Alma Mater Studiorum · Università di Bologna

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ARTIFICIAL INTELLIGENCE

Optimizing Small Language Models: An Experimental Investigation in Compressing Distilled LLaMA-Based Models

Supervisor:
Prof. Francesco Conti

Co-supervisor:
Luca Bompani

Presented by:
Angelo Galavotti

Session I
Academic Year 2024/2025

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

Abstract

Pending

Contents

List of Figures

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 struggles to process effectively.

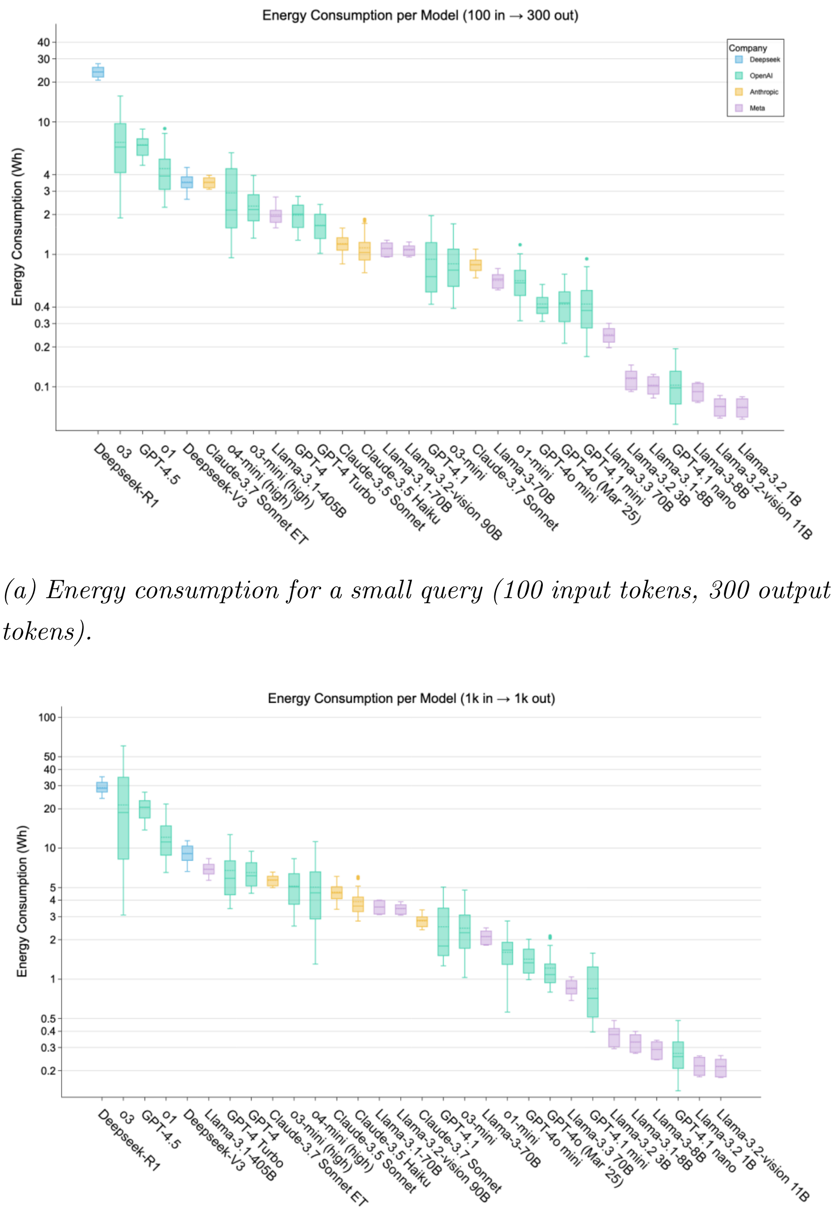
These environmental costs scale directly with model size and usage frequency, creating a fundamental tension between AI capabilities and sustainability.

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



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

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

Figure 1.1: Comparison of the 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].

compression techniques. We begin with a distilled model, a 1B parameter variant of LLaMA 3 [30], 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 memory usage and compute requirements. LoRA, meanwhile, 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 adapted to new tasks or domains with minimal overhead.

The reason behind the focus on maximizing memory efficiency is related to the fact that memory represents the most expensive constraint in our target deployment scenario, where the intended hardware platform consists of a low-power RISC-V SoC with severely constrained memory resources (further details on the target hardware can be found in Section 2.4). Crucially, optimizations that reduce memory footprint also translate directly into computational complexity improvements, creating a dual benefit for resource-constrained environments.

1.3.2 How Can Optimization Techniques Help?

Optimization techniques tackle the environmental, social, and infrastructure problems that come with large-scale LLMs. When models run more efficiently, they need far less power for each query. Smaller models can actually 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

Will write this when the document is finished.

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 in (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 represent an evolution of the *Multi-Layer Perceptron* (MLP), and incorporate 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 long-term 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 and context understanding, especially when compared to newer architectures such as Transformers

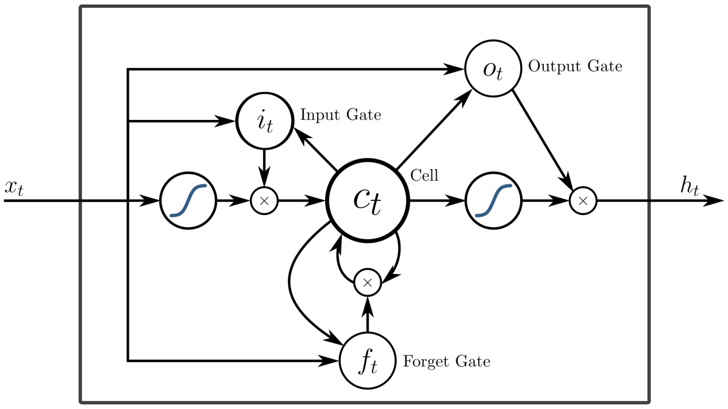


Figure 2.1: Architecture of an LSTM cell showing the three gating mechanisms. The input gate (left) controls information incorporation, the forget gate (center) manages memory retention, and the output gate (right) regulates information flow to subsequent layers. The cell state (horizontal line) maintains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

(see Section 2.2). Nevertheless, LSTMs are still widely used for various *natural language processing* (NLP) tasks, including text generation [13].

2.1.1 Tokenization

Neural networks operate exclusively with numerical data, thus textual input needs to be converted into vector representations before processing. 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 abandoning recurrent connections entirely 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).

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?"

- **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 ±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: $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$. 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 block. 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.). To achieve this, the input matrices Q, K, V are projected h times using different learned linear projections:

$$(QW_i^Q, KW_i^K, VW_i^V) \tag{2.2}$$

Given this, each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

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

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

Each head uses different projection matrices $W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$, where typically $d_k = d_v = d_\text{model}/h$ for h heads.

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. 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:

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

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

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 an external input (encoder-decoder models) or 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, modeling the joint distribution $p(x_1, x_2, \ldots, x_n)$ autoregressively. 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 the ones employed in large language models such as GPT [8] and, of course, LLaMA [18].

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 2.2 shows a comparison of encoder-decoder and decoder-only transformer architectures.

2.3 Introducing the LLaMA Family of Models

The *Large Language Model Meta AI* (LLaMA) family [18], developed by Meta AI, comprises a series of transformer-based autoregressive language models designed as competitors to OpenAI's GPT model family. The original LLaMA models were introduced in early 2023, followed by LLaMA 2 [19] later that year and LLaMA 3 [20] 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] [20].

The main feature of LLaMA, however, is 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 experiment with, fine-tune, and deploy large-scale language models without relying on closed systems. This openness has led to a proliferation of derivative models, and thus to the making of the project central to this thesis.

2.4 A Look at the 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. The envisioned

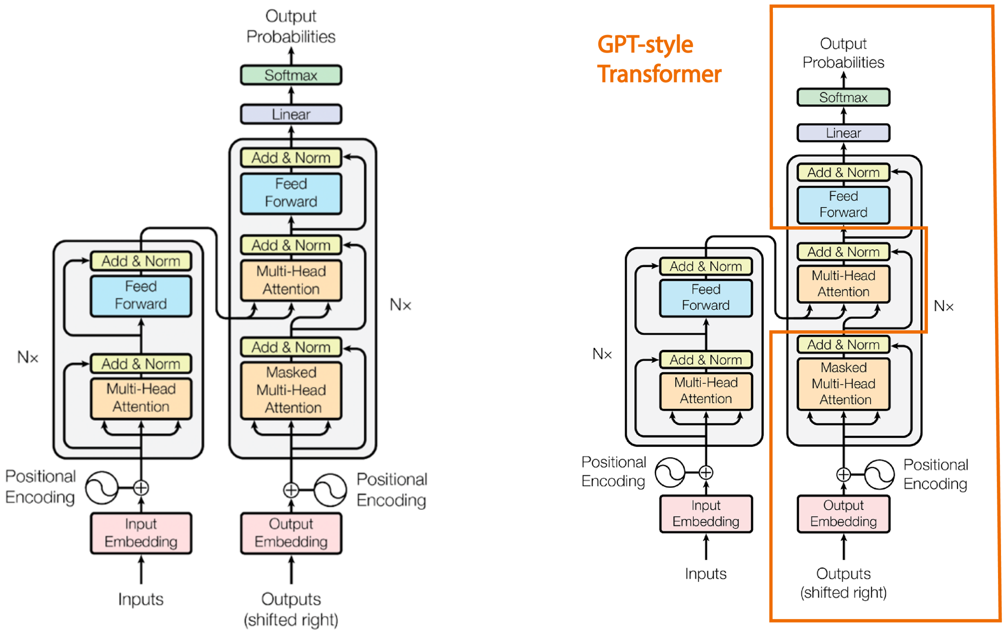


Figure 2.2: On the left, a standard Transformer architecture, which consists of an encoder and a decoder. On the right, we can observe the same architecture, while highlighting the only the layers used by a GPT-style model, which is a decoder-only transformer. The main difference is that the decoder does not attend to the encoder's output, allowing for auto-regressive generation. The image is a modified version of the one found on Vaswani et al. research paper [15].

platform is Siracusa [21], a heterogeneous RISC-V *System-on-Chip* (SoC) fabricated in 16 nm CMOS technology, which exemplifies the stringent resource constraints typical of edge XR devices designed for *Extended Reality* (XR) applications. 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 2.3.

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.

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

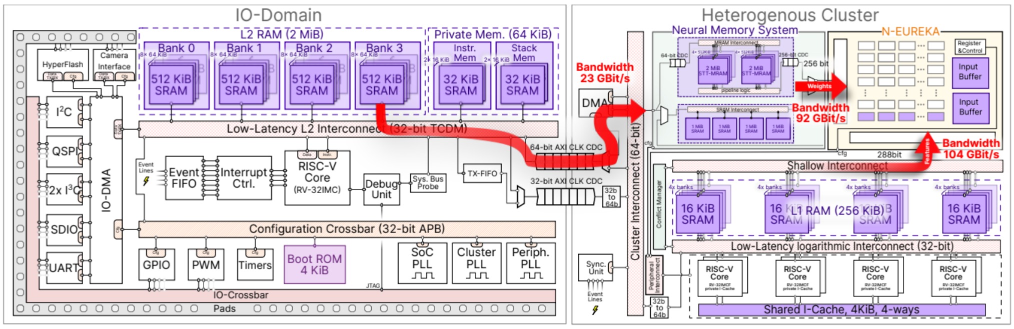


Figure 2.3: A visual representation of Siracusa's architecture. On the upper right we can see the N-EUREKA accelerator, while on the lower right the RISC-V cluster is visible. This image was sourced from the original paper [21].

practical deployment on 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 (further explored in Section 5.2).

2.5.1 Knowledge Distillation

Knowledge distillation [22] 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) [23] 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. This 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 its central importance to this work, pruning 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) [24] 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. 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.

- The width-pruning targets differ significantly: while 2SSP focuses on FFN components, the approach presented in this thesis targets all attention weight matrices. Additionally, the structured width-pruning strategies are entirely distinct: 2SSP removes complete rows and columns by slicing weight matrices rather than using ratio-based structural pruning (see Section 3.3.2).

- The method for computing layer importance during depth-pruning employs a compound metric rather than relying solely on perplexity-based measures (see Section 3.3.1).

Another promising approach to structured pruning is demonstrated by *Sheared LLaMA* [25], 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.

This chapter begins with preliminary experiments on layer manipulation that revealed crucial insights about architectural redundancy in transformer models and shaped the main compression strategy. The chapter 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 practical implementation details. Finally, the evaluation methods and their underlying motivation are detailed.

3.1 Preliminary Research: Franken-LLaMA

Before embarking on systematic compression techniques for our target 1B parameter LLaMA model, we conducted preliminary research to understand the behavior of transformer architectures under structural modifications. This exploratory consisted on the “Franken-LLaMA” project [26], which involved experimenting with selective layer skipping and repetition in the larger LLaMA2-7B-Chat [19] model 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 [27], with modifications to Hugging Face's transformers library [28] 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. This weight sharing approach aligned directly with the target hardware constraints outlined in Section 2.4, where the memory limitation makes parameter reduction a critical optimization target.

25 different layer configurations were tested, each of which was evaluated through qualitative text generation tasks and quantitative assessment on the HellaSwag dataset [29]. The results revealed several that conservative modification, often maintained reasonable performance while reducing computational overhead. In particular, skipping the layers more towards the middle of the model rather than its ends resulted in low performance degradation. For instance, the configuration that skipped layers 23-27 achieved a HellaSwag score of 0.38 compared to the baseline's 0.34, suggesting that certain middle layers may contribute less to final performance than expected.

However, more aggressive modifications typically led to 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, maintaining a balanced representation across different depths remains crucial for coherent language generation.

These preliminary findings informed the subsequent approach to systematic compression: it revealed that strategic layer removal could sometimes improve performance metrics, suggesting that pruning techniques might offer promising avenues for optimization; at the same time, it showed how 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 Hugging Face model documentation [30].

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, the computational requirements make it rather prohibitive. Distillation essentially requires training a model from scratch, demanding extensive GPU resources that far exceed the computational budget available for the project. Thus, an existing pre-distilled model was utilized instead which already provides a compact yet capable foundation.

The pipeline follows a five-stage progression, with each stage building upon the previous one. The 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 to remove less critical weights within the remaining layers. This is a finer-grained approach compared to the depth-pruning, as it operates at the parameter level.

3. The third stage introduces *Low-Rank Adaptation* (LoRA) (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, reducing the memory footprint of individual parameters.

5. The final stage includes an optional *Eigenspace Low-Rank Approximation* (EoRA) 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 explicitly stated otherwise. The modular design supports selective execution through comprehensive tuning options, such as the ability to skip specific stages or experiment with different parameter configurations without modifying the core implementation. Having intermediate models allows to conduct ablation studies more easily and adapt the pipeline to specific hardware constraints.

3.3 Pruning techniques

Figure 3.1 shows a visual comparison between the two pruning techniques. Following depth-wise pruning, the second stage of the compression pipeline applies width-wise pruning. As mentioned in Section 3.3.1, it operates at the parameter level, eliminating individual weights within the remaining layers to achieve fine-grained compression. Width pruning can be either:

- **Unstructured**:

- **Structured**:

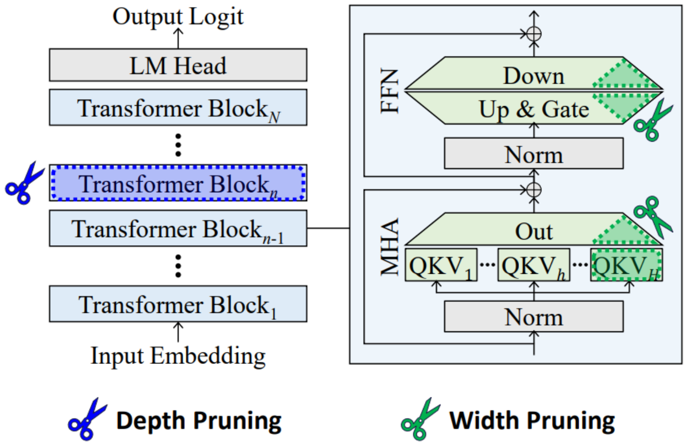


Figure 3.1: In depth pruning, we remove the single transformer layers (left). On the other hand, in width pruning we remove single neurons from the weight matrices (right). The image was sourced from [32].

3.3.1 Depth Pruning

Depth pruning represents a structured approach to neural network compression that targets the architectural dimension of model depth rather than individual parameter elimination. Unlike width pruning, which reduces the size of weight matrices by removing neurons or attention heads while preserving the total number of layers, depth pruning removes entire layers or blocks from the network architecture. In the context of transformer-based language models, this typically involves eliminating complete attention blocks, each containing both a multi-head attention module and feed-forward network components.

The depth pruning approach follows the methodology established by Kim et al. in their Shortened LLaMA work [32], which demonstrated that removing entire transformer blocks can achieve competitive performance while delivering significant inference speedups, particularly under memory-constrained conditions. Rather than reducing individual weight dimensions as in width pruning, depth pruning eliminates entire layers from the model architecture, creating a more direct path to computational savings.

The depth pruning process begins with calculating layer importance scores using one of several metrics implemented in the pipeline. Three primary importance calculation methods are supported: magnitude-based scoring, Taylor expansion-based gradient analysis, and perplexity-based evaluation. The magnitude approach computes the L1 norm of weights within each transformer block, providing a simple baseline for layer significance. The Taylor method leverages first-order gradient information to estimate the impact of layer removal on model performance, following the approximation $L(W=0) \approx L(W) + \nabla L \cdot (-W)$ where the gradient-weight product indicates layer importance. The perplexity method evaluates each layer by temporarily removing it and measuring the resulting degradation in language modeling performance on a calibration dataset.

Following the insights from Shortened LLaMA, the implementation protects the first four and last two layers from pruning, as these positions have been shown to be critical for maintaining performance (which is in line with what was discovered in Section 3.1). The remaining layers are ranked according to their importance scores, with the least significant layers selected for removal.

The removal process involves creating a new model architecture with the selected layers entirely eliminated, maintaining the original transformer structure for the remaining blocks. This approach contrasts sharply with width pruning techniques that create sparse weight matrices, as depth pruning produces models with clean, dense architectures that map naturally onto hardware accelerators without requiring specialized sparse computation support.

The implementation extends the Shortened LLaMA approach by integrating it into a comprehensive compression pipeline, where depth pruning serves as the foundation for subsequent optimization techniques. The layer importance calculations are performed efficiently using calibration datasets, typically requiring only 10 samples from standard corpora to produce reliable importance rankings. This efficiency makes depth pruning particularly attractive for resource-constrained scenarios where extensive calibration is impractical.

3.3.2 Width Pruning

WANDA introduces a novel pruning metric that combines weight magnitudes with input activation statistics to determine parameter importance. The core insight behind this approach stems from the observation that large language models exhibit emergent large magnitude features in their hidden states—outlier activations that are significantly larger than typical values and are crucial for model performance.

The WANDA importance metric for each weight W_{ij} is defined as:

$$S_{ij} = |W_{ij}| \cdot \|X_j\|_2 \tag{3.1}$$

where $|W_{ij}|$ represents the absolute value of the weight and $\|X_j\|_2$ evaluates the L2 norm of the j-th input feature across all calibration samples. This formulation addresses a key limitation of traditional magnitude-based pruning: it fails to account for input activations, which play an equally important role in determining neuron outputs, especially in the presence of large magnitude features.

3.3.3 Structured N:M Sparsity Implementation

While WANDA was originally designed for unstructured sparsity, the compression pipeline implements its structured variant to leverage hardware acceleration capabilities. Specifically, the implementation focuses on structured N:M sparsity patterns, where at most N out of every M consecutive weights are non-zero.

The structured pruning process operates as follows:

1. **Importance Calculation**: For each linear layer, compute the WANDA metric for all weights using calibration data

2. **Grouping**: Organize weights into groups of M consecutive elements within each output channel

3. **Selection**: Within each group, identify the N weights with the highest importance scores

4. **Pruning**: Set the remaining (M-N) weights to zero

This structured approach enables the use of specialized hardware accelerators, such as NVIDIA's sparse tensor cores, which can efficiently handle N:M sparsity patterns during inference.

3.3.4 Implementation Details

The WANDA implementation integrates seamlessly into the compression pipeline through the `prune_wanda` function. Key implementation aspects include:

Calibration Setup: The algorithm uses the same calibration dataset as the depth pruning stage, typically consisting of 10 samples from the C4 corpus, ensuring consistency across optimization stages.

Layer-wise Processing: WANDA processes each transformer layer sequentially, updating activations after each layer is pruned to ensure that subsequent layers receive realistic input distributions.

Per-output Comparison: Following the original WANDA methodology, weights are compared and pruned on a per-output basis rather than globally across the layer. This approach maintains balanced pruning ratios across different output features, which proves crucial for preserving model performance.

Binary Search for Sparsity: For the variant implementation, a binary search algorithm finds the optimal threshold parameter α that achieves the target sparsity ratio while maximizing the cumulative importance of retained weights.

3.3.5 Advantages and Computational Efficiency

WANDA offers several advantages over more complex pruning methods:

- **Single Forward Pass**: Unlike methods requiring iterative weight updates, WANDA completes pruning in a single forward pass through the model
- **No Weight Updates**: The algorithm requires no modifications to the remaining weights, suggesting that effective sparse subnetworks exist within the original model
- **Low Computational Overhead**: With $O(d^2)$ complexity compared to $O(d^3)$ for second-order methods, WANDA provides significant computational savings
- **Hardware Compatibility**: The structured N:M variant directly maps to accelerator capabilities without requiring specialized sparse computation libraries

The integration of WANDA as the width pruning stage creates a foundation for subsequent optimization techniques while maintaining the model's core functionality. The structured sparsity patterns ensure that the compressed model can benefit from hardware acceleration during inference, making this approach particularly suitable for deployment scenarios with strict performance requirements.

3.4 LoRA

3.5 Quantization and EoRA

3.6 Evaluation

Chapter 4

Experimental Results and Analysis

intro here

4.1 Preliminary observation of the results

Before examining the results based on In this section, we will present and comment on some examples of text generated by the

4.2 Results on TriviaQA

4.3 Results on WikiText

Chapter 5

Conclusion and Future Work

5.1 Future work

- speak about Distillation
- speak about experimenting with other quantization methods
- speak about adding support for other llms
- kV cache compression

5.2 Final remarks

The objective of this project was to research and implement a methodology for compressing LLaMA based LLMs,

Bibliography

[1] Mike Perkins, *Academic Integrity considerations of AI Large Language Models in the post-pandemic era: ChatGPT and beyond*, https://www.researchgate.net/publication/368775737_Academic_integrity_considerations_of_AI_Large_Language_Models_in_the_post-pandemic_era_ChatGPT_and_beyond.

[2] Daniel Mügge, *AI Is Threatening More Than Just Creative Jobs—It's Undermining Our Humanity*, 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*, https://arxiv.org/abs/2104.10350.

[4] Thomas Spencer, Siddharth Singh, *What the data centre and AI boom could mean for the energy sector*, 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* https://arxiv.org/pdf/2505.09598v1

[6] Google, *Google Environmental Report 2023*, 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*, https://arxiv.org/abs/2304.03271.

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

[9] OpenAI, *GPT-4 is OpenAI's most advanced system, producing safer and more useful responses*, https://openai.com/index/gpt-4/.

[10] Chris Nicholson, *A Beginner's Guide to LSTMs and Recurrent Neural Networks.* https://skymind.ai/wiki/lstm.

[11] S. Hochreiter, J. Schmidhuber, *Long Short-Term Memory*, 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*, https://arxiv.org/abs/1303.5778.

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

[14] Philip Gage, *A New Algorithm for Data Compression*, http://www.pennelynn.com/Documents/CUJ/HTML/94HTML/19940045.HTM

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

[16] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, https://arxiv.org/abs/1409.0473.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, `https://arxiv.org/abs/1810.04805`.

[18] 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*, `https://arxiv.org/abs/2302.13971`.

[19] 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*, `https://arxiv.org/abs/2307.09288`.

[20] Llama Team, AI @ Meta, *The Llama 3 Herd of Models*, `https://arxiv.org/abs/2407.21783`.

[21] 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*, `https://arxiv.org/abs/2312.14750`.

[22] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, *Distilling the Knowledge in a Neural Network*, `https://arxiv.org/pdf/1503.02531`.

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

[24] Fabrizio Sandri, Elia Cunegatti, Giovanni Iacca, *2SSP: A Two-Stage Framework for Structured Pruning of LLMs*, `https://arxiv.org/pdf/2501.17771`.

[25] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, Danqi Chen *Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning*, https://arxiv.org/abs/2310.06694.

[26] Angelo Galavotti, *FRANKEN-LLAMA code repository*, https://github.com/AngeloGalav/franken-llama

[27] 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*, https://arxiv.org/abs/1912.01703.

[28] Hugging Face, *Transformers library documentation*, https://huggingface.co/docs/transformers/index.

[29] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, Yejin Choi *HellaSwag: Can a Machine Really Finish Your Sentence?* https://arxiv.org/abs/1905.07830.

[30] Llama Team, AI @ Meta, *Llama-3.2-1B*, https://huggingface.co/meta-llama/Llama-3.2-1B

[31] Hanjuan Huang, Hao-Jia Song, Hsing-Kuo Pao, *Large Language Model Pruning*, https://arxiv.org/abs/2406.00030.

[32] 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*, https://arxiv.org/abs/2402.02834.

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

[34] Mingjie Sun, Zhuang Liu, Anna Bair, J. Zico Kolter, *A Simple and Effective Pruning Approach for Large Language Models*, https://arxiv.org/abs/2306.11695.

[35] 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*, https://arxiv.org/abs/2106.09685.

[36] ModelCloud, *GPTQModel*, https://github.com/ModelCloud/GPTQModel.

[37] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, Dan Alistarh *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*, https://arxiv.org/abs/2210.17323

[38] 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*, https://arxiv.org/abs/2410.21271.

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

[40] Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher, *Pointer Sentinel Mixture Models*, https://arxiv.org/abs/1609.07843.

[41] 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*, https://arxiv.org/pdf/1910.10683.

Acknowledgements

Pending