguation) - Wikipedia (disambiguation) - Wikipedia (disambiguation) - Wikipedia (disambiguati[..] |
| D2 + W1:16 + W | The Chipmunks were created by Tom Pollock, a British animator and cartoonist. Pollock was inspired by the 1950s carto[..] |
| D2 + W1:16 + W + Q | James Howard Morris, a British actor, was the main inspiration behind the character of Chip's father, who is the main[..] |
| D2 + W1:8 + W | The Chipmunks was created by Matt Williams, a former animator and voice actor for the Nickelodeon cartoon series The [..] |
| D2 + W1:8 + W + Q | James T. Green ( 1969 – 1980 ) |
| D2 + W1:8 + W + Q + E | The Chipmunks was created by the British cartoonist and animator, James T. Landry. He was inspired by the 1960s carto[..] |
| D2 + W2:4 + T + Q | The Chipmunks (disambiguation) (disambiguation) (reality test) (disambiguation) (reality test) (disambiguation) (disa[..] |
| D2 + W2:4 + W + Q | The Chipmunks were the " chipmunks ", a group of children who were the victims of the 1980s @-@ 80s. The Chipmunks we[..] |
| D2 + W3:4 + T | John James (disambiguation) (disambiguation) [..] |
| D2 + W3:4 + T + Q | The Chipcals (disambiguation) (disambiguation) [..] |
| D2 + W3:4 + T + Q + E | John James, John James, [..] |
| D2 + W3:4 + W | The Chipmunks's first answer, the first answer, [..] |
| D2 + W3:4 + W + Q | The Chipmunks's first two @-@ @-@ two [..] |
| D2 + W3:4 + W + Q + E | The Chipmunks's first answer, The Chipmunks's first answer, [..] |

Continued on next page

**Table A.3 – continued from previous page**

| Config | Generated Text |
|---|---|
| D2 + W4:8 + T | John C. Smith, Jr. (disambiguation) - Chipmunk Jr. (disambiguation) - Chipmunk Jr. (disambiguation) - Chipmunk Jr. (d[..] |
| D4 + W | The Chipmunks were created by Canadian comedian and musician John L. Linder, who was a member of the Canadian comedy [..] |
| D4 + W + Q | The Chipmunks were the men behind the Chipmunks cartoon series. Chipy, the Chipmunks'main character, was a chipster w[..] |
| D4 + W3:4 + T | John (disambiguation) (disambiguation) (disambiguation) (disambiguation) (disdisdisiguation) (disdisiguation) (disdis[..] |
| D4 + W3:4 + T + Q | The 1saurus (disambiguation) is the name of the world's highest point of the world? |
| D4 + W3:4 + W | The first question of the question, the question was the question of the question, and the question of the question w[..] |
| D4 + W3:4 + W + Q | The first question, the question was the question of the question, and the question was not not so much of the questi[..] |
| D4 + W4:8 + W | The Chipmunk was a young boy who was born in 1950. He was a young boy who was a member of the British Royal Navy, and[..] |
| D4 + W4:8 + W + Q | The Chipmunk was a 2000 film directed by the film director John Kostin. The film was directed by the film director Jo[..] |
| D6 + W1:16 + T + Q + E | Charles Dickens (disambiguation) and Dr. Jolly Jolly (disambiguation) and Dr. M. Snow (fiction) and Dr. Snow (fiction[..] |
| D6 + W1:16 + W | The Chipmunks, the original version of the cartoon, was created by the Canadian cartoonist John H. Griffith, who was [..] |
| D6 + W1:16 + W + Q | The Chipmunks, the original version of the cartoon, was created by the Canadian cartoonist Jeff Sarnick and the Ameri[..] |
| D6 + W3:4 + T | The Great Road (disambiguation) (disambiguation) (disambiguation)) (disambiguation)) (disambigig) and aressststlylylyl[..] |
| D6 + W3:4 + T + Q | The World World World World World [..] |
| D6 + W4:8 + T | John F. F. F. B. W. W. [..] |
| D6 + W6:8 + T | John O. (disambiguation) is the name of the 1990s film 'The Great Man'? |
| D6 + W6:8 + T + Q | The Chip Chip, the Chip Chip, is the name for the first body of which body? |
| D6 + W6:8 + W | The Chipmunks of the 2000s were the first of the 2000s in the 2000s. The 2000s were [..] |
| D6 + W6:8 + W + Q | The Chipmunks of the 2000s were the first of the 2000s in the 2000s. The Chipimunks of the 2000s [..] |
| D8 + W | The Chipmunks, the first child of the family, was the first child of the [..] |
| D8 + W + Q | The Chipmunks were the first to record the first recording of the song " The Chipmunks ". The song was recorded by th[..] |
| D8 + W + Q + E | The Chipmunks were the first to be recorded in the 1970s. The Chipmunks were the first to be recorded in the 1970s. T[..] |
| D8 + W1:8 + W | The Chipmunks were the first to be recorded in the 1980s by the BBC. The Chipmunks were the first to be recorded in t[..] |
| D8 + W3:4 + T | John (disambiguation) and the film of the film The The[..] |
| D8 + W3:4 + T + Q | The Black Black Black Black Black [..] |
| D8 + W3:4 + W | the first first @-@ 5 @-@ 5 @-@ 5 [..] |
| D8 + W3:4 + W + Q | the 11thththththththththththththththth[..] |

# Appendix B

# Alternative LoRA Integration Strategies

This appendix examines the effects of applying LoRA adaptation earlier in the compression pipeline, specifically after depth pruning but before width pruning. The idea behind this experiment is that this modified sequence should decouple the two pruning methods, and influence the interference between LoRA weight modifications and WANDA's importance-based pruning decisions.

### B.0.1 Experimental Results

The evaluation encompasses 12 different configurations tested with LoRA adapters trained on both WikiText and TriviaQA datasets. The complete results for the Depth→LoRA→Width pipeline are illustrated in Table B.1. Although limited in scope compared to the comprehensive experiments presented in the main chapters, these preliminary results provide a foundation for future systematic investigations into optimal technique sequencing.

*Table B.1: Evaluation results for alternatively ordered Depth→LoRA→Width pipeline, which combines depth pruning, LoRA fine-tuning, and width pruning in sequential order.*

| **Config** | **LoRA** | **TriviaQA (%) ↑** | | **WikiText ↓** |
|---|---|---|---|---|
| | **Type** | **Closed** | **Open** | **PPL** |
| *Baseline Instruct* | *No* | *50.6* | *81.8* | *26.61* |
| Depth 2 + Width 1:8 | WikiText | 26.3 | 58.8 | 21.93 |
| Depth 2 + Width 1:8 | TriviaQA | **28.2** | 59.3 | 52.51 |
| Depth 2 + Width 1:16 | WikiText | 27.6 | 59.4 | **21.42** |
| Depth 2 + Width 1:16 | TriviaQA | 27.2 | **59.6** | 51.29 |
| Depth 2 + Width 4:8 | WikiText | 8.3 | 29.9 | 49.33 |
| Depth 2 + Width 4:8 | TriviaQA | 9.8 | 29.3 | 139.97 |
| Depth 2 + Width 6:8 | WikiText | 0.3 | 0.3 | 1583.17 |
| Depth 2 + Width 6:8 | TriviaQA | 0.4 | 0.1 | 3652.35 |
| Depth 6 + Width 1:8 | WikiText | 10.7 | 22.2 | 38.87 |
| Depth 6 + Width 1:8 | TriviaQA | 14.6 | 24.1 | 324.16 |
| Depth 6 + Width 1:16 | WikiText | 11.4 | 23.6 | 37.97 |
| Depth 6 + Width 1:16 | TriviaQA | 15.6 | 26.1 | 347.78 |
| Depth 6 + Width 4:8 | WikiText | 5.3 | 12.4 | 85.89 |
| Depth 6 + Width 4:8 | TriviaQA | 5.9 | 5.1 | 513.98 |
| Depth 6 + Width 6:8 | WikiText | 0.1 | 0.3 | 3623.92 |
| Depth 6 + Width 6:8 | TriviaQA | 0.1 | 0.1 | 87116.36 |
| Depth 8 + Width 1:8 | WikiText | 5.1 | 15.3 | 58.80 |
| Depth 8 + Width 1:8 | TriviaQA | 8.6 | 0.4 | 2610.21 |
| Depth 8 + Width 1:16 | WikiText | 5.4 | 17.6 | 57.67 |
| Depth 8 + Width 1:16 | TriviaQA | 9.5 | 0.5 | 4270.03 |
| Depth 8 + Width 4:8 | WikiText | 3.6 | 10.6 | 127.94 |
| Depth 8 + Width 4:8 | TriviaQA | 3.3 | 2.3 | 1088.10 |
| Depth 8 + Width 6:8 | WikiText | 0.1 | 0.2 | 2877.98 |
| Depth 8 + Width 6:8 | TriviaQA | 0.3 | 0.1 | 37762.15 |

### B.0.2 Comparison and Brief Analysis

The preliminary comparison between the *Depth→LoRA→Width* (D→L→W) and *Depth→Width→LoRA* (D→W→L) pipeline orderings, as presented in Table B.2, reveals interesting patterns in how sequencing affects results overall.

The D→W→L pipeline demonstrates superior performance compared to the conventional D→L→W approach, particularly when width pruning becomes more aggressive. This expected advantage stems from LoRA's ability to directly address the quality degradation caused by width pruning when applied to the compressed architecture.

*Table B.2: Comparison of pruning pipeline orders: Depth→LoRA→Width (D→L→W) vs. Depth→Width→LoRA (D→W→L). The configurations naming follows the same notation described in Section A.2.2.*

| Config | LoRA Type | D→L→W | | | D→W→L | | |
|---|---|---|---|---|---|---|---|
| | | TriviaQA (%) ↑ | | WikiText | TriviaQA (%) ↑ | | WikiText |
| | | Closed | Open | PPL ↓ | Closed | Open | PPL ↓ |
| D2 + W1:8 | WikiText | 26.3 | 58.8 | 21.93 | 26.5 | **63.8** | 21.76 |
| D2 + W1:8 | TriviaQA | **28.2** | 59.3 | 52.51 | 27.4 | 60.4 | 51.89 |
| D2 + W1:16 | WikiText | 27.6 | 59.4 | **21.42** | **28.1** | 61.8 | **21.42** |
| D2 + W1:16 | TriviaQA | 27.2 | **59.6** | 51.29 | 28.0 | 59.5 | 50.99 |
| D2 + W4:8 | WikiText | 8.3 | 29.9 | 49.33 | 11.8 | 44.0 | 34.10 |
| D2 + W4:8 | TriviaQA | 9.8 | 29.3 | 139.97 | 14.7 | 45.0 | 116.95 |
| D6 + W1:16 | WikiText | 11.4 | 23.6 | 37.97 | 11.5 | 25.6 | 37.82 |
| D6 + W1:16 | TriviaQA | 15.6 | 26.1 | 347.78 | 16.8 | 27.5 | 310.53 |
| D6 + W4:8 | WikiText | 5.3 | 12.4 | 85.89 | 0.9 | 9.6 | 55.46 |
| D6 + W4:8 | TriviaQA | 5.9 | 5.1 | 513.98 | 8.9 | 16.2 | 568.93 |
| D6 + W6:8 | WikiText | 0.1 | 0.3 | 3623.92 | 1.7 | 5.8 | 139.97 |
| D6 + W6:8 | TriviaQA | 0.1 | 0.1 | 87116.36 | 2.2 | 0.7 | 3273.94 |
| D8 + W1:8 | WikiText | 5.1 | 15.3 | 58.80 | 5.8 | 16.8 | 58.58 |
| D8 + W1:8 | TriviaQA | 8.6 | 0.4 | 2610.21 | 8.7 | 1.3 | 132836.59 |

The advantage of D→W→L becomes most pronounced in heavily pruned configurations: when width pruning is aggressive, LoRA's ability to recover performance is severely compromised if applied first, which makes post-compression adaptation the more effective approach. On the other hand, in moderately pruned variants, the difference of results between the two pipeline orderings is not as significant. When width pruning ratios are conservative (such as 1:16), the destructive impact is minimal, thus the sequencing order becomes less critical for overall accuracy preservation.

What is particularly surprising, however, is that the D→L→W pipeline still maintains relatively good performance even when LoRA fine-tuning precedes aggressive width pruning. This resilience further proves the idea, which was also presented in Section 4.1, that the primary source of redundancy in these compact models lies within individual layers rather than across the layers themselves. The fact that subsequent width pruning does not catastrophically destroy the LoRA-adapted representations suggests that the intra-layer parameter space contains sufficient redundancy to absorb significant compression while preserving the essential adaptations learned during fine-tuning.

# Acknowledgements

After this long journey, I can't believe I'm finally closing this chapter of my life. The past few years have been filled with big accomplishments and just as big disappointments, which have definitely made me rethink my priorities and grow as a person. At the same time, the last 10 months have made all the difference: a fire has reignited in my soul and I've created some of the best projects I've ever made, work I'm really proud of. Now I have this insatiable hunger for greatness, and I can't wait to show the world what I'm capable of when I put my mind to it.

Now, shifting focus to the people who made this possible. The following section is for those I want to thank and who I'm incredibly grateful to know. Buckle up, because this is gonna be a long one.

I want to thank Professor Conti, who supervised this thesis and shared invaluable insights on the best practices for model optimization. We're perfectly aligned in our vision, since making AI efficient is a topic I hold very dear to my heart.

Next, I want to thank Luca Bompani, an amazing co-supervisor. He helped me a lot with both the thesis and the project, while also welcoming my ideas and providing valuable criticism when I was at my peak Dunning-Kruger, which I greatly appreciated.

Obviously, I thank my parents, who gave me unconditional support along the way, and my brother. This thesis is dedicated to them.

A special thank you goes to my friend Leon Pizzaroli. He is probably one of the best people anyone could ever meet, and he had my back whenever I was at my lowest. I genuinely wish for everyone to know someone as kind, gentle and *so chill* as him. Of course I also thank Tony, Giaco, Donnoh, Fred, Massaro and

Kevin: they're all really great guys with remarkable energy who led me to some very strange adventures (in a good way!).

I am grateful to Galf, my friend since high school with whom I've shared many beautiful moments, and Giovanni a.k.a. Profex, who is literally the most talented chemist in Italy.

A big thank you goes to my calisthenics group, "Bimbe di Ruggio". Those guys are genuinely amazing people who gave me a ton of motivation and have occupied a reasonable chunk of the last 3 years, while keeping me fit at the same time!

Now this paragraph is dedicated to the 2 months I spent at CERN in the summer of 2023, which have been the best months of my life so far. I want to thank all the "Choccy Guys" and "Cool Guys" from CERN. I thank Guilherme, my supervisor there who was not just a great mentor, but a very good friend as well. A huge thank you to Alina, Aya, Bruno, Daniel, Eliacim, Gadea, Guilherme (not the supervisor, this is a different one), Henar, Inés, Jessica, Joaquin, Jordi, Luis, Matìas, Matthias, Melike, Niko, Omar, Pablo, Rim, Stefan, Thomas, and many others. It's incredible how some people can change your life so much just by knowing them for such a short time.

Last, but definitely not least, I want to thank Letizia, who is the greatest travel companion anyone could have, and who is probably the most kind-hearted person in the world. She's incredibly caring and has been there for me through everything. She's also the de facto third supervisor of this thesis. Thank you very much Leti.

Wow, that *really* was quite long, and there are so many more I could mention. But sitting here, I realize something: all those disappointments, all those moments I wanted to quit, they all led me to this moment, and these incredible people made every struggle worth it. And this? This is just the beginning.

Let's see what this "greatness" is all about.