Technical University Berlin

Master's Thesis

Green Tech or Digital Polluter?

Understanding Emission Drivers of Different Generative AI Customization and Implementation Approaches.

Daniel Wetzel

Technical University Berlin
Berlin, Germany
daniel@d-wetzel.de

supervised by

Prof. Dr.

Timm Teubner

Technical University Berlin
Berlin, Germany
teubner@tu-berlin.de

Prof. Dr.-Ing.

Stefan Tai

Technical University Berlin
Berlin, Germany
tai@tu-berlin.de

November 18, 2024

DECLARATION OF AUTHORSHIP

I hereby declare that the thesis submitted is my own, unaided work, completed without any external help. Only the sources and resources listed were used. All passages taken from the sources and aids used, either unchanged or paraphrased, have been marked as such.

Where generative AI tools were used, I have indicated the product name, manufacturer, the software version used, as well as the respective purpose (e.g. checking and improving language in the texts, systematic research). I am fully responsible for the selection, adoption, and all results of the AI-generated output I use.

I have taken note of the Principles for Ensuring Good Research Practice at TU Berlin dated 8 March 2017. `https://www.tu.berlin/en/working-at-tu-berlin/important-documents/guidelinesdirectives/principles-for-ensuring-good-research-practice`

I further declare that I have not submitted the thesis in the same or similar form to any other examination authority.

Generative AI models & tools used for this academic work (apart from the benchmarked and tested models):

Grammarly	AI assistant used for spell-checking, grammar-checking & writing style enhancements.
GPT o1-Preview, GPT-4o & GPT-4-Turbo	Utilized via ChatGPT web app and a custom-built UI for coding assistance, refinement of system prompts & prompt templates.
GitHub Copilot	Used as coding assistant via the VSCode extension.

ACKNOLEDGEMENTS

This thesis is accompanied by an interactive Streamlit Dashboard [1] and a public code repository available on GitHub [2]. The data collected from the experiments is stored in the repository and visualized in the Streamlit Dashboard. The code for the dashboard is also accessible in the GitHub repository.

Interactive Streamlit Dashboard [1]
`https://thesis.d-wetzel.de`

GitHub Code Repository [2]
`https://github.com/danielwetzel/llm-customization`

This research was supported by Deloitte Consulting GmbH. Special acknowledgment is extended to Felix Hoffmann, who mentored and supervised the research on the professional side.

Felix Hoffmann

Deloitte Consulting GmbH
Berlin, Germany
felhoffmann@deloitte.de

While Deloitte funded the cloud compute costs associated with this thesis, as well as the costs related to the Azure OpenAI API, they had no direct influence on the content of this thesis. Deloitte did not incentivize any direction of the content nor had any influence on the research outcomes.

This thesis represents independent academic work, completed without influence from any third party.

ABSTRACT - ENGLISH

Generative Artificial Intelligence (Gen AI), particularly Large Language Models (LLMs), have rapidly advanced and are widely adopted in various domains. However, these models consume significant energy during training and inference, leading to a substantial environmental impact. This thesis investigates methods to reduce the energy consumption of LLM inference without compromising model quality. Specifically, it examines the effectiveness of quantization, prompt engineering, Retrieval-Augmented Generation (RAG), and serving engine optimization. Experiments were conducted using models such as LLaMA 3.1 and Mistral Nemo across different quantization levels and prompt configurations. Model quality was evaluated using the Arena Hard Auto Benchmark, automated with an LLM-as-a-Judge approach. The results demonstrate that quantization significantly enhances energy efficiency. Reducing model precision from 16-bit to 8-bit yielded substantial energy savings of up to 60% with minimal impact on model capabilities. Further quantization to 4-bit led to additional energy reductions, albeit with a more noticeable decrease in quality. Prompt engineering and RAG improved model quality, particularly for models with lower baseline capabilities. Yet, it increased energy consumption due to longer input sequences. Serving engine optimization, specifically using the vLLM engine, substantially improved processing speed and energy efficiency compared to traditional implementations. Based on these findings, a decision framework is developed to guide practitioners in optimizing LLM deployments for both efficiency and sustainability. The framework provides practical guidelines to achieve energy-efficient LLM applications without sacrificing quality. This work contributes significantly to advancing sustainable AI practices by offering actionable insights into optimizing LLM inference.

ABSTRACT - GERMAN

KI Sprachmodelle (LLMs) haben in den letzten Jahren eine rasante Entwicklung erfahren und finden bereits maßgeblichen Anklang im Alltag vieler Menschen. Allerdings verbrauchen sie während des Trainings und der Inferenz ehrhebliche Mengen an Energie, was zu einer beträchtlichen Umweltbelastung führt. Diese Masterarbeit untersucht Methoden, um den Energieverbrauch bei der LLM-Inferenz zu reduzieren, ohne dabei die Modellqualität zu beeinträchtigen. Im Fokus stehen die Wirksamkeit von Quantisierung, Prompt Engineering, Retrieval-Augmented Generation (RAG) und der Nutzung von optimierten Serving Engines. Experimente wurden unter Anderem mit den Modellen LLaMA 3.1 und Mistral Nemo durchgeführt, wobei verschiedene Quantisierungsstufen und Prompt-Konfigurationen getestet wurden. Die Modellqualität wurde mithilfe des Arena Hard Auto Benchmarks bewertet, welcher automatisiert von einem LLM-Richter ausgeführt wurde. Die Ergebnisse zeigen, dass Quantisierung die Energieeffizienz signifikant steigern kann. Die Reduktion der Modellpräzision von 16 Bit auf 8 Bit führte zu Energieeinsparungen von bis zu 60% bei minimalen Leistungseinbußen. Eine weitere Quantisierung (auf 4 Bit) ermöglichte zusätzliche Energieeinsparungen, allerdings mit einer merklichen Abnahme der Qualität. Prompt Engineering und RAG konnten, insbesondere bei Modellen mit geringerer Ausgangsleistung, die Modellqualität verbessern. Sie erhöhten jedoch, aufgrund der längeren Eingabesequenzen, auch den Energieverbrauch. Das Verwenden einer optimierten Serving Engine, explizit das Verwenden der vLLM-Engine, führte im Vergleich zu traditionellen Implementierungen zu einer erheblichen Steigerung der Verarbeitungsgeschwindigkeit und Energieeffizienz. Auf Grundlage dieser Erkenntnisse wurde ein Entscheidungs-Framework entwickelt, welches Entscheidungsträgern bei der Optimierung von LLM-Implementierungen hinsichtlich Effizienz und Nachhaltigkeit unterstützen soll. Das Framework bietet praktische Leitlinien, um energieeffiziente LLM-Anwendungen zu realisieren, ohne die Qualität zu beeinträchtigen. Die Arbeit leistet damit einen wichtigen Beitrag zur Förderung nachhaltiger KI-Praktiken, indem sie Einblicke in die Optimierung der LLM-Inferenz bietet.

Contents

1 INTRODUCTION

Generative Artificial Intelligence (Gen AI) is rapidly transforming various sectors. At its core are Large Language Models (LLMs), which enable machines to generate human-like text with high accuracy. However, this technological progress incurs significant energy consumption. Training and deploying LLMs require vast computational resources, leading to substantial carbon emissions. As more organizations adopt Gen AI, the environmental impact becomes a critical concern. This thesis tackles the challenge of reducing energy consumption during LLM inference without compromising model quality. By investigating techniques such as quantization and retrieval-augmented generation (RAG), the research aims to enhance efficiency in LLM deployments. Through theoretical analysis and empirical experiments, it offers practical solutions for sustainable AI practices.

1.1 Background and Motivation

Gen AI has emerged as a transformative technology with the potential to drive significant efficiency gains across various sectors [3],[4],[5],[6]. Recent analyses predict that Gen AI could add trillions of dollars in value to the global economy [6] and transform business operations drastically [3]. LLMs are at the forefront of this revolution, enabling machines to process and generate human-like text with unprecedented accuracy [7],[8].

In the past year, numerous new LLMs have been released, reflecting rapid advancements in the field. Proprietary models such as OpenAI's GPT series [8], [9],[10],[11],[12], Anthropic's Claude series [13],[14], and Google's Gemini series [15],[16],[17] have demonstrated remarkable capabilities in tasks ranging from translation to complex reasoning. Open-source models like BLOOM [18], the LLaMA family [19],[20],[21],[22], and the Mistral family [23],[24],[23] have also made significant contributions and reaching capabilities almost on part with leading closed source LLMs.

The rapid adoption of Gen AI technologies has led to an exponential increase in the deployment of LLMs for real-world applications [4]. As organizations integrate these models into their services, the demand for computational resources to support both training and inference phases escalates significantly. Training LLMs requires vast amounts of data and computational power, often involving billions of parameters and trillions of training tokens [8],[18],[20],[21]. This intensive computational demand results in substantial energy consumption and associated carbon emissions. Considerable efforts have been made to estimate the energy consumption and emissions during the training phase of LLMs [8],[18],[20],[21], leading to increased awareness and optimization strategies.

While optimizing the training efficiency of LLMs has received significant attention [25],[26],[21], less emphasis has been placed on energy consumption during the inference phase. Inference, which involves deploying the trained model to generate outputs in response to user inputs, can account for a substantial portion of an LLM's total energy footprint over its lifecycle [18]. As LLMs become increasingly integrated into consumer applications with high usage volumes, the cumulative energy consumption during inference poses a considerable environmental concern.

Notably, leading technology companies have reported significant increases in carbon emissions attributed to the growing computational demands of AI. Google's greenhouse gas emissions in 2023 were 48% higher than in 2019, according to their latest environmental report [27],[28]. The company attributes this rise to the increasing energy requirements of their data centers, exacerbated by the explosive growth of AI technologies [27]. Google acknowledges that integrating AI into their products intensifies energy demands, stating that *"as we further integrate AI into our products, reducing emissions may be challenging due to increasing energy demands from the greater intensity of AI compute"* [28]. This trend highlights the broader industry challenge of balancing the benefits of AI advancements with the imperative to reduce environmental impact.

While cloud providers such as Google, Microsoft, and Amazon utilize high proportions of carbon-free energy and engage in carbon offsetting [28],[29],[30], the increasing energy demands still pose a significant threat to the climate. In response to rising energy needs, Amazon has recently acquired a nuclear power plant together with a data center [31]. Although nuclear power is considered a low-carbon energy source, concerns remain about its sustainability due to challenges associated with nuclear waste management. This initiative highlights the broader industry effort to balance escalating energy requirements with environmental responsibility.

Despite growing awareness of AI's carbon footprint [18],[20],[21], there is a gap in research concerning effective methods to reduce energy consumption during LLM inference without compromising model quality.

This thesis addresses this gap by investigating how customizations to LLMs, such as quantization [32],[33],[34],[35] and RAG [36], can reduce energy consumption during inference or increase the output quality of smaller and more energy-efficient models. By optimizing model configurations and employing advanced techniques to improve efficiency, it is possible to lessen the environmental impact of Gen AI deployments while maintaining or enhancing model performance.

1.2 Research Problem and Objectives

The widespread adoption of Gen AI and LLMs has led to significant increases in energy consumption during both training and inference phases [8],[18],[20],[21]. While considerable efforts have been directed toward optimizing the energy efficiency of LLM training [25],[26],[21], energy consumption during inference remains underexplored. Inference, which entails generating outputs in response to user inputs, constitutes a substantial portion of an LLM's total energy footprint due to continuous and large-scale deployment in various applications [18].

The central research question addressed in this thesis is:

How can customizations to Large Language Models reduce energy consumption during inference without compromising model quality?

To answer this question, the following objectives are established:

1. **Identify and analyze the key factors contributing to high energy consumption during LLM inference.** This involves examining the computational processes of inference operations and pinpointing the most energy-intensive components.

2. **Evaluate the impact of model customizations on energy consumption and output quality.** Experiments will be conducted to determine the effectiveness of techniques such as quantization and retrieval-augmented generation (RAG) in reducing energy usage without degrading output quality.

3. **Develop a framework for optimizing LLM deployment to achieve energy-efficient inference.** The framework will provide guidelines for implementing customization strategies that balance energy consumption with model performance, facilitating sustainable deployment of Gen AI technologies.

4. **Assess the environmental implications of deploying optimized LLMs in real-world applications.** This includes quantifying potential reductions in carbon emissions and evaluating the scalability of the proposed solutions across different platforms and use cases.

To achieve these objectives, a combination of theoretical analysis and empirical experimentation is employed. The computational processes involved in LLM inference are analyzed to identify energy-

intensive operations. Experiments are conducted using various model customizations, measuring their effects on energy consumption and output quality. The findings inform the development of a decision framework designed to guide practitioners in optimizing LLM deployments.

The research demonstrates that specific customizations to LLMs can significantly reduce energy consumption during inference while maintaining or enhancing model quality. Techniques such as quantization and advanced prompt engineering improve efficiency without compromising performance. The developed framework offers practical guidelines for achieving sustainable AI deployments.

By addressing these objectives and presenting actionable solutions, the thesis contributes to the development of sustainable AI practices. The findings guide practitioners and researchers in adopting energy-efficient techniques, promoting responsible use of advanced AI models while maintaining optimal performance.

1.3 Thesis Overview

This thesis presents a comprehensive exploration of strategies to reduce energy consumption during LLM inference without compromising model quality. The research is structured across seven chapters, each building upon the previous to develop a coherent understanding of the subject.

Chapter 2 introduces the background and motivation for the study, defines the research problem, outlines the objectives, and provides an overview of the thesis structure. Chapter 2 reviews fundamental concepts and existing literature relevant to LLMs, energy consumption during inference, quantization techniques, and retrieval-augmented generation (RAG). It identifies gaps in current research and sets the context for the subsequent analysis.

Chapter 3 details the methodological approach adopted in the research. It describes the experimental setup, including the infrastructure design and tools used for measuring energy consumption during inference. The chapter also explains the benchmarks and metrics employed for evaluating model quality.

Chapter 4 presents preliminary studies that investigate initial assessments of model inference efficiency. It explores bottlenecks affecting energy consumption during LLM inference and discusses optimization techniques for improving performance.

Chapter 5 reports the findings from the experiments conducted. It analyzes the impact of model customizations, such as quantization and RAG, on energy consumption and output quality. The chapter provides empirical evidence supporting the effectiveness of these techniques.

Chapter 6 develops a decision framework for optimizing LLM deployment to achieve energy-efficient inference. It offers practical guidelines and considerations for practitioners aiming to reduce energy consumption without compromising model performance.

Finally, Chapter 7 concludes the thesis by summarizing the key findings, discussing implications for sustainable AI practices, and suggesting directions for future work. This structured approach ensures a logical progression from identifying the research problem to developing practical solutions, facilitating a comprehensive understanding of how customizations to LLMs can enhance energy efficiency during inference.

2 BACKGROUND AND RELATED WORKS

This chapter provides the theoretical foundation necessary to contextualize the research conducted in this thesis. t begins with an introduction to the fundamental processes of tokenizing and vectorizing text input, essential for converting human language into a format that computational models can process. Following this, a comprehensive overview of the Transformer architecture is presented, highlighting the core framework that underpins modern Large Language Models (LLMs). Key concepts related to the calculation of emissions and energy consumption, both during code execution and within cloud environments, are then explored. The chapter proceeds to discuss methods for evaluating the output quality of LLMs and approaches to customizing and implementing them, emphasizing strategies for enhancing efficiency and sustainability.

In the latter part of this chapter, existing frameworks and methodologies for estimating the carbon footprint of LLMs are reviewed, including an analysis of current foundation models and their respective emissions. By identifying gaps in the current literature, the chapter highlights areas where further research is needed to improve the understanding of the environmental impact of LLM deployments.

2.1 Tokenizing and Vectorizing Text Input

Natural Language Processing (NLP) involves converting human language into a format that computational models can process. This conversion comprises two fundamental steps: *tokenization* and *vectorization*.

Tokenization is the process of breaking down text into smaller units known as *tokens* [37]. These tokens can be words, subwords, or characters. Word-level tokenization splits text based on spaces and punctuation, isolating individual words. However, challenges such as handling rare or out-of-vocabulary words have led to the development of subword tokenization methods like Byte Pair Encoding (BPE) [38]. BPE combines the benefits of word-level and character-level tokenization by representing words as sequences of subword units, effectively reducing the vocabulary size while retaining the ability to represent rare words.

Following tokenization, each token is mapped to a numerical representation through *vectorization*. This mapping is essential because machine learning models require numerical input to perform computations. One prevalent method for vectorization is the use of *word embeddings* [39], which are dense, low-dimensional vectors that capture semantic and syntactic properties of words.

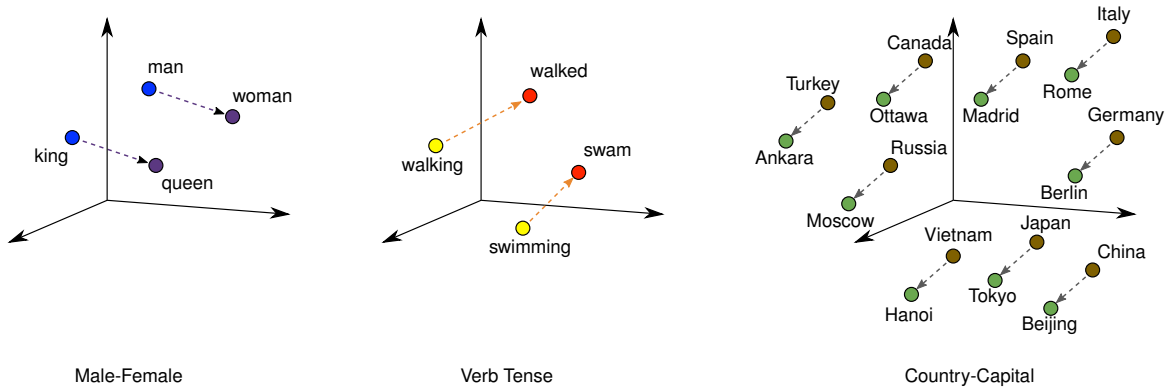

Figure 1: Vectorized words in a three-dimensional space (Source: Google Developer Guides [40])

Word embeddings are learned from large corpora of text using unsupervised learning techniques. Models such as Word2Vec [41] and GloVe [42] generate embeddings by leveraging the distributional hypothesis,

which states that words appearing in similar contexts tend to have similar meanings [43]. In the resultant vector space, words with related meanings are positioned close to each other.

Figure 1 illustrates a simplified representation of words in a three-dimensional vector space, where semantic relationships are encoded as distances and directions. Words with similar meanings are located near each other, capturing linguistic nuances in their spatial arrangement.

An important property of word embeddings is their ability to capture linguistic regularities through vector arithmetic [44]. Certain relational patterns can be expressed as vector operations. For instance, the analogy "king is to queen as man is to woman" is represented mathematically as:

$$\overrightarrow{\mathbf{V}_{\text{king}}} \quad + \quad (\overrightarrow{\mathbf{V}_{\text{woman}}} - \overrightarrow{\mathbf{V}_{\text{man}}}) \quad \approx \quad \overrightarrow{\mathbf{V}_{\text{queen}}}$$

Similarly, verb tense transformations can be captured:

$$\overrightarrow{\mathbf{V}_{\text{swimming}}} \quad + \quad (\overrightarrow{\mathbf{V}_{\text{walked}}} - \overrightarrow{\mathbf{V}_{\text{walking}}}) \quad \approx \quad \overrightarrow{\mathbf{V}_{\text{swam}}}$$

These equations demonstrate that consistent vector differences correspond to specific linguistic relationships, such as gender distinctions or tense changes.

To represent larger units of text, such as sentences, word vectors can be combined. Simple methods include averaging the embeddings of individual tokens [45], while advanced models utilize architectures that consider word order and context. Incorporating *positional encodings* [7] allows models to capture the sequence of tokens, which is crucial for understanding the meaning of sentences.

The processes of tokenizing and vectorizing text input form the foundation for modern NLP models. By converting text into numerical representations that encode linguistic information, mathematical operations and machine learning techniques can analyze language data. This is critical for tasks such as text classification, machine translation, and language modeling, where understanding semantic and syntactic properties is essential.

2.2 Overview of the Transformers Architecture

In order to understand the customization and implementation strategies discussed in this thesis, it is essential to have a basic understanding of the Transformer architecture. The architecture was introduced by Vaswani et al. in 2017 [7] and is still the backbone of current Large Language Models (LLMs).

As shown in Figure 2, the Transformer consists of two main parts: the encoder (on the left) and the decoder (on the right). Both are composed of multiple identical layers, with each layer containing multi-head attention mechanisms and feed-forward networks.

The Transformer processes sequences of data by first converting the input into continuous vector representations through an *embedding* layer. As depicted in the bottom left of the diagram, each token in the input sequence is mapped to a high-dimensional vector that captures its semantic meaning. Since the Transformer processes the tokens in parallel, it requires a mechanism to encode the order of tokens in a sequence. To handle this, *positional encodings* are added to the embeddings (seen at the bottom of both encoder and decoder). These encodings are created using sine and cosine functions, providing the model with information about the position of each token.

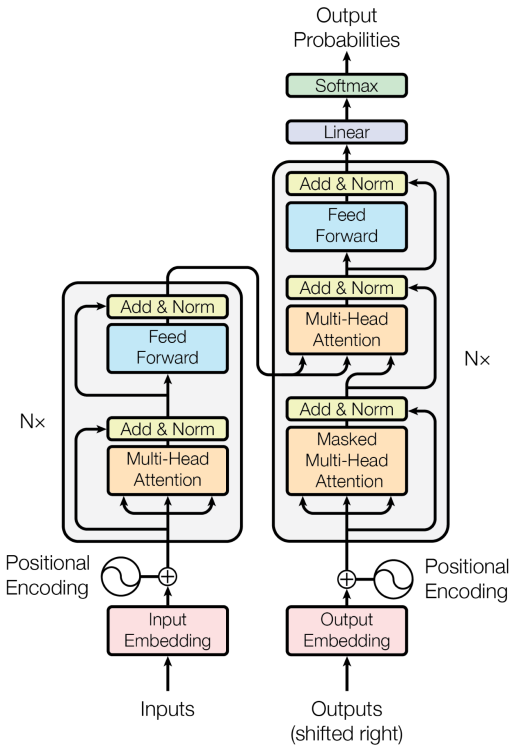

Figure 2: The Transformer - model architecture (Source: Vaswani et al. [7])

One of the core innovations, marked in yellow in Figure 2, is the *self-attention mechanism*, which allows each token in the input sequence to focus on other tokens, regardless of their position. For every token, the model computes three vectors: *query*, *key*, and *value*, which are used to compute the attention scores. These scores are calculated by taking the dot product between the query and key vectors, then normalizing them using the softmax function (as indicated by the arrows connecting the attention components in the diagram). This allows each token to weigh its importance relative to other tokens in the sequence.

The *multi-head attention mechanism*, illustrated as multiple parallel blocks in the diagram, extends the self-attention mechanism by allowing the model to project the queries, keys, and values into multiple subspaces, referred to as attention heads. Each head captures different aspects of the input sequence, and their outputs are concatenated and linearly transformed to form the final output for each token. This multi-head approach, shown clearly in both the encoder and decoder, enables the model to attend to multiple parts of the sequence simultaneously.

In addition to multi-head attention, each layer also contains a *feed-forward network* (FFN), shown in blue in Figure 2. The FFN applies two linear transformations with a ReLU activation in between, adding non-linearity to the model. This step is applied independently to each token's representation, allowing the model to learn more complex relationships in the data.

The overall architecture follows an *encoder-decoder structure*, as shown in the image. The encoder (left side) transforms the input sequence into a set of contextual representations. These are passed to the decoder (right side), which generates the output sequence. The decoder contains an additional layer of multi-head attention, seen in the middle, which attends to the encoder's output while generating the target sequence. Furthermore, as indicated by the "Masked Multi-Head Attention" block in the decoder, a masking mechanism is applied to prevent tokens from attending to future tokens, ensuring that output generation remains sequential during training.

To improve training stability and efficiency, *residual connections* (indicated by the arrows looping back in the diagram) are used to allow information to bypass certain layers. This helps maintain the flow of information through the network and avoids issues such as vanishing gradients. Additionally, *layer normalization* is applied at multiple points in the architecture to ensure that the outputs of each layer are scaled properly.

Finally, the output of the decoder is passed through a linear layer, followed by a softmax function (depicted at the top of Figure 2), which converts the final output into a probability distribution over the target vocabulary. This allows the model to generate the next token in a sequence by selecting the word with the highest probability, iteratively constructing the entire output sequence.

2.3 Calculating Code and Cloud Emissions

Understanding and accurately calculating the energy consumption of code execution is essential for assessing the environmental impact of machine learning models, particularly large-scale models like LLMs. This subsection discusses methods for measuring the energy consumption of code and how to translate this consumption into carbon emissions within cloud computing environments. Both direct and indirect approaches to energy measurement will be examined. Factors such as data center efficiency and regional energy mixes influencing emission calculations will also be explored.

2.3.1 Calculating the Energy Consumption of Code

Measuring the energy consumption of code is essential for quantifying the environmental impact of machine learning models. Accurate estimation enables researchers and practitioners to understand and mitigate the carbon footprint associated with training and deploying large-scale models, such as Large Language Models (LLMs).

Methods for calculating energy consumption of code are categorized into direct measurements using hardware sensors and indirect estimations based on software profiling. Direct measurement involves using instruments like wattmeters or accessing hardware sensors to monitor energy usage of components such as CPUs and GPUs during code execution. However, this approach can be impractical due to the need for kernel-level resources or direct hardware access.

Alternatively, software-based tools provide indirect estimations by monitoring resource utilization and applying power models. Tools like CodeCarbon [46] and Carbontracker [47] run alongside the code. They estimate energy consumption based on metrics such as CPU and GPU utilization, memory usage, and hardware specifications. These tools periodically sample usage statistics and calculate energy consumption over time by multiplying the estimated power usage by the duration of each interval. The total energy consumption is obtained by summing the energy usage over all intervals.

CodeCarbon, for instance, is an open-source library designed to estimate the carbon emissions of machine learning code executions [46]. It queries the power consumption of hardware components, either through direct readings (if available) or using predefined power profiles for different hardware. The estimated power consumption is multiplied by the time interval to obtain the energy consumed during that period. By summing these values over the entire execution, CodeCarbon computes the total energy consumption.

In their study, Heguerte et al. [48] evaluated several tools for estimating energy consumption when training deep learning models. They found that CodeCarbon provided relatively accurate estimations among software-based methods, making it a preferable option. The study highlighted that the choice of tool and measurement methodology can significantly impact the estimated energy consumption.

Estimating energy consumption on cloud platforms, such as AWS EC2 instances, is more challenging due to limited access to hardware details and sensors. Davy [49] proposed methods for estimating the energy

consumption of EC2 instances by analyzing hardware specifications. By creating power consumption models based on component loads, these methods help estimate the energy usage of code running in virtualized environments where direct measurement is not feasible.

In summary, calculating the energy consumption of code involves either direct measurement or indirect estimation. While direct measurement offers higher accuracy, software-based tools like CodeCarbon provide practical solutions for most users. Understanding how these tools work and their limitations is crucial for obtaining reliable estimates of energy consumption in machine learning workflows.

2.3.2 Translating Energy Consumption to Cloud Emissions

Translating the energy consumption of code execution into associated carbon emissions is crucial for evaluating the environmental impact of machine learning workloads in cloud environments. A key factor in this translation is the Power Usage Effectiveness (PUE), a metric that measures the energy efficiency of a data center.

PUE is defined as the ratio of the total facility energy consumed by a data center to the energy consumed by its IT equipment:

$$\text{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}}.$$

A lower PUE indicates higher efficiency, as more energy is utilized by computing equipment rather than ancillary services like cooling and lighting.

According to a study by IDC [50], the average PUE for data centers in 2023 is 1.22. This figure, cited by Amazon Web Services (AWS) [51], aligns with reports from other major cloud providers. Microsoft reports an average PUE of 1.22 for their data centers, with their newest generation achieving ratings as low as 1.12 [52]. Their sustainability report for 2023 [29] expresses the belief that a PUE of 1.0 is attainable. Google publishes an average PUE of 1.09 over the last 12 months and 1.08 for the first quarter of 2024, with the lowest PUE reaching 1.06 [53]. These values demonstrate the ongoing efforts by cloud providers to enhance energy efficiency.

Tools such as CodeCarbon [46] enable users to translate energy consumption directly into carbon emissions by utilizing live energy mix data for the approximate location and applying a defined PUE. By inputting the energy consumed by the code and the PUE of the data center, CodeCarbon accounts for the total energy usage, including overheads, and estimates the resultant emissions based on the regional electricity carbon intensity.

Major cloud providers have initiated extensive programs to reduce the carbon footprint of their data centers [28]–[30]. They have invested in renewable energy projects and implemented measures to increase energy efficiency. Google, for example, provides detailed information on the percentage of renewable energy used in each region, distinguishing between the renewable energy supplied by the local grid and additional renewable sources they procure to achieve their targets [54]. In contrast, Microsoft and Amazon do not share such detailed regional data.

All three providers support their sustainability claims with net-emissions calculations, which include offsets from projects with a positive impact on carbon dioxide reduction. Assessing the validity and implications of net emissions is beyond the scope of this thesis. These claims and the fact that net emissions may be significantly lower than gross emissions are acknowledged. However, due to the complexity of this topic, average emissions for specific regions will be used for the calculations of this research.

To obtain accurate emission data for different regions, information from electricityMaps [55] is utilized. The tool provides real-time and historical energy data, including the carbon intensity of electricity

generation in various regions. They offer an intuitive web application with a map interface and accessible APIs, facilitating the integration of their data into research projects. Notably, electricityMaps is also the data provider chosen by Google for assessing emissions in their regions [54].

In conclusion, translating energy consumption to cloud emissions requires accounting for the data center's PUE and the regional energy mix. Tools like CodeCarbon, combined with reliable data sources like electricityMaps, enable accurate estimation of emissions from code execution in cloud environments. This approach provides a practical means to assess and potentially reduce the environmental impact of machine learning workloads.

2.4 Measuring Large Language Model Quality

Evaluating the quality of large language models (LLMs) is a crucial task that necessitates benchmarks that are both precise and reliable. Historically, two primary methods have been used to assess these models: human evaluation and automated multiple-choice questions. Each method offers unique benefits but also presents notable limitations that impact their effectiveness in capturing the true capabilities of LLMs. A novel approach was developed to overcome these limitations and combine the advantages of both methods: The LLM-as-a-Judge framework.

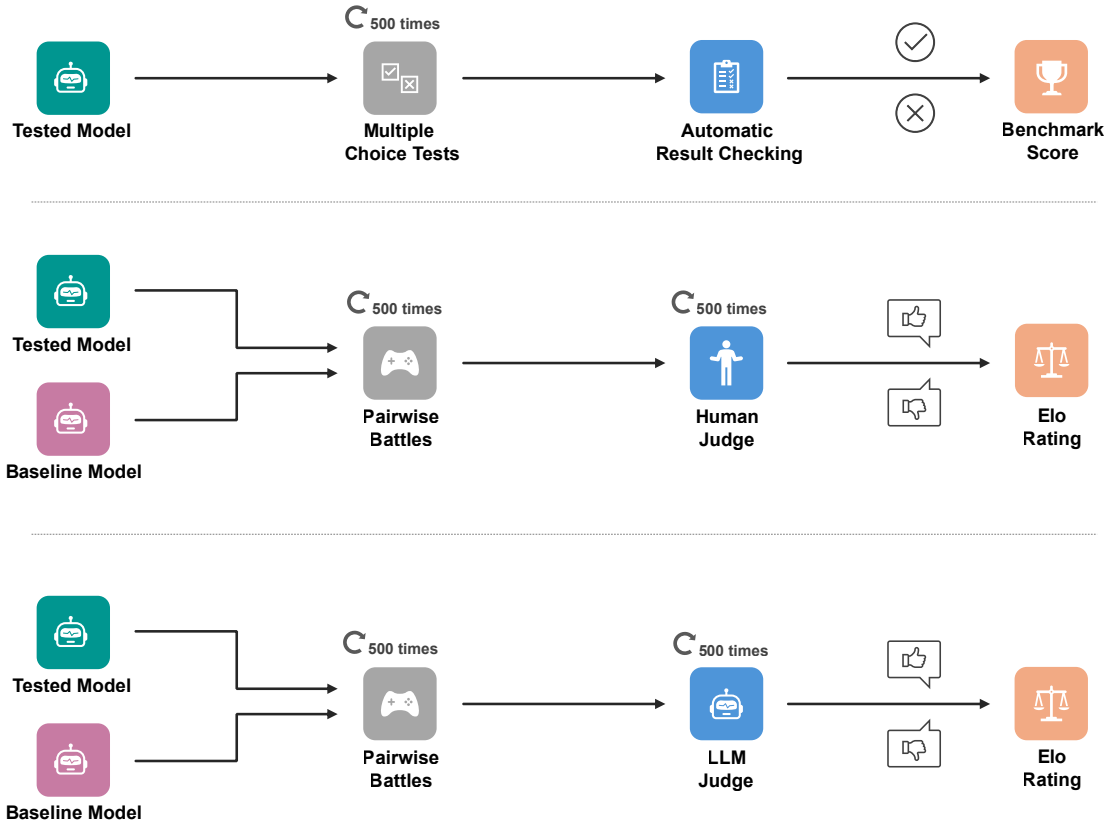

Figure 3: Illustration of the different Benchmark Approaches

In Figure 3, all three approaches are visualized in their evaluation flow. The following chapter will describe the approaches and their alterations, like the Open LLM Leaderboard. Afterwards, a specific implementation of the LLM-as-a-Judge framework, the Arena-Hard-Auto benchmark, will be reviewed. This specific benchmark will be used in within this research to evaluate the output quality of different LLM configurations.

Human Evaluation Tests

Human evaluation tests are often considered the gold standard for assessing LLMs due to their ability to rate models based on open-ended questions and very specific tasks. In these evaluations, human experts manually review and judge the outputs generated by models, considering aspects such as relevance, coherence, fluency, and creativity. This approach provides a comprehensive assessment of text quality, particularly in tasks requiring deep understanding or nuanced judgment, such as creative writing or complex reasoning scenarios.

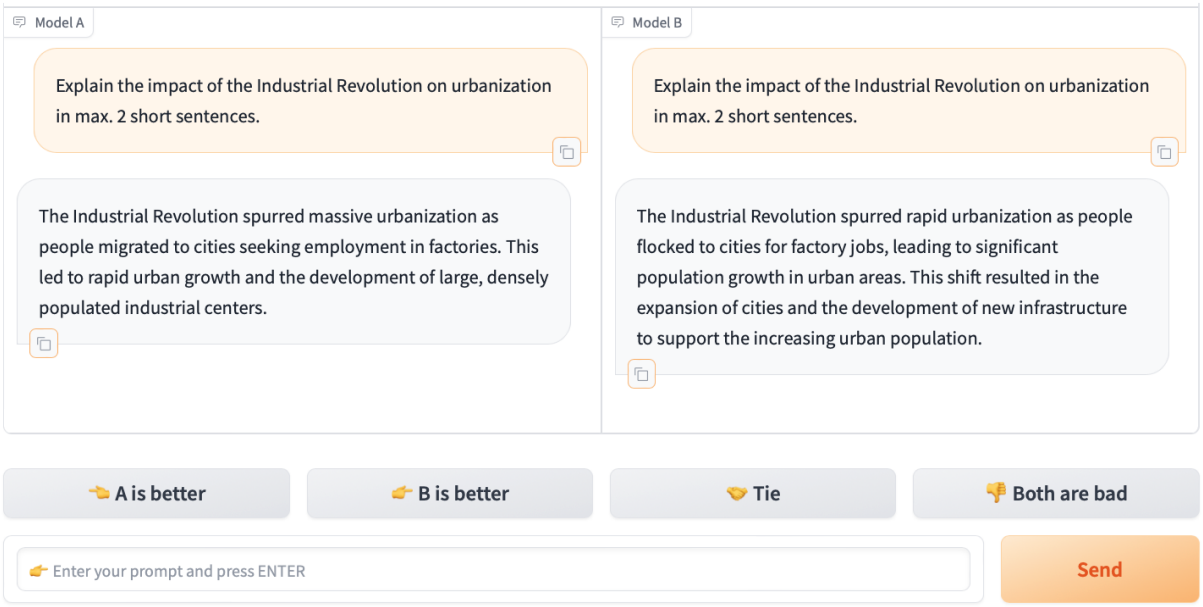

Figure 4: Example of a human evaluation benchmark, showing side-by-side comparisons of model outputs. (Source: Screenshot of the LMSys Chatbot Arena [56])

A common method of human evaluation involves side-by-side comparisons of model outputs. Evaluators are presented with responses from different models to the same prompt and asked to determine which response is superior based on predefined criteria. This process is illustrated in Figure 4, where evaluators compare different model outputs directly to assess effectiveness and accuracy.

However, human evaluation presents significant challenges. It is resource-intensive, requiring considerable time and financial investment. Coordinating multiple evaluators across different tasks and models can become logistically complex, especially for frequent or large-scale evaluations. Consistency and objectivity are additional challenges, as evaluators must be thoroughly trained to apply standardized criteria uniformly. As a result, human evaluation is often reserved for final assessments or specific scenarios where automated systems fall short, rather than being used for routine benchmarking [57].

To address these challenges, the LMSys team at UC Berkeley, in collaboration with Stanford and UCSD, developed the Chatbot Arena platform. This crowdsourced approach gathers diverse perspectives from a large pool of non-expert evaluators, providing a scalable solution for evaluating LLMs in practical scenarios. Users interact with two anonymized LLMs and select the better response, enabling the collection of varied user inputs that reflect real-world use cases. Over time, the platform has collected over 1.700.000 user votes for over 130 different models, making it one of the most comprehensive sources for LLM evaluation [56].

The platform employs the Bradley-Terry model [58] to rank models based on pairwise comparisons. This model estimates the likelihood of one model being preferred over another, providing a statistical

foundation for the rankings. Additionally, E-values [59] are used to calibrate and combine results from numerous comparisons, ensuring the robustness and reliability of the final rankings. These innovations aim to retain the depth of human insight while enhancing the scalability and practicality of evaluations [57].

While the crowdsourced approach offers scalability and real-world relevance, it also presents challenges. The quality of non-expert evaluations can vary, and large-scale participation is necessary to ensure the statistical validity of the results. Despite these challenges, this approach represents a significant advancement in LLM evaluation, especially when combined with automated tools to streamline the process.

Automated Multiple-Choice Benchmarks

Automated multiple-choice benchmarks have gained popularity due to their efficiency and objectivity. Benchmarks such as MMLU-Pro [60] (an improved version of MMLU [61]), MATH [62], SuperGLUE [63] (an enhanced version of GLUE [64]), and HellaSwag [65] are widely used to evaluate specific capabilities of LLMs, such as factual recall, logical reasoning, and linguistic understanding. These benchmarks typically present a model with a question and four possible answers, from which the model must select the correct one.

For example, a standard multiple-choice question might be:

Question	**Results**
What is the capital of France? A) Berlin B) Madrid **C) Paris** D) Rome	**Correct Answer: *C*** **Model Output: *C***

Table 1: Illustration of a multiple-choice benchmark question

Here, the correct answer is "C) Paris." The primary advantage of this approach is its scalability and efficiency. Multiple-choice tests can be administered across many models and repeated easily, making them suitable for large-scale benchmarking. Additionally, the standardized format allows for straightforward comparisons between models, helping to establish leaderboards based on performance metrics.

However, multiple-choice benchmarks have significant limitations. They do not align with the primary task LLMs are designed for: Generating Text. By limiting the model's responses to a predefined set of answers, these benchmarks fail to assess the model's ability to produce nuanced, creative, or contextually relevant outputs, which are crucial for many real-world applications.

Moreover, these benchmarks' public availability and standardized nature have led to their misuse. It became common practice for LLM providers to fine-tune their models on popular benchmarks at the end of the training process to achieve higher scores. This practice can lead to models being optimized to favor specific answer patterns rather than genuinely processing the content of the questions. For instance, if a model repeatedly encounters a question about the capital of France with the correct answer always being option "C," it may be adjusted to select this pattern rather than accurately processing the question.

This issue becomes particularly evident when the order of the answer options is altered:

Question	Results
What is the capital of France? **A) Paris** B) Berlin C) Rome D) Madrid	**Correct Answer: *A*** **Model Output: *C***

Table 2: Illustration of an incorrect answer to a multiple-choice benchmark after answer-swapping

If the model's accuracy drops when the correct answer "Paris" is moved from option "C" to "A" (see Table 2), it suggests that the model has been fine-tuned to rely on specific answer positioning rather than accurately processing the content. This specific behaviour could be observed in 2 separate studies by Gupta et. al. [66] as well as Pezeshkpour and Hruschka [67]. These studies suggest a quality drop of up to 75% depending on the benchmark and the model used after shuffling the answer positioning.

These limitations highlight the need for alternative benchmarking approaches that more accurately assess the generative and contextual capabilities of LLMs, ensuring evaluations better reflect their real-world applications.

Open LLM Leaderboard: Addressing Benchmark Saturation

The Open LLM Leaderboard [68] by Hugging Face aims to address some limitations associated with traditional benchmarks. It bundles multiple benchmarks within a standardized and reproducible framework to evaluate LLMs under consistent conditions. By testing models on identical questions in the same sequence and applying uniform evaluation criteria, the leaderboard ensures a fair comparison of LLMs. This approach prevents providers of foundation models from selectively choosing only the benchmarks that show the most favorable results for their models [69], [70]. Initially, the Open LLM Leaderboard gained traction as a reliable source for comparing state-of-the-art models, with over 2 million unique visitors and 300,000 community interactions in just ten months. However, as LLMs improved and traditional benchmarks like HellaSwag [65], MMLU [61], and ARC [71] became more familiar, the leaderboard faced challenges related to benchmark saturation. Benchmark saturation occurs when models excel on overused benchmarks, leading to inflated performance metrics that may not reflect true model capabilities. This highlighted the need for more challenging and novel datasets to better assess model performance. [72]

In response, Hugging Face introduced the Open LLM Leaderboard v2 [73], incorporating more rigorous benchmarks such as MMLU-Pro [60], GPQA [74], and MuSR [75]. These benchmarks test deeper reasoning abilities, knowledge recall, and instruction-following capabilities, with a focus on tasks that align more closely with human preferences. This upgraded version addresses previous limitations by providing a more accurate reflection of a model's general capabilities.

LLM-as-a-Judge: Mixing both Approaches

While the Open LLM Leaderboard offers a robust solution to some of the presented challenges, the LLM-as-a-Judge framework introduces an innovative approach that could provide a more comprehensive assessment of model quality. This framework leverages strong LLMs not only as test-takers but also as evaluators, combining the strengths of human evaluation without the need for a human judge.

The LLM-as-a-Judge framework operates on the premise that a strong enough LLM can evaluate the outputs of other, weaker models. In this setup, models generate open-text responses to benchmark questions, reflecting real-world applications where LLMs are expected to produce coherent and

contextually appropriate text. The responses are then evaluated by the judge model with a specialized system prompt. This system prompt encourages Chain of Thought (CoT) Reasoning to enhance the quality of the evaluation as well as the guidance to present the score in a fixed format at the end of the output. This ensures that the score can be extracted via Reg-Ex Pattern Matching.

The framework offers several key advantages:

- **Alignment with Real-World Applications:** The open-text format allows for a more comprehensive demonstration of an LLM's capabilities, better reflecting the types of tasks these models are expected to perform in real-world settings.
- **Consistency and Scalability:** By using an LLM as the judge, the evaluation process becomes more consistent and scalable than traditional human evaluations. This standardization makes it possible to apply the same benchmarks across multiple model settings and at larger scales.
- **Versatility:** The framework is capable of evaluating performance across a variety of tasks and domains. The same judge model can be employed for different questions and contexts, enhancing its utility as a benchmarking tool.

This approach represents a significant step forward in LLM benchmarking, integrating the depth of open-text evaluation with the efficiency of automated testing. By incorporating LLMs into the evaluation process itself, the LLM-as-a-Judge framework promises more accurate and efficient assessments of model quality, ultimately contributing to a better comparison of different models and model settings [76], [77], [57].

The Arena-Hard-Auto Benchmark

The arena-hard-auto benchmark [78] introduced by the LMSYS team is a practical implementation of the LLM-as-a-Judge method. It is designed to evaluate models based on 500 particularly challenging open-text questions across various knowledge domains. This benchmark ensures a thorough and statistically grounded comparison of different models, providing a robust test of LLM capabilities [76], [77].

Generating the Responses

Before the actual evaluation can begin, all the tested models need to generate responses to the 500 benchmark questions. To streamline this process, the framework offers an automated script that prompts selected models asynchronously in parallel. In addition to these models, a selected baseline model should also generate the responses. This model will function as a base to compare each model to individually. Its score will always be set to 50 with a base ELO rating of 1000. The preset baseline model is `GPT-4-0314`, the first version of the `GPT-4-`Model by OpenAI.

Evaluation Process and Judge Model

The evaluation process for each model and each question begins with the judge model being prompted with a specific system prompt and the relevant question. The judge model is also provided with responses from both the tested and baseline model. The task is then to decide which answer to the specified question is better. To mitigate possible biases in this decision process the model names are blinded—both models are only referred to as `model_a` and `model_b`. The system prompt and prompt template used in the arena-hard-auto benchmark can be found in Appendix A.

The judge model is prompted to answer the question prior to evaluating both responses. Furthermore, it is encouraged to use a chain-of-thought reasoning approach before giving its final verdict. Both aim to improve the quality of the evaluation, ensuring that the reasoning behind the final decision is transparent and robust. The idea behind this approach is that the answer tokens up to that point are also used as context for each newly generated token. In other words, the text of the already generated answer

influences the probability of the subsequently generated text. This means that both the prior answer and the step-by-step reasoning influence the probabilities for the final verdict at the end of the generated text, leading to an overall more accurate evaluation. [79]

Result Parsing and Battle Setup

After the judge model completes the evaluation, the results are parsed to categorize the outcomes as follows:

- `[[A»B]]` for a decisive win by Model A.
- `[A>B]` for a clear but less pronounced win by Model A.
- `[A=B]` indicating a tie.
- `[B>A]` and `[[B»A]]` for wins by Model B, with the latter being more decisive.

Each outcome is assigned a weight. For example, decisive wins (`[[A»B]]` or `[[B»A]]`) are weighted by a factor of 3, while other outcomes are unweighted (assigned the value 1). The battle dataset is then expanded accordingly. For example, if a decisive win (`[[A»B]]`) occurs 200 times, it is translated to 600 wins for model A in the dataset (200 instances multiplied by the weight of 3). Finally, the transformed dataset only contains three entries:

- `output["winner"] = "tie"`
- `output["winner"] = "model_a"` ($\times$ `WEIGHT`)
- `output["winner"] = "model_b"` ($\times$ `WEIGHT`)

This transformation allows for a better handling of the scores in the following calculation steps.

ELO Score Calculation

The ELO score calculation begins by setting up a logistic regression model. Here's how the process works step by step:

1. **Constructing the Design Matrix** X: Each battle between models M_a and M_b is represented in a design matrix X, where:

$$X_{i,j} = \begin{cases} \log(\text{BASE}) & \text{if } j = M_a \\ -\log(\text{BASE}) & \text{if } j = M_b \\ 0 & \text{otherwise} \end{cases}$$

   This matrix encodes which models participated in each battle.

2. **Setting up the Outcome Vector** Y: The outcome of each battle is stored in a vector Y, where $Y_i = 1$ if M_a wins and $Y_i = 0$ if M_b wins. For ties, each battle is split into two, with one assigned as a win for M_a and the other as a win for M_b.

3. **Performing Logistic Regression**: Logistic regression is applied to estimate the ELO coefficients:

$$\text{ELO score} = \text{SCALE} \times \text{Coefficient} + \text{INIT\_RATING}$$

   Where `SCALE` is typically 400 and `INIT_RATING` is 1000.

4. **Adjusting ELO Scores**: The ELO scores are adjusted so that the baseline model has a fixed score of 1000:

$$\text{ELO}_{\text{adjusted}} = \text{ELO} + (1000 - \text{ELO}_{\text{baseline}})$$

Win Rate and Score Presentation

Once ELO scores are computed, they are used to predict win rates:

$$P(M_a \text{ wins over } M_b) = \frac{1}{1 + 10^{\frac{\text{ELO}_b - \text{ELO}_a}{400}}}$$

The win rate is then converted to a score on a 0-100 scale, where a score of 50 corresponds to an ELO of 1000:

$$\text{Score (0-100)} = 100 \times P(M_a \text{ wins over } M_b)$$

Bootstrapping for Confidence Intervals

To estimate uncertainty in these scores, bootstrapping [80], [81] is used. The dataset is sampled with replacement, and ELO scores are recalculated for each bootstrap sample. The 95% confidence interval is derived from the distribution of these scores:

$$\text{CI}_i = \left(\text{Percentile}_{2.5}(r_i^{(b)}), \text{Percentile}_{97.5}(r_i^{(b)})\right)$$

High Agreement with Human Evaluations

The arena-hard-auto benchmark demonstrates a high level of agreement with human evaluation metrics. The LMSYS team reported that this benchmark achieved a state-of-the-art 89.1% agreement with human preference rankings, making it a reliable predictor of downstream performance and validating its efficacy in distinguishing between LLMs in a manner that aligns closely with human judgment. As a reference, the agreement scores between different human judges are as low as 80%, indicating that the observed results lie within an expected deviation between human judges. [76]

Limitations and Mitigation Steps of the Benchmarks

Despite its strengths, the Arena-Hard-Auto benchmark has some limitations, notably a bias of judge models towards the same model family (e.g., `GPT-4o` has a bias towards models from the `GPT-4-`Family) and a positional bias based on the order in which assistant responses are presented. To mitigate the first issue, the benchmark employs a swapping mechanism, where each question is run twice—once with the baseline as Assistant A and once as Assistant B. This results in a total of 1000 evaluations (500 questions x 2), balancing the positional biases and providing a more accurate assessment of model performance.

2.5 Customization Approaches

Customization approaches are methods used to tailor large language models (LLMs) to specific tasks or improve their output quality and efficiency. This subsection explores several key techniques, including model fine-tuning, knowledge distillation, model pruning, quantization, and prompt engineering with knowledge embedding. Each method offers unique benefits in enhancing model quality and reducing computational requirements, thereby contributing to decreased energy consumption during model deployment. By examining these approaches, the potential for optimizing LLMs for specific applications while addressing energy efficiency concerns becomes evident.

2.5.1 Model Fine-Tuning

Fine-tuning is a fundamental process for adapting pre-trained LLMs to various downstream tasks. By updating the model parameters with task-specific data, fine-tuning enhances the model's output quality on targeted applications such as sentiment analysis, machine translation, and question answering. Despite its effectiveness, fine-tuning presents several challenges, including computational cost, catastrophic forgetting, and the non-transferability of model-specific tuning parameters to new models.

One significant concern is *catastrophic forgetting*, wherein a model *"forgets"* previously learned information upon learning new tasks [82]. Kalajdzievski proves that fine-tuning not only leads to a reduction in a model's generalization and reasoning capabilities, but it also affects the safety guardrails negatively. These guardrails are embedded during pre-training to prevent the generation of undesirable or even harmful text. Although incorporating separate guardrail models to monitor outputs is a possible mitigation strategy, the risk of catastrophic forgetting remains a critical factor in the decision to fine-tune a model.

Another significant challenge is the sunk cost associated with fine-tuning due to its inherent model-specificity. The additional training performed is tailored to the model's architecture and parameters, rendering tuned parameters from one model non-transferable to newer or different models. Even methods like Low-Rank Adaptation (LoRA), which introduce extra adapter layers, are tied to the specific models they were trained on, limiting their applicability across models.

Nevertheless, fine-tuning still allows for significant increases in model output quality for very specialized use cases. Due to this potential, various fine-tuning approaches have been developed to enhance efficiency and mitigate forgetting as much as possible. These methods include full fine-tuning, parameter-efficient fine-tuning (PEFT), prompt tuning, LoRA, Quantized LoRA (QLoRA), and Weight-Decomposed Low-Rank Adaptation (DoRA).

Full fine-tuning involves updating all the parameters of the pre-trained model. While this method can achieve high performance on downstream tasks, it is computationally intensive and requires substantial storage for each fine-tuned model instance, which becomes impractical for very large models with billions of parameters.

To reduce computational and storage costs, **parameter-efficient fine-tuning** methods [83] update only a small subset of the model's parameters or add lightweight modules. Examples include adapter layers introduced by Houlsby et al. [84], which add small bottleneck layers between existing layers, and *prompt tuning* [85], which prepends trainable vectors (soft prompts) to the model's input without modifying the model architecture.

Low-Rank Adaptation (LoRA) [86] is a PEFT method that addresses the inefficiencies of full fine-tuning by freezing the original model weights and injecting trainable rank-decomposition matrices into each layer of the Transformer architecture. LoRA represents the weight updates as low-rank matrices, significantly reducing the number of trainable parameters. For instance, LoRA can reduce the number of trainable parameters by up to 10,000 times compared to full fine-tuning and decrease GPU memory requirements during training by a factor of three. An important advantage of LoRA is that it does not introduce additional inference latency because the low-rank updates can be merged with the original weights after training.

Quantized LoRA (QLoRA) [87] builds upon LoRA by applying 4-bit quantization to the frozen pre-trained model during fine-tuning. This further reduces memory usage and computational requirements, enabling the fine-tuning of very large models on hardware with limited resources without significant performance degradation.

Recently, **Weight-Decomposed Low-Rank Adaptation (DoRA)** [88] was proposed to enhance the learning capacity and training stability of LoRA. DoRA decomposes the pre-trained weights into magnitude and direction components and employs LoRA to adapt the directional component efficiently. By focusing on directional updates, DoRA aims to replicate the learning capacity of full fine-tuning while maintaining the parameter efficiency of LoRA. Experimental results indicate that DoRA consistently outperforms LoRA on various tasks without introducing additional inference latency.

Despite these advancements, the issue of catastrophic forgetting remains unresolved. Furthermore, the tuned parameters are model-specific and cannot be directly transferred to different models, perpetuating the sunk cost associated with fine-tuning.

Understanding how fine-tuning scales with compute resources, dataset size, and model size is crucial for efficiently adapting LLMs to downstream tasks. Recent research by Zhang et al. [89] explores these scaling laws in the context of fine-tuning. A detailed assessment of this research is presented in Section 2.6.3 of this thesis.

2.5.2 Knowledge Distillation

Knowledge distillation [90] is a process in which the knowledge from a large, complex model (the teacher model) is transferred to a smaller, more efficient model (the student model). This technique enables the student model to replicate the performance of the teacher model while being computationally cheaper and faster, making it especially valuable for large-scale inference scenarios.

Distillation is frequently used by organizations that provide foundation models across various sizes, such as the LLaMA [19], [20], [91] families. In these cases, the largest model in the family is trained first, and smaller models are trained on the outputs of the larger models. While distillation requires the full training of a new model, which is extremely resource-intensive, it is justified in large-scale scenarios where the model inference costs significantly outweigh the model training costs.

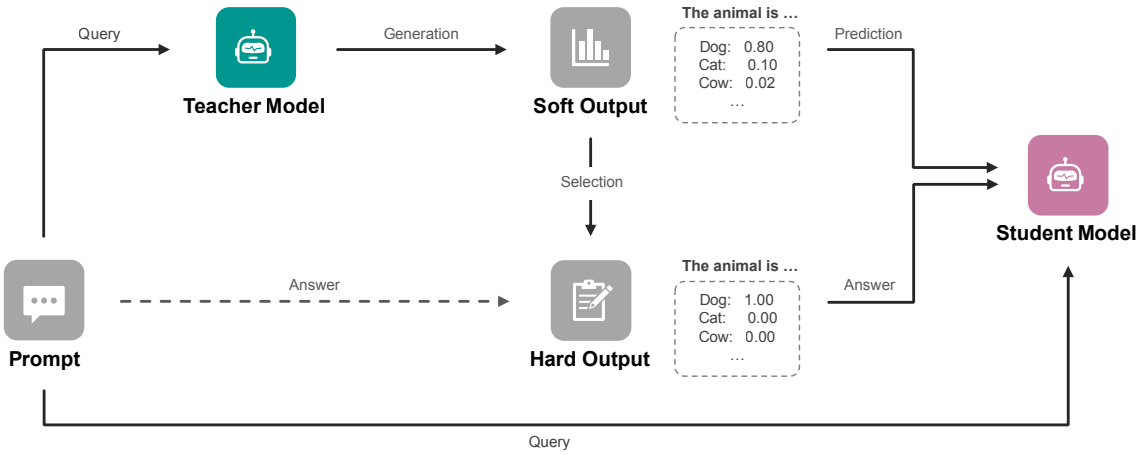

Figure 5: Illustration of the Model Distillation Process

The distillation process (see Figure 5) begins with the teacher model generating two types of outputs: the *soft output* and the *hard output*. The soft output is a probabilistic distribution that reflects the teacher model's confidence across all possible classes or tokens in the vocabulary. For example, the teacher may assign an 80% probability to the word *dog*, 10% to *cat*, and 2% to *cow*. This information is valuable because it highlights not just the most likely class but also the relationships between classes—such as how *dog* and *cat* are more semantically related than *dog* and *cow*. In contrast, the hard output is the

final predicted class based on the highest probability. In this example, the hard output would be *dog* with 100% certainty, as it has the highest confidence.

The student model is then trained to predict "what the teacher would output" in response to a given input. It learns using two types of losses:

1. **Distillation Loss (Soft Output)**: This loss helps the student model learn from the teacher's soft outputs by minimizing the difference between the student's predicted distribution and the teacher's distribution. This process allows the student to capture additional semantic information, such as the relationships between different classes.

2. **Cross-Entropy Loss (Hard Output)**: This loss helps the student model learn to predict the correct class, just like in traditional supervised learning. The student compares its predicted class with the teacher's hard output or the ground truth, ensuring it can make accurate final predictions.

Knowledge distillation is particularly useful for training smaller versions of large models in scenarios where the smaller models will be used for large-scale inference. While the process is resource-intensive during training, the benefits of faster inference, reduced energy consumption, and smaller model sizes make it practical when models are deployed at scale.

Recent research papers on knowledge distillation, such as DistilBERT [92] and MiniLLM [93], have demonstrated the effectiveness of this technique in creating smaller, more efficient models while retaining much of the original model's performance.

DistilBERT, a smaller version of BERT, successfully reduced the original model's size by 40%, from 110 million parameters in BERT-base to 66 million parameters in DistilBERT. Despite this reduction, it retained 97% of BERT's performance on downstream tasks and achieved a 60% faster inference speed. Its success was driven by a combination of three loss functions: masked language modeling (MLM) loss, distillation loss, and cosine-distance loss, which allowed the student model to better mimic the teacher's behavior across tasks like sentiment analysis and question answering.

Similarly, MiniLLM applied knowledge distillation to compress large language models like GPT-2 and GPT-J, reducing their size by 60-80% while maintaining 90-95% of the teacher model's performance. MiniLLM introduced reverse Kullback-Leibler (KLD) divergence to focus the student model on the most relevant parts of the teacher's output, improving generalization on tasks like long-text generation and summarization.

2.5.3 Model Pruning

Model pruning is a technique used to reduce the size of a neural network by removing parameters (weights) that have minimal impact on generating the desired output. This method is particularly relevant for general-purpose LLMs, which are often over-parameterized for specific tasks. In over-parameterized models, many weights may not significantly contribute to the accuracy of the model's predictions, making them candidates for removal. By pruning such unimportant weights, the model can be made more efficient without substantial losses in output quality.

The primary benefit of pruning is the ability to reduce computational requirements, which in turn decreases the energy consumption associated with running large models. Reducing the number of active parameters directly lowers the number of calculations the model needs to perform during inference, resulting in faster execution and lower energy consumption. This is particularly important for LLMs, whose deployment at scale often requires significant computational resources. As highlighted by Ma et al. [94], pruning techniques can significantly reduce the energy consumption and carbon footprint of these models.

The key concept in pruning is that not all weights in a model contribute equally to its outputs. Some weights play a more crucial role in generating accurate predictions, while others may be redundant or unnecessary. This unequal importance of weights makes it possible to remove less important ones without severely affecting model performance. By analyzing which weights are least important for a given task, those parameters can be removed or set to zero, effectively reducing the model's size and complexity [94].

The basic mechanism of pruning involves identifying the weights that contribute the least to the model's accuracy and setting them to zero. This process creates sparsity in the weight matrices, which reduces the number of active parameters and, consequently, the computational cost of running the model. This approach is particularly useful in LLMs that are designed to handle a wide range of tasks. In multi-purpose models, certain weights may be crucial for some tasks but unnecessary for others. This allows for task-specific pruning, where parameters that are not relevant to a particular task can be safely removed, tailoring the model to perform better in specialized use cases [95].

Pruning has a direct impact on the size of a model and its energy efficiency. Techniques like LLM-Pruner, as described by Ma et al. [94], can reduce the number of parameters by up to 20% while maintaining over 94% of the model's original quality (measured in multiple-choice benchmarks). These size reductions translate to faster inference times and reduced energy consumption, as fewer parameters need to be processed during each inference. However, more aggressive pruning strategies may lead to noticeable performance losses. Models that undergo higher pruning ratios often require post-tuning to regain some of the performance that was lost during the pruning process. Therefore, while moderate pruning can offer significant benefits with minimal trade-offs, more aggressive approaches need to be balanced with additional fine-tuning [95].

Pruning serves as a foundation for further optimization techniques. Some Quantization techniques capitalize on the idea that not all parameters need to be represented with the same precision, leaving out critical weights in the quantization to retain as much performance as possible. This method will be discussed in more detail in the following section.

2.5.4 Quantization

Quantization is a technique for reducing the computational complexity of large models by representing weights and activations with fewer bits. It shares similarities with pruning in that it seeks to reduce model size and computational cost. However, instead of removing parameters, quantization reduces the precision with which parameters are represented. This can significantly reduce the size of large models, making their inference faster and more energy efficient.

Several different quantization techniques exist, each offering unique advantages. Huang et al. [32] provide a comprehensive comparison of these methods, applying them to models such as LLaMA-3 and LLava-Next. Two of the most widely used techniques are AWQ (Activation Weight Quantization) [33] and GPTQ (Gradient Post-Training Quantization) [34]. Another promising approach is FP8 quantization, where model weights are represented using an 8-bit floating-point format [35].

FP8 quantization introduces two distinct 8-bit floating point representations to maximize quality retention while maintaining computational efficiency:

- **E4M3**: This format uses one sign bit, four exponent bits, and three mantissa bits. It can represent values ranging from ± 448 and nan (not-a-number). E4M3 is typically useful in scenarios where the range of values is smaller but precision is important. [96]
- **E5M2**: This format consists of one sign bit, five exponent bits, and two mantissa bits. It can represent values up to ± 57344, including infinity and nan. This format offers a much larger dynamic range, though at the cost of reduced precision. [96]

The idea of quantization is to reduce the size of the model weights without causing significant losses in output quality. For instance, Huang et al. [32] suggest that 8-bit quantization results in almost no quality degradation. Moving to 4-bit quantization introduces a minimal drop in quality, typically around 1-2%, depending on the quantization method and benchmark. These tests are done on standardized multiple-choice benchmarks and offer a great indication of the quality retention that can be expected during quantization. Yet, the tests are suboptimal in reflecting real-world use cases accurately. Therefore, the research in chapter 5.6 is dedicated to providing an assessment that mimics the practical use case of LLMs more closely.

When going below 4-bit quantization, much more pronounced quality degradation is observed. Quantization at this level is typically not suitable for most use cases due to the extreme precision loss, but it can still be applied in highly optimized edge scenarios. Achieving functional performance with such low-precision quantization requires exceptionally advanced techniques, very precise scaling, and post-quantization fine-tuning.

An example of hyper-optimized inference setups can already be seen in production systems. Apple's recent showcase of their ML models, branded as Apple Intelligence, provides an outlook on the future of model inference optimizations. During the beta phase of their release, a 3 billion parameter model ran at 30 tokens per second with a time-to-first-token of just 0.6 milliseconds on an iPhone 15 with only 8GB of RAM. Even faster inference speeds are expected for the official release. Apple has disclosed that the models utilize mixed-precision quantization at 2 bits and 4 bits with an average of 3.5 bits per weight, allowing for drastic memory footprint reductions without significant decreases in quality [97], [98]. Although techniques such as Apple's are promising in edge scenarios, most quantization frameworks commonly apply 8-bit and 4-bit quantization techniques, which are more broadly applicable and maintain a balance between efficiency and quality retention.

For instance, in FP8 quantization, accuracy can be recovered with relatively simple methods like rounding-to-nearest (RTN) quantization. Furthermore, dynamic on-the-fly scaling can be utilized during inference. In that scenario, the activations have their minimum and maximum values calculated before each forward pass (the generation of each token). These values are then used to scale the quantized weights. In contrast, static quantization involves scaling the weights prior to its usage. This is done based on a batch of data that mimics the expected inputs and outputs for the final use case. It ensures that the activations represent the typical range of values the model will encounter, especially during text generation in specific domains. Proper scaling of the weights is crucial, as it allows for optimal representation of values within a smaller bit range, preserving output quality.

Handling outliers is another critical factor in scaling quantized weights. Outliers, extreme values that fall at the edges of the value range, can distort this process and lead to quality degradation if not properly accounted for. Suppose only a small number of weights or activations lie at the far end of the value range. In that case, special handling is required to ensure the overall scaling doesn't compress the other values too aggressively, which could negatively impact output quality.

Building upon the theories of model pruning, Lin et al. [33] have proposed the idea of *salient weights*. These weights contribute disproportionately high to the model's quality and must be preserved at higher precision in mixed-precision quantization techniques. The original paper expects about 0.1% to 1% of a model's weights to be considered *salient* and highly critical for ensuring the model's output quality. By assigning higher precision to these salient weights, mixed precision methods ensure that the model retains as much quality as possible, even when lower precision is applied to the majority of the weights. This concept is, for example, the backbone of the AWQ 4-bit quantization framework.

Reducing the precision from 16-bit to 8-bit reduces the model size by half, and from 8-bit to 4-bit halves the size again, allowing for significant reductions in model size and energy consumption. Output quality

retention is key, and modern GPU architectures natively support INT8 and FP8 computations, which further accelerates processing at these lower precisions compared to 16-bit precision. This hardware acceleration makes 8-bit precision particularly desirable, as it strikes a balance between reduced model size, higher computational efficiency, and retained output quality.

2.5.5 Prompt Engineering and Knowledge Embedding

Prompt Engineering has become a rapidly evolving research domain in the field of LLMs. This domain aims to design effective input prompts that enable LLMs to generate more accurate and relevant responses. In the last few years, numerous novel frameworks and automated enhancement algorithms have emerged, all suggesting considerable potential for improving model output quality without fine-tuning or retraining the model. Chen et al. [99] provide a comprehensive meta-analysis of Prompt Engineering, which acts as a guide for this chapter.

Before diving into advanced techniques, it is important to address a key precondition: *correct spelling and clear wording* in prompts. While this measure seems trivial, it offers substantial potential, if not already done correctly. It ensures that the model receives a clear and unambiguous input for the correct processing of information. When Prompt Engineering is considered, this should be one of the first steps applied before advancing with further techniques. [100]

All of the further techniques can then be clustered into three distinct categories:

1. **Prompt Structuring**: Focuses on enhancing the prompt structure to allow for better outputs. Notable Frameworks and Ideas: Utilization of Prompt Templates [101] & [21], XML-Tags [102], Role Prompting [103]

2. **Reasoning Enforcement**: Focuses on encouraging the model to output step-by-step reasoning prior to the final output. Notable Frameworks and Ideas: Chain-of-Thought (CoT) [79], Tree of Thoughts (ToT) [104], Answer-Prefilling [105]

3. **In-Context Learning (Knowledge Embedding)**: Focuses on providing the model with additional information as context to solve the task. Notable Frameworks and Ideas: Retrieval-Augmented Generation (RAG) [36], Multi-Shot Prompting [106], Automatic Reasoning and Tool Usage (ART) [107]

Each of the categories will be described in this chapter.

Prompt Structuring

Prompt structuring involves the strategic arrangement of input to optimize a model's ability to generate relevant and coherent outputs. One commonly employed technique is the use of prompt templates [22], [101], which structure prompts in a standardized format to guide the model. For instance, templates specify how to phrase questions or present contextual information, ensuring that the model processes inputs systematically.

In current instruction-tuned models, it is standard practice to define three roles within the prompt template: system, user, and assistant. Models fine-tuned on such templates excel in user-query-based tasks, particularly in dialogue systems. These models can handle multiple pairs of user-assistant messages, structuring conversations effectively. An example prompt template for the LLaMA 3.1 model is shown below:

```
<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>

    Cutting Knowledge Date:  December 2023
    Today Date:  23 September 2024

    You are a helpful assistant.

<|eot_id|>

<|start_header_id|>user<|end_header_id|>

    What is the capital of France?
<|eot_id|>

<|start_header_id|>assistant<|end_header_id|>

    [ ... ]
```

Figure 6: LLaMA 3.1 Prompt Template by Meta [21]

Another useful technique is the application of XML-Tags, which allow explicit instructions to be embedded into the prompt. For example, tags such as <question> and <context> can clearly delineate different sections of the input, helping the model to focus on relevant parts of the text. This results in more accurate and targeted outputs [102]. Similarly, role prompting, where the model is assigned a specific role (e.g., "teacher" or "scientist"), further enhances the model's capacity to generate responses that align with user expectations by tailoring its behavior to the assigned role [103].

These prompt structuring techniques are particularly valuable when the model needs to generate creative or detailed outputs, such as in storytelling, answering complex user queries, or solving multi-step problems. By structuring the prompt effectively, the user can guide the model to produce coherent, contextually relevant responses while minimizing the generation of irrelevant content.

Reasoning Enforcement

Chain of Thought [79] is probably one of the most notable reasoning enhancement frameworks. It proves to enhance the outputs of LLMs specifically for reasoning tasks [108], [109] and sparked several new approaches to optimize the framework even further [110], [111], [112], [104]. While each of these approaches introduces unique improvements, they all share the same foundational concept: the model is guided to generate step-by-step reasoning before delivering a final conclusion.

The efficacy of these reasoning frameworks can be illustrated by tests conducted for the *checkmate_in_one* task in the BIG-bench benchmark, designed by Srivastava et al. [113]. This task presents a language model with a chessboard in algebraic notation and requires it to identify the correct move that would result in a checkmate. While increasing the model size did not result in better test results, they noticed that the accuracy for predicting correct chess moves, in general, increased steadily with the model size. This led to the conclusion that larger models, in fact, contained a better "understanding" of chess concepts. Yet, the models were not able to output the correct results out of the box. Larger models contain more information on the concepts of chess, but basic prompting fails to extract the necessary knowledge to solve the task. However, reasoning frameworks such as CoT helped overcome this limitation by prompting the model to generate intermediate reasoning steps, leading to a notable increase in accuracy for predicting the correct move, as shown by Yang et al. [112].

A possible hypothesis explaining the effectiveness of reasoning frameworks like CoT is based on how the model's token prediction process operates. When a model generates new tokens, it incorporates previously generated tokens as part of its input. This means that each new token influences the probabilities of the tokens that follow. By generating intermediate reasoning steps, the model uses these tokens to adjust and refine its predictions as it moves toward the final output. This doesn't imply that the model is "thinking" through the problem step-by-step in a human-like fashion. Instead, the generated reasoning tokens help steer the model toward more accurate conclusions by modifying the context and guiding the probabilities for subsequent predictions.

Researchers from Google DeepMind have contributed additional insights into reasoning frameworks through their novel work on decoding techniques [114]. In this research, they aimed to enforce reasoning in LLMs without requiring highly structured prompts. Typically, language models use *greedy decoding* to predict the next token in a sequence, which often favors concise, direct answers over more thorough and reasoned outputs. As discussed by Wang and Zhou, this decoding method can be limiting for tasks that benefit from more detailed reasoning. To address this, they developed a custom *top-k decoding* strategy for the first token that is generated. This method allowed the model to explore less certain tokens at the beginning of its reasoning sequence, promoting more in-depth reasoning. Once the model had started its output with a less certain token, standard greedy decoding was resumed, resulting in more accurate and coherent outputs (see Figure 7).

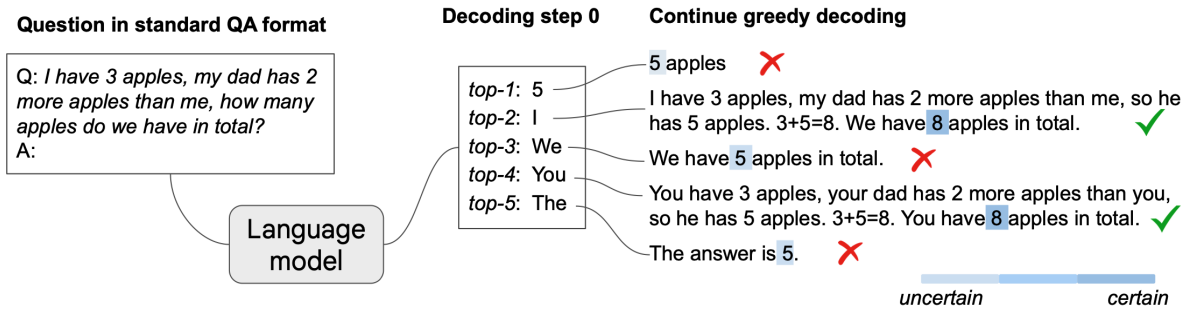

Figure 7: Illustration of CoT-decoding (Source: Wang and Zhou [114])

This experiment highlights that models frequently possess the necessary information to solve complex tasks but may struggle to generate the correct output without appropriate guidance. Reasoning frameworks, such as CoT, provide this guidance by steering the model toward generating the most relevant output sequences.

A significant evaluation of reasoning methods was conducted through the *InstructEval* benchmark [115]. This benchmark compared various prompting methods across different models and settings. It was concluded that reasoning frameworks like CoT produced the most noticeable improvements in *Zero-Shot* scenarios, where the model is tasked with solving a problem without prior examples or context. In contrast, *Few-Shot* scenarios, where the model is provided with several example question-answer pairs, showed that reasoning frameworks had a smaller impact. This suggests that few-shot settings inherently guide models to better outputs without needing additional reasoning frameworks. These results further reinforce the importance of reasoning techniques, particularly in situations where models must solve novel tasks with limited prior context.

Another useful technique for enforcing structured outputs is Answer-Prefilling [105]. In this method, parts of the expected output are pre-filled to help the model complete the response in a desired format.

This is especially useful when the output requires specific formatting or when particular steps must be followed, such as in mathematical calculations or structured lists. By providing the model with a skeleton of the expected response, answer-prefilling ensures that the generated text follows a logical and coherent structure.

In-Context Learning (Knowledge Embedding)

In-Context Learning (ICL) or knowledge embedding refers to techniques where additional information is provided within the prompt to help the language model generate more accurate and contextually relevant outputs. Unlike reasoning enforcement, which encourages the model to arrive at an answer through step-by-step reasoning, ICL focuses on embedding task-specific knowledge directly into the prompt itself. This can be achieved through strategies like multi-shot prompting, where examples of questions and corresponding answers are included in the prompt, or more advanced methods such as Retrieval-Augmented Generation (RAG).

A key concept behind ICL is that LLMs, though trained on vast datasets, may not always have the necessary, task-specific information readily available for every prompt. Instead of relying solely on the model's internalized knowledge, ICL strategies dynamically inject relevant information into the prompt to help the model generate accurate responses. This technique has proven especially valuable in cases where the model needs to handle domain-specific tasks or answer questions about up-to-date events that are beyond the model's training data.

RAG is a framework that combines the generation abilities of LLMs with retrieval-based systems. In this setup, an external knowledge base (such as a document store or a database) is queried in real-time to retrieve relevant information that is then embedded into the prompt. This allows the model to utilize the most up-to-date and relevant information when generating a response. For instance, in customer support or medical applications, a RAG system can retrieve the latest documents or guidelines, embedding them into the model's input so that the LLM can generate answers that reflect the latest knowledge available. The combination of retrieval and generation enables LLMs to overcome the limitations of static knowledge embedded during their training, making them far more adaptable to real-world use cases [36].

In multi-shot prompting, the model is given multiple examples of a task (e.g., questions and answers) within the same prompt to help guide it toward generating the correct response. This approach is particularly useful for tasks where the model might struggle with zero-shot learning (i.e., tasks where it has not been provided any explicit examples beforehand). By giving the model a few examples, the prompt provides it with the context necessary to complete the task more effectively. Multi-shot prompting is highly adaptable and has been shown to improve model performance on a wide range of NLP tasks, from text classification to machine translation [8].

Another methodology that emerged together with the popularizing of LLM Agents is the method of Automatic Reasoning and Tool Usage (ART) [107], [116], [117], [118]. This method describes how LLMs can be prompted to use specific pre-defined tools. LLaMA 3.1, for example, introduced a completely new role called `ipython`[21]. This role is a skeleton that describes the output values of called functions within a chat conversation. The basic idea of ART is that LLMs are given a set of functions they can call within the system prompt. The list of tools usually contains a comprehensive tool description and a list of function arguments the LLM needs to generate. Current models are then able to output which tool should be used as well as the arguments that should be provided. The function call is then parsed in the generation pipeline (through eg. RegEx pattern matching). An example function definition can be found in the following illustration.

```
"name": "get_delivery_date",
"description": "Get the delivery date for a customer's order. Call
this whenever you need to know the delivery date, for example when a
customer asks 'Where is my package'",
"parameters": {
    "type": "object",
    "properties": {
        "order_id": {
            "type": "string",
            "description": "The customer's order ID.",
        },
    },
    "required": ["order_id"],
    "additionalProperties": false,
}
```

Figure 8: Example function definition for GPT-4o from OpenAI [119]

It is important to note that the definition in Figure 8 is only a definition for the LLM (in this case GPT-4o). After the model calls this function, the call needs to be caught. After the call is caught the function then needs to be executed with the return value being forwarded back to the model. Afterwards, the function calls work similar to a RAG process. The model is fed the output of the function as additional context to answer a user given question.

In-Context Learning has proven highly effective in improving the quality of model outputs, particularly for tasks that require domain-specific knowledge or up-to-date information. By embedding relevant examples, facts, or tools into the prompt, users can guide LLMs to generate responses that are more accurate and relevant to the task at hand. This has far-reaching implications for industries that require real-time updates and specialized knowledge, such as healthcare, finance, and customer service. ICL techniques continue to evolve, with ongoing research exploring new ways to incorporate external knowledge and reasoning capabilities into LLMs.

2.6 Scaling Laws and Calculation Frameworks

Scaling laws provide valuable insights into how the performance of large language models (LLMs) depends on factors such as model size, dataset size, and computational budget. Understanding these relationships enables researchers to make informed decisions about allocating resources effectively. This subsection explores the theoretical underpinnings of scaling laws and presents calculation frameworks for estimating compute requirements. It begins by detailing how to calculate the floating-point operations (FLOPs) involved in training transformer-based LLMs. Subsequently, it discusses how IsoFLOPs curves can be used to predict optimal training settings under fixed computational budgets.

2.6.1 Calculating Compute Requirements

Estimating the computational requirements for training large language models is crucial for efficient resource allocation and model optimization. The computational cost is often measured in floating-point operations (FLOPs), representing the number of arithmetic operations needed during training. Understanding how to calculate FLOPs enables researchers to predict training time and hardware needs accurately.

A commonly used estimation for the total number of FLOPs required to train a transformer-based LLM is:

$$C_{\text{total}} \approx 6ND,$$

where N is the number of non-embedding parameters in the model, and D is the total number of training tokens. This estimation originates from analyses by Kaplan et al. [25] and is elaborated upon by Bahdanau [120] and Casson [121], who provide comprehensive explanations of FLOPs calculations in transformer models.

The factor of 6 accounts for the operations performed during both the forward and backward passes of training. To derive this estimation, the FLOPs required for each component of the transformer architecture are analyzed. In the forward pass, the main computational components are the embedding layer, the attention mechanism, the feedforward network, and the output layer.

The **embedding layer** maps input tokens and positions to continuous vectors. The FLOPs required here are relatively small compared to other components and are often neglected in large models.

The **attention mechanism** involves several operations. The *QKV projections* use linear layers to project the input into queries (Q), keys (K), and values (V). The FLOPs per token for these projections are calculated as:

$$C_{\text{QKV}} \approx 2N_{\text{layer}}d_{\text{model}}(3d_{\text{attn}}),$$

where N_{layer} is the number of layers, d_{model} is the model dimension, and d_{attn} is the attention dimension (typically equal to d_{model}). The factor of 2 accounts for the multiply-accumulate operations in matrix multiplications.

The *attention scoring and masking* compute attention scores by taking the dot product of queries and keys and applying a mask. The FLOPs for this operation per token are:

$$C_{\text{Attn\_Scores}} \approx 2N_{\text{layer}}n_{\text{ctx}}d_{\text{attn}},$$

where n_{ctx} is the context length. This term scales with the context length but becomes negligible in large models where $d_{\text{model}} \gg n_{\text{ctx}}$ [8]. Especially in use cases where the context length of a model is scaled up to significant amounts of well over 10.000 tokens, this contributes to the non-linear scaling of energy consumption for output token increases (see section 5.3). Nevertheless, this factor contributes insignificantly in most use cases and is therefore often disregarded.

The *output projection* projects the concatenated attention outputs back to the model dimension. The FLOPs per token are:

$$C_{\text{Attn\_Proj}} \approx 2N_{\text{layer}}d_{\text{attn}}d_{\text{model}}.$$

The **feedforward network** (FFN) consists of two linear layers with a non-linear activation in between. The FLOPs per token for the FFN are:

$$C_{\text{FFN}} \approx 4N_{\text{layer}}d_{\text{model}}d_{\text{ff}},$$

where $d_{\text{ff}} = 4d_{\text{model}}$ is the dimension of the feedforward network.

The **output layer** projects the model outputs to logits over the vocabulary. The FLOPs per token are:

$$C_{\text{Logits}} \approx 2d_{\text{model}}n_{\text{vocab}},$$

where n_{vocab} is the size of the vocabulary. In large models, this term is small compared to the total FLOPs and can often be neglected.

By summing the FLOPs of these components and neglecting smaller terms like nonlinearities and biases, the total FLOPs per token for the forward pass simplify to:

$$C_{\text{forward}} \approx 2N.$$

The backward pass requires computing gradients for all parameters, involving additional computations. It is generally estimated that the backward pass requires approximately twice the FLOPs of the forward pass [26], due to the need to compute gradients with respect to both weights and activations. Therefore, the FLOPs per token for the backward pass are:

$$C_{\text{backward}} \approx 4N.$$

Combining both passes, the total FLOPs per token during training are:

$$C_{\text{total per token}} \approx C_{\text{forward}} + C_{\text{backward}} \approx 6N.$$

Multiplying by the total number of training tokens D, the overall computational cost is:

$$C_{\text{total}} \approx 6ND.$$

For example, consider a model with $N = 175 \times 10^9$ parameters (similar to GPT-3 [8]) trained on $D = 300 \times 10^9$ tokens. The total computational cost would be:

$$C_{\text{total}} \approx 6 \times 175 \times 10^9 \times 300 \times 10^9 = 3.15 \times 10^{23} \text{ FLOPs}.$$

An alternative method for calculating FLOPs is presented by DeepMind in the Chinchilla paper [26]. Their approach provides a detailed breakdown of the FLOPs per sequence for each operation, including embeddings, attention mechanisms (with softmax and masking), and output logits. While this method includes additional components, the total FLOPs estimated are comparable to the simplified calculation, especially for large models where certain terms become insignificant.

Casson [121] discusses these methods extensively and offers practical insights into FLOPs calculations. His blog post includes code examples and a FLOPs calculator for transformer models. The Medium article by Bahdanau [120] also provides a clear derivation of the FLOPs equations and explains each component in detail. These resources are excellent for understanding the nuances of FLOPs estimation.

In practice, additional factors can influence the total computational cost. For instance, if activation checkpointing is employed to manage memory limitations, the backward pass may require recomputing activations, effectively adding extra forward passes during backpropagation. This increases the total FLOPs, potentially raising the factor from 6 to 8 in the estimation [121].

Understanding FLOPs calculations is essential for measuring model efficiency and hardware utilization. The approximation $C_{\text{total}} \approx 6ND$ is instrumental in predicting the computational resources required for training LLMs. It highlights the relationship between model size, dataset size, and computational budget, allowing researchers to make informed decisions about scaling models efficiently.

Moreover, this estimation is crucial when considering hardware limitations and optimizing training configurations. Knowing the FLOPs helps in calculating the Model FLOPs Utilization (MFU), which measures how efficiently the required model FLOPs are executed relative to the theoretical peak FLOPS of the hardware [122]. MFU is calculated as:

$$\text{MFU} = \frac{C_{\text{total}} \times D_{\text{throughput}}}{P},$$

where $D_{\text{throughput}}$ is the observed throughput in tokens per second, and P is the theoretical peak FLOPS of the hardware. This metric is valuable for assessing hardware efficiency and guiding optimizations in model training.

While the quadratic terms in attention computations (those scaling with n_{ctx}^2) can be significant for small models or very long context lengths, they become less impactful as models scale up in size [8]. For large models like GPT-3, the context-dependent FLOPs account for less than 3% of the total FLOPs, justifying their omission in the simplified estimation. However, for models with extremely long context lengths or specialized applications, these terms may need consideration.

Accurate estimation of FLOPs is vital for the efficient training of transformer models. The approximation $C_{\text{total}} \approx 6ND$ provides a practical guideline, balancing simplicity and precision. The resources by Bahdanau [120] and Casson [121] offer deeper insights into the theoretical and practical aspects of FLOPs estimation in transformer models.

2.6.2 Predicting Optimal Training Settings Using IsoFLOPs Curves

Building upon the understanding of computational requirements, researchers have developed scaling laws to predict how the performance of large language models depends on factors like model size, dataset size, and computational budget. These scaling laws enable informed decisions on how to allocate resources for optimal model performance.

Meta applied these refined scaling laws in developing LLaMA 3 [21], utilizing IsoFLOPs curves to predict optimal training settings. IsoFLOPs curves are graphical representations that illustrate the relationship between model performance and model size for a fixed computational budget measured in floating-point operations. Each curve represents a constant compute budget. By plotting validation loss against model size, one can identify the optimal model size that minimizes the loss for that budget.

To generate these curves, models of varying sizes are trained on different amounts of data, all constrained to the same total computational cost. The shape of the IsoFLOPs curves reveals how model size and training data quantity trade off against each other. The minimum point on each curve indicates the model size and data quantity that yield the best performance under the specified compute budget.

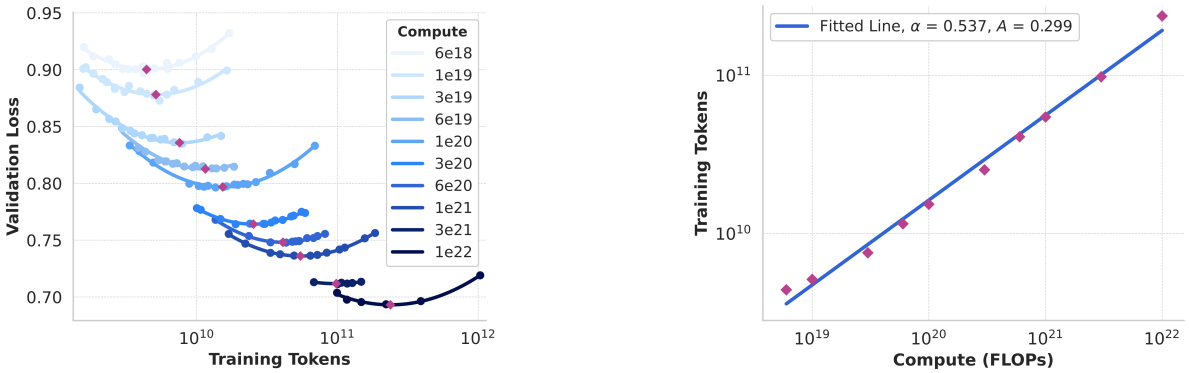

Figure 9: IsoFLOPs curves showing validation loss versus model size for different compute budgets (left). Regression on the number of training tokens for compute-optimal models (right). (Source: Meta[21])

An important observation made by Meta [21] is that larger models exhibit increased robustness to changes in training settings, particularly concerning the balance between model size and the number of training tokens. As the compute budget increases, the IsoFLOPs curves become flatter near their minima (see Figure 9). This flattening indicates that performance becomes less sensitive to small deviations from the optimal model size or training data quantity. Therefore, at larger scales, there is greater flexibility in choosing model parameters without significantly compromising performance.

To predict optimal training settings, Meta conducted extensive scaling experiments by training models ranging from 40 million to 16 billion parameters across compute budgets from 6×10^{18} to 1×10^{22} FLOPs [21]. They assumed a power-law relationship between the compute budget C and the optimal number of training tokens $D^*(C)$:

$$D^*(C) = AC^{\alpha},$$

where A and α are empirically determined constants. From their experiments, they found:

$$A = 0.299 \qquad\qquad \alpha = 0.537.$$

The total compute budget is related to the model size N (number of parameters) and the number of training tokens D^* by:

$$C = 6ND^*,$$

where the factor of 6 approximates the number of FLOPs required per parameter per token during training (see Section 2.6.1).

Using these relationships, Meta calculated the optimal number of training tokens and model size for their desired compute budget. Specifically, for a compute budget of $C = 3.8 \times 10^{25}$ FLOPs, they predicted that training on $D^* = 16.55$ trillion tokens would be ideal. By rearranging the previous equation, they calculated the optimal model size as:

$$N = \frac{C}{6D^*}.$$

Substituting the values for C and D^*, they found the optimal model size to be approximately $N = 402$ billion parameters. Based on this prediction, Meta decided to train their flagship model with 405 billion parameters, aligning closely with the computed optimal size.

The slight discrepancy between the calculated optimal model size and the one chosen by Meta may be due to several factors. Practical constraints such as hardware limitations and training stability may influence the choice of model size. Additionally, the flattening of IsoFLOPs curves at higher compute budgets indicates that performance becomes less sensitive to exact model size [21]. Therefore, selecting a model with 405 billion parameters would still achieve near-optimal performance within the compute budget. Empirical adjustments and considerations of diminishing returns may also lead to choosing a model size that balances performance improvement and computational efficiency.

Using IsoFLOPs curves and scaling laws provides a systematic framework for predicting optimal training settings for large language models. The application of these principles by Meta [21] demonstrates their reliability and accuracy. While replicating such extensive experiments to calculate novel A and α values is beyond the scope of this thesis, adopting them allows for informed predictions about optimal model configurations. Thus, the literature values will be used in the later creation of the proposed framework.

2.6.3 Applying Scaling Laws to Model Fine-Tuning

Understanding how scaling laws apply to model fine-tuning is essential for optimizing large language models (LLMs) for specific tasks. Recent research by Zhang et al. [89] investigates how various scaling factors influence the quality of LLM outputs via fine-tuning. The study focuses on factors such as model size, pre-training data size, fine-tuning data size, and the number of trainable parameters in parameter-efficient tuning (PET) methods.

Zhang et al. conducted systematic experiments to explore these scaling effects. They considered two primary fine-tuning methods: full-model tuning (FMT) and PET methods, including prompt tuning [85] and Low-Rank Adaptation (LoRA) [86]. Their study concentrated on the data-limited regime, where the fine-tuning data size is much smaller than the model size, reflecting practical scenarios in deploying LLMs.

They pretrained two sets of bilingual LLMs: English–German (En–De) and English–Chinese (En–Zh), with model sizes ranging from 1 billion to 16 billion parameters. The pre-training data consisted of up to 283 billion tokens for En–De and up to 206 billion tokens for En–Zh. The models were fine-tuned on downstream tasks, including machine translation (WMT14 En–De and WMT19 En–Zh) and multilingual summarization (MLSum), using varying amounts of fine-tuning data from 8,000 to 25 million examples.

The researchers proposed a multiplicative joint scaling law to model the relationship between fine-tuning loss and the scaling factors:

$$\hat{L}(X, D_f) = A \left(\frac{1}{X^{\alpha} D_f^{\beta}} \right) + E,$$

where $\hat{L}(X, D_f)$ is the predicted loss, X represents a scaling factor (LLM model size, pre-training data size, or PET parameter size), D_f is the fine-tuning data size, and A, α, β, and E are empirically determined constants. This scaling law captures how output quality improves with increasing X and D_f, effectively modeling the interaction between these factors.

Key findings from the study include:

First, fine-tuning data scaling follows a power law. For a fixed model size, the fine-tuning loss decreases as a power law with respect to the fine-tuning data size:

$$\hat{L}(D_f) = AD_f^{-\beta} + E.$$

This indicates that increasing the amount of fine-tuning data consistently enhances a model's output quality. Importantly, in their experiments, the authors did not observe a point where the benefits of increasing fine-tuning data diminished. This suggests that, within their test setup, adding more fine-tuning data continued to yield improvements. However, this does not mean that such a saturation point does not exist; it may be that their experiments did not reach that level.

Second, LLM fine-tuning benefits more from model size scaling than from pre-training data scaling. The scaling exponent α for model size (α_m) is larger than that for pre-training data size (α_p), indicating that increasing the model size has a greater impact on output quality than increasing the pre-training data size. Therefore, allocating resources to larger models may yield better output quality in fine-tuning.

Third, scaling the PET parameter size yields diminishing returns. Increasing the number of trainable parameters in PET methods, such as extending the prompt length in prompt tuning or increasing the rank in LoRA, has minimal impact on output quality. The scaling exponent α_t for PET parameter size is close to zero, indicating a weak scaling relationship. Moreover, larger PET configurations can lead to training instability, particularly in prompt tuning.

Fourth, the optimal fine-tuning method depends on the task and data availability. The scaling trend and actual values are highly dependent on the downstream task, and critical points for one task may not generalize to others. The existence of such points suggests that the selection of fine-tuning methods should be based on the availability of fine-tuning examples. When only a few thousand fine-tuning examples are available, PET methods should be considered first, either prompt tuning or LoRA. With slightly larger datasets, LoRA would be preferred due to its stability and better scalability with fine-tuning data. For million-scale datasets, full-model tuning is recommended.

In conclusion, the scaling laws presented by Zhang et al. provide valuable insights into the fine-tuning of LLMs. By recognizing the interplay between model size, pre-training data, fine-tuning data, and tuning parameters, practitioners can optimize LLMs for specific tasks more effectively. Understanding these scaling relationships allows for informed decisions when allocating resources for model fine-tuning, potentially leading to better output quality without unnecessary computational expenses.

2.7 Existing Carbon Footprint Estimates of Generative AI Models

The development of large language models (LLMs) has raised significant concerns regarding their computational demands and associated carbon emissions. Estimating the carbon footprint of these models is crucial for understanding their environmental impact and for guiding efforts to mitigate it.

Luccioni et al. [18] conducted a comprehensive study estimating the carbon footprint of BLOOM, a 176-billion-parameter multilingual language model developed by Hugging Face. They assessed BLOOM's carbon emissions throughout its life cycle, including model training, equipment manufacturing (embodied emissions), and deployment.

The authors found that training BLOOM's final model consumed approximately 433,196 kWh of electricity, resulting in emissions of about 24.7 tCO_2 equivalents (CO_2eq) when considering only dynamic power consumption. Accounting for all processes, including hardware manufacturing and idle power consumption, the total emissions amounted to 50.5 tCO_2eq.

Notably, Luccioni et al. also provided calculations for the inference phase of BLOOM. By deploying the model via an API endpoint on a cloud platform, they monitored the energy consumption over 18 days. They observed that even when the model was not processing any requests, it continued to consume a

significant amount of energy to maintain the model in memory. Their analysis showed that approximately 70% of the energy consumption during inference could be attributed to idle consumption. This finding highlights the importance of considering idle energy usage when deploying large models, as it constitutes a substantial portion of the operational carbon footprint.

In addition to BLOOM, other large-scale models have begun to report their energy consumption and emissions. OpenAI released estimates for training GPT-3, a 175-billion-parameter model [8]. They reported that training GPT-3 consumed about 1,287 MWh of electricity, resulting in emissions of approximately 552 tCO_2eq when accounting for the power usage effectiveness (PUE) of the data center.

Meta has also published detailed reports on the energy consumption and emissions for the training phases of their LLaMA models [19], [20], [22]. In the original LLaMA paper [19], they provided estimations of the carbon footprint for their models. Training the LLaMA-7B model required 82,432 GPU-hours on NVIDIA A100-80GB GPUs with a thermal design power (TDP) of 400 W [123], consuming 36 MWh of electricity and resulting in emissions of 14 tCO_2eq, assuming a U.S. average carbon intensity of 0.385 kg CO_2eq/kWh and a PUE of 1.1. Training the LLaMA-65B model required 1,022,362 GPU-hours, consuming 449 MWh and resulting in emissions of 173 tCO_2eq.

In LLaMA 2 [20], Meta provided updated estimates. Pretraining utilized a cumulative 3.3 million GPU-hours on A100-80GB GPUs with a TDP of 350–400 W. Specifically, the LLaMA 2 7B model required 184,320 GPU-hours, consuming 81 MWh of electricity, and resulted in emissions of 31.22 tCO_2eq. The LLaMA 2 70B model required 1,720,320 GPU-hours, consuming 758 MWh, leading to emissions of 291.42 tCO_2eq.

For LLaMA 3 [21], Meta continued to report emissions. Training LLaMA 3 8B required 1.3 million GPU-hours on H100-80GB GPUs with a TDP of 700 W [124], resulting in emissions of 390 tCO_2eq. Training LLaMA 3 70B required 6.4 million GPU-hours, emitting 1,900 tCO_2eq.

In their latest release, LLaMA 3.1 [21], Meta reported that training the LLaMA 3.1 8B model utilized 1.46 million GPU-hours on H100-80GB GPUs, resulting in emissions of 420 tCO_2eq. The LLaMA 3.1 70B model required 7 million GPU-hours, emitting 2,040 tCO_2eq.

Comparing the energy consumption and emissions of models with similar sizes across different versions of LLaMA reveals significant increases over time. To illustrate this trend, a comparison of the 7/8B models and the 65/70B models is presented in chronological order in Table 3.

Model	**Parameters**	**GPU Hours**	**Energy Consumed**	**Emissions**
LLaMA [19]	7B	82,432	36 MWh	14 tCO_2eq
LLaMA 2 [20]	7B	184,320	81 MWh	31.22 tCO_2eq
LLaMA 3 [21]	8B	1,300,000	910 MWh	390 tCO_2eq
LLaMA 3.1 [21]	8B	1,460,000	1,022 MWh	420 tCO_2eq
LLaMA [19]	65B	1,022,362	449 MWh	173 tCO_2eq
LLaMA 2 [20]	70B	1,720,320	758 MWh	291.42 tCO_2eq
LLaMA 3 [21]	70B	6,400,000	4,480 MWh	1,900 tCO_2eq
LLaMA 3.1 [21]	70B	7,000,000	4,900 MWh	2,040 tCO_2eq

Table 3: Comparison of energy consumption and emissions for LLaMA models of similar sizes.

Table 3, demonstrates the substantial increase in energy consumption and emissions for models of similar sizes in successive versions. The 7/8B models show an increase in GPU-hours from 82,432 for LLaMA-7B to 1.46 million for LLaMA 3.1 8B. Correspondingly, emissions increased from 14 tCO_2eq to 420 tCO_2eq.

A similar trend is observed for the 65/70B models. GPU-hours increased from 1.02 million for LLaMA-65B to 7 million for LLaMA 3.1 70B, and emissions increased from 173 tCO_2eq to 2,040 tCO_2eq.

This evolution in energy consumption and emissions may be attributed to several factors. One primary factor is the increase in the amount of training data and the number of training epochs used in newer versions to improve model performance. Additionally, the transition from NVIDIA A100 GPUs to more powerful H100 GPUs with higher TDPs (700 W compared to 400 W) contributes to higher energy consumption [123], [124]. The H100 GPUs offer increased computational capabilities, allowing training on larger datasets or with more complex training regimes but at the cost of higher power consumption.

Moreover, as models evolve, they often incorporate architectural improvements and advanced training techniques that may increase computational complexity. While these enhancements can lead to better model performance, they also contribute to longer training times and increased energy consumption, even for models with similar parameter counts.

These observations highlight the importance of optimizing training processes to reduce energy consumption and carbon emissions. Meta, for example, has implemented methods to predict optimal training settings before actual training, thereby minimizing trial and error. By utilizing scaling laws and IsoFLOPs curves, as described in chapter 2.6.2, they are able to identify optimal "sweet spots" in model configurations.

In summary, the estimation of the carbon footprint of generative AI models is becoming an integral part of model development and deployment. By reporting energy consumption and emissions, organizations like Hugging Face, OpenAI, and Meta contribute to a better understanding of the environmental challenges associated with large-scale AI models. The comparison of emissions across models of similar sizes demonstrates the need for continued efforts to optimize training efficiency and reduce the environmental impact of large-scale AI models.

2.8 Gap in the Field of Research

Despite significant advancements in understanding and mitigating the carbon footprint of Large Language Models (LLMs) during their training phases, there remains a noticeable gap in the literature concerning the energy consumption and emissions associated with LLM inference. Training large-scale models, such as GPT-3 [8] and BLOOM [18] and the LLaMA model family [19], [20], [21], has been extensively studied, with detailed reports on computational requirements and environmental impacts. These studies have led to the development of scaling laws and optimization techniques aimed at reducing emissions during training (see Chapter 2.6).

However, the inference phase—where models are deployed to generate outputs for user queries—constitutes a substantial portion of the total energy consumption over an LLM's lifecycle [18]. Inference can be particularly resource-intensive due to the deployment of models with billions of parameters, necessitating continuous computational power to serve real-time requests. The energy consumption during inference not only depends on the model size but is also influenced by customization strategies, deployment architectures, and usage patterns.

The assessment conducted on the BLOOM model by Luccioni et al. [18] provides valuable insights by analyzing the end-to-end lifecycle emissions, including those from inference. Their work highlights that idle energy consumption during inference accounts for a significant portion of total emissions, emphasizing the need for efficient deployment strategies. However, this analysis is limited to a single model and does not encompass the diverse range of LLM architectures and customization techniques currently available.

Moreover, while techniques such as quantization (see Chapter 2.5.4), Prompt Engineering and RAG Knowledge Embedding (see Chapter 2.5.5) have been proposed to enhance the efficiency of LLMs, there

is a lack of comprehensive studies evaluating their impact on energy consumption during inference. Existing research primarily focuses on the output quality implications of these techniques rather than their environmental benefits. Consequently, practitioners lack a clear understanding of how to implement these methods to optimize energy efficiency without compromising model quality.

Therefore, there is a pressing need for a systematic framework that enables the evaluation and optimization of LLM inference from an energy consumption and emissions perspective. Such a framework should consider the interplay between model customization techniques, deployment strategies, and environmental factors. By addressing this gap, researchers and practitioners can make informed decisions to reduce the carbon footprint of LLM deployments, contributing to more sustainable AI practices.

This thesis aims to fill this gap by analyzing the potential of various customization approaches—such as quantization and retrieval-augmented generation—in reducing energy consumption during LLM inference. By providing empirical evaluations and proposing best practices for efficient LLM deployment, this work seeks to advance the understanding of sustainable AI development and encourage the adoption of environmentally friendly techniques in the field.

3 METHODOLOGY

This section presents the methodological approach adopted to analyze how customizations to Large Language Models (LLMs), such as quantization and retrieval-augmented generation (RAG), influence energy consumption and output quality. The experimental setup is described, including the infrastructure design and the tools used for measuring energy consumption during inference. Details of the model quality benchmarks are provided, explaining the evaluation framework and metrics applied. Finally, a decision framework is proposed to guide practitioners in optimizing LLM deployments for efficiency and sustainability.

3.1 Infrastructure Design and Experimental Setup

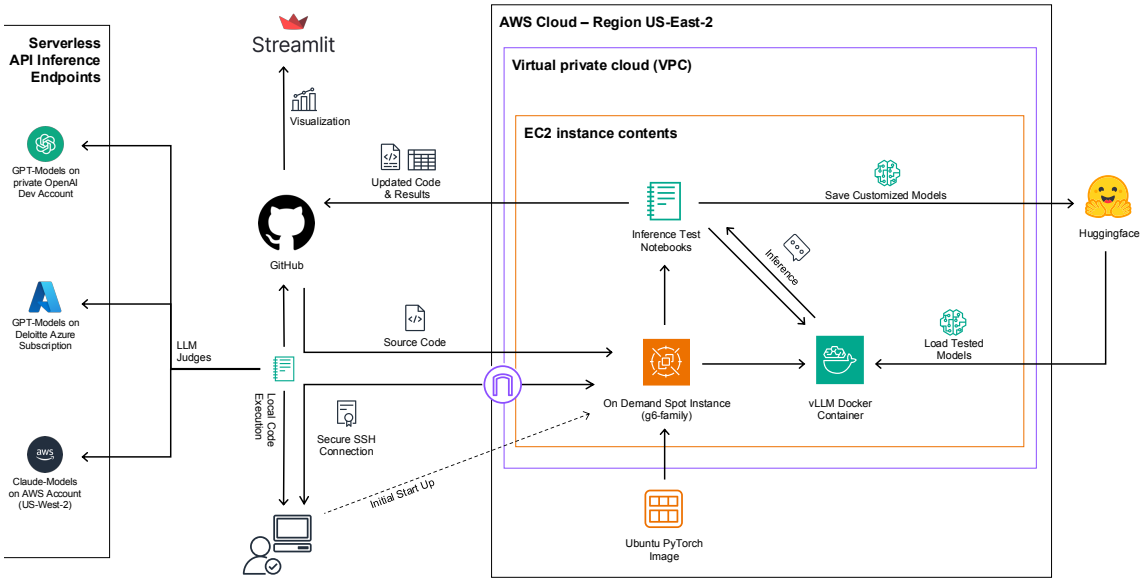

Figure 10: Architecture of the inference test infrastructure used for experiments

The architecture implemented for the inference tests was designed to prioritize cost-effectiveness, scalability, and cloud-agnostic flexibility. The complete architecture is depicted in Figure 10. Rather than adhering strictly to cloud architecture best practices, the setup was constructed as a fast and adaptable environment to conduct inference tests while minimizing cloud compute costs.

A key design decision was to avoid the use of cloud provider-specific features, thereby maintaining cloud agnosticism. This approach ensures that the codebase is transferable across different cloud environments without modification. For instance, despite the availability of AWS-exclusive services such as SageMaker JumpStart—which provides internal access to a wide range of LLMs—the architecture relies on HuggingFace's open-source repository to load pre-trained models. This choice ensures that models can be easily accessed and tested across various cloud platforms.

The serverless API endpoints for inference were connected to multiple LLM providers, including GPT models running on a private OpenAI development account, GPT models hosted on a Deloitte Azure subscription, and Claude models running on AWS in the `us-west-2` region.

AWS was selected as the primary infrastructure provider for the cloud compute environment. Tests were conducted in the `us-east-2` region within a Virtual Private Cloud (VPC). The EC2 instances were of the `g6-family` running Ubuntu 22.04, rather than Amazon Linux, to reinforce the cloud-agnostic nature

of the architecture. To reduce costs, Spot Instances were chosen instead of dedicated EC2 instances. By using those the overall costs could be reduced 65% to 85%. To streamline the setup of the EC2 instances, the pyTorch Ubuntu image preset was selected. This image contains pyTroch with all of its dependencies and drivers necessary to utilize the GPU acceleration.

Each EC2 instance was controlled via a secure SSH connection, which handled remote code execution and environment setup. A startup script automated the installation of development libraries, dependencies, and the initialization of the GitHub repository containing all necessary source code and submodules. This automation allowed for seamless instance deployment with minimal manual intervention. The code itself was written and executed in VSCode which allows for remote code editing using the SSH connection.

Inference testing involved running Python Scripts and Jupyter Notebooks on the host machines as well as using vLLM Docker Containers to serve the customized models. The results were pushed back to the GitHub repository, ensuring that the codebase remained up to date with the latest findings. After each session, the instance's SSD storage was deleted to further reduce costs, necessitating a full re-initialization of the instance for subsequent sessions.

A Streamlit dashboard served as the central hub for visualizing test results. It pulled updated code and results from the GitHub repository and provided interactive visualizations for each inference test. The dashboard was initially hosted in the Streamlit Community Cloud, but later migrated to a privately hosted webserver for better availability. The hosting of this webserver is not part of this research and will therefore not be described further.

The architecture was designed to be flexible regarding the models and data it could process, enabling the use of customized models. Upon completion of inference, the tested models could be saved back to HuggingFace, allowing for further reuse or deployment.

In summary, the architecture combined several components and connections to achieve its objectives. The serverless API endpoints interfaced with multiple LLM providers, allowed for the utilization of the currently leading models acting as LLM Judges in the quality benchmarks. GitHub played a crucial role in hosting the source code and the results, serving as an essential tool for version control. HuggingFace was employed for loading and saving models, offering large file support to house various language models. The remote code execution on AWS EC2 instances allowed for the necessary flexibility in compute demands for different model sizes and settings. Streamlit was chosen for the visualization because of the fast development of interactive dashboards to visualize the data.

The integration of these components, along with the use of Spot Instances, temporary storage, and automated startup scripts, rendered the architecture both cost-efficient and highly scalable, with minimal overhead in terms of setup time and maintenance.

3.2 Energy Consumption Measurement Methodology

Accurately measuring energy consumption during the inference phase of Large Language Models (LLMs) is crucial for quantifying their environmental impact and identifying optimization opportunities. Inference involves real-time processing of diverse inputs and can constitute a significant portion of an LLM's total energy consumption [18]. Consistent measurement methodologies are essential for reliable comparisons and analyses.

In this thesis, the CodeCarbon library [46] was employed to estimate energy consumption during inference. CodeCarbon is an open-source tool that monitors the utilization of computational resources such as CPUs and GPUs, considering hardware specifications and workload characteristics. It is suitable for cloud-based environments where direct hardware measurements are impractical due to limited access to physical infrastructure [49].

During initial measurements, the CPU and RAM energy consumption readings were consistently low and exhibited minimal variation, regardless of workload intensity. This raised concerns about the accuracy of CodeCarbon's measurements for these components. To investigate this issue, system metrics were monitored using Weights & Biases [125], confirming that CPU and RAM utilization remained low during inference tasks. The low energy readings reflected actual resource usage rather than measurement errors. This observation is attributed to LLM inference being highly GPU-intensive, with CPUs and RAM playing minor roles in computational loads. Cloud instances, such as the AWS EC2 instances used in the experiments (see Section 3.1), are provisioned with balanced CPU and RAM relative to GPU capabilities. For LLM inference, this configuration results in over-provisioned and underutilized CPU and RAM resources.

Energy consumption measurements focused on GPUs, CPUs, and RAM. Other components, such as network interfaces and storage devices, were not included due to the complexities of measuring their energy usage in virtualized cloud environments. On cloud platforms, different instances may share physical resources among multiple users, making it difficult to attribute energy consumption accurately. Additionally, while the largest instances fully utilize the physical server, comparing energy usage across different instance sizes is complicated due to virtualization layers and resource sharing. Therefore, energy consumption measurements were standardized to the primary computational components to maintain consistency.

Although CodeCarbon can estimate carbon emissions based on energy consumption and the energy mix at the server's location, this thesis did not utilize the tool's emission estimates. The energy grid's carbon intensity varies throughout different times of day and year. Since the experiments were conducted over an extended period of time, comparing emissions between tests would be unreliable due to these fluctuations. To maintain consistency and comparability, the analysis focused solely on energy consumption.

Standardization was essential to ensure fair comparisons across different tests and configurations. The following measures were implemented:

- **Serving Engine and Model Size Tests:** Energy consumption was measured for processing 7,500 prompts (Sections 5.1 and 5.2).
- **Output Token Length Tests:** Energy consumption was reported per 10,000 prompts to assess the impact of varying generated output token lengths (Section 5.3).
- **Input Token Length Tests:** Energy consumption was standardized per 100,000 generated output tokens to exclude the impact of varying answer lengths on the tests (Section 5.4).
- **Quantization and Prompt Engineering Tests:** Energy consumption was measured over the entire arena-hard-auto benchmark, consisting of 500 questions asked twice (totaling 1,000 prompts). The same set of guidance prompts was pre-generated and used consistently across all models to ensure that differences in energy consumption or performance were due to experimental variables (Sections 5.6 and 5.5).

Inference tests were conducted on AWS EC2 instances equipped with NVIDIA GPUs. Instances were dedicated solely to inference tasks to eliminate interference from other processes and background activities. CodeCarbon was integrated into the inference workflow to monitor energy consumption continuously during task execution.

By focusing on energy consumption and standardizing experimental conditions, the measurements provided a reliable basis for analyzing how model size, input and output token lengths, quantization levels, and prompt engineering techniques influence energy usage. These insights informed the development of optimization strategies and the decision framework presented in Chapter 6.

3.3 Model Quality Evaluation Procedures

Evaluating the output quality of LLMs is crucial for understanding the impact of various customization techniques on performance. In this study, the LLM-as-a-Judge framework, detailed in Section 2.4, was utilized to assess model quality effectively. Specifically, the automated arena-hard-auto benchmark (discussed in Section 2.4) was employed, with GPT-4o serving as the judge model and GPT-4-0314 as the baseline [78]. This approach aligns with established methodologies for evaluating LLMs.

The GPT-4o model was selected as the judge due to its superior performance among available LLMs at the time of testing. Using GPT-4-0314 as the baseline ensured consistency with prior studies, maintaining continuity with the original benchmark framework [78]. The benchmark comprises 500 particularly challenging questions across diverse domains, providing a comprehensive evaluation of model capabilities. The experimental setup involved running each question twice to account for positional biases, resulting in a total of 1,000 evaluations. Statistical methods, including confidence interval calculations, were applied to ensure the reliability of the results.

An interactive visualization of the benchmark process is available within the Streamlit dashboard on the 'Explain Benchmark' page [1]. This tool aids in understanding the evaluation procedure and the comparative performance of different models.

To evaluate quantization quality retention, tests were conducted across 4-bit and 8-bit quantization levels for the LLaMA 3.1 70B, LLaMA 3.1 8B, and Mistral Nemo models. The quantization techniques applied are discussed in Section 2.5.4. This selection allowed for testing across three different model sizes and two distinct model families, providing insights into how quantization affects models of varying scales and architectures. Each model was tested with and without quantization as well as with and without guidance to assess the impact on output quality.

For the prompt engineering and retrieval-augmented generation (RAG) tests, additional models were included: GPT-4o, GPT-4o-mini, Mistral Large 2407, LLaMA 3.1 405B, and LLaMA 3 70B. The methodologies for prompt engineering and RAG are detailed in Section 2.5.5. Including these models enabled a broader analysis of how advanced prompting techniques influence model performance across a spectrum of capabilities.

It is important to note that GPT-4o was used to generate the guidance prompts as well as to perform the evaluation. While this may introduce potential biases, the benchmark was designed to mitigate such effects. The evaluation was conducted blindly. This means that the judge model did not have access to the identities of the models generating the answers, as model names were anonymized (e.g., `model_a` and `model_b`). Furthermore, the judge model could not access the guidance provided to the models, ensuring that assessments were based solely on the outputs.

Prior to conducting the benchmarks involving guidance, the GPT-4o model was prompted to automatically generate guidance for each question based on a reference answer, also produced by GPT-4o. This guidance included relevant background information, step-by-step plans, practical tips, and additional considerations. The use of GPT-4o for both guidance generation and evaluation was carefully managed to maintain the integrity of the assessment process.

In summary, the use of the LLM-as-a-Judge framework with the arena-hard-auto benchmark provided a robust method for evaluating model quality across various customization techniques. By testing multiple models and configurations, and applying rigorous statistical analysis, the study gained valuable insights into the impact of quantization, prompt engineering, and model scaling on LLM performance.

3.4 Development of the Decision Framework

Implementing LLMs efficiently and sustainably requires a systematic approach that balances performance, energy consumption, and cost. Based on the empirical findings from the experiments, a comprehensive decision framework was developed to guide practitioners in optimizing LLM deployments. The framework addresses key factors that significantly impact energy efficiency and model quality, including model customization and implementation strategies.

The decision framework consists of six sequential steps. First, precise requirements for the LLM deployment are established. This involves specifying model quality goals, setting energy consumption targets, determining budget limits, identifying available hardware resources, and understanding expected usage patterns. Defining these parameters provides a foundation for informed decision-making.

Second, the hosting strategy is selected, deciding between self-hosting the model or utilizing external APIs. Self-hosting offers control and customization but may lead to energy inefficiency if the model is underutilized. Alternatively, external APIs provide shared infrastructure and pay-per-use models, enhancing energy efficiency through higher hardware utilization. They may, however, lack transparency regarding the hosting environment, making it difficult to assess the carbon footprint.

Third, customization options are evaluated to enhance model performance or reduce energy consumption. Quality-enhancing customizations include advanced prompt engineering and retrieval-augmented generation (RAG). These techniques can significantly improve model outputs, especially for models with lower baseline performance, but may increase energy consumption due to longer input sequences. Energy-reducing customizations include quantization, model size selection, and input/output token length optimization. Quantization reduces computational demands by lowering numerical precision, leading to substantial energy savings with minimal loss in model quality.

Fourth, experiments with different configurations are performed to assess their impact on model quality and energy consumption. Energy consumption is estimated using established calculation frameworks, and model quality is evaluated using automated benchmarks. Operational costs, including energy expenses and hardware costs, are calculated to ensure alignment with budget constraints.

Fifth, trade-offs between model quality, energy consumption, and cost are analyzed to make informed decisions. Determining if the quality loss from quantization or using a smaller model is acceptable for the application is crucial. Visualizing different configurations can aid in comparing options and selecting the optimal solution that best satisfies the initial requirements and constraints.

Finally, the selected solution is implemented according to a detailed deployment plan. Continuous monitoring is essential to ensure the deployment meets performance and efficiency targets. Regular evaluation allows for adjustments and further optimization as needed. Building a flexible platform that allows easy integration of new models and technologies ensures continuous adaptation to advancements in the field, maintaining optimal performance and efficiency over time.

To assist practitioners in applying this decision framework, an interactive dashboard was designed. The dashboard consolidates the quantitative results from the experiments into regression models that capture the relationships between energy consumption and key factors affecting LLM inference. Users can input parameters such as model size, quantization levels, token lengths, and serving engine choices to visualize the impact of these factors on energy consumption. This tool facilitates exploring different optimization strategies, providing actionable insights without requiring exact hardware specifications.

Overall, the decision framework provides a structured methodology for optimizing LLM implementations. By following these steps and utilizing the interactive dashboard, practitioners can achieve a balance between performance and sustainability.

4 PRELIMINARY STUDIES / PRE-TESTS

Before conducting the main experiments, a series of preliminary tests were carried out to verify whether the theoretical assumptions on model inference efficiency aligned with actual performance. These initial tests aimed to explore the relationship between model configurations, such as parameter size and token lengths, and energy consumption during inference. However, the results did not align with the mathematical projections, prompting further investigation. This chapter presents the findings from that research, focusing on the key bottlenecks that were identified in model inference efficiency.

4.1 Investigating Bottlenecks in Model Inference Efficiency

In the initial phase of this research, a series of experiments were conducted using HuggingFace's widely known Transformers library [126]. The objective of these tests was to determine the relationship between model configurations and energy consumption during inference. Specifically, three different setups were tested: varying model parameter sizes, adjusting the input token lengths, and adjusting the output token lengths.[1] However, contrary to expectations, the energy consumption did not increase proportionally with the mathematical projections based on floating-point operations per second (FLOP/s) (see Chapter 2.6). This anomaly indicated the presence of significant bottlenecks that impede theoretical computational efficiency. These bottlenecks lead to a noticeable disparity between the predicted and actual energy consumption and computation time during inference, highlighting the need for a deeper investigation of the underlying causes.

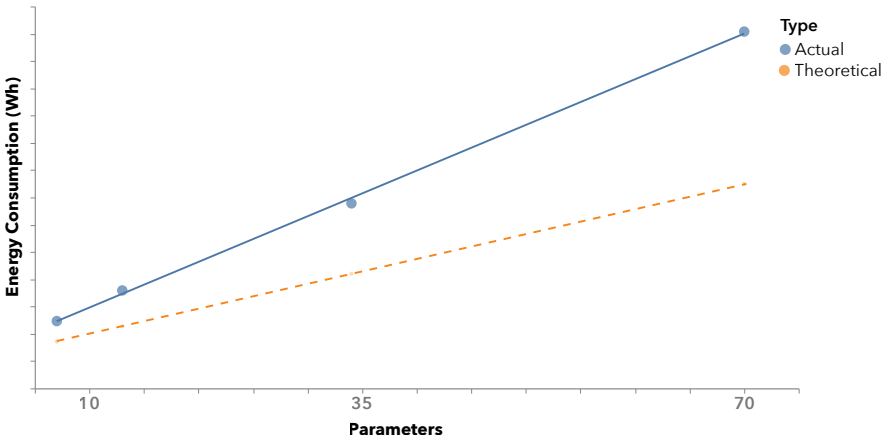

Figure 11: Energy based on theoretical Complexity Increases vs. Actual Energy Consumption

As illustrated in Figure 11, the increase in model parameter sizes did not result in a proportional increase in energy consumption. Complexity refers to the number of calculations required, measured in floating-point operations per second (FLOPs). Despite the expected rise in computational demands with larger model sizes, the observed energy consumption increase was significantly higher than anticipated. When comparing the expected processing time based on the maximum FLOPs that NVIDIA L4 GPUs can achieve (121 teraFLOPS/s[127]), this discrepancy becomes even more pronounced.

Considering the test scenario of generating approximately 1,000,000 output tokens (as in the model size benchmarks in Section 5.2), the theoretical compute time for eight L4 GPUs would be 14.3 seconds for a 7B parameter model and 167 seconds for a 70B parameter model. However, these theoretical times are more than 450 times less than the actual compute times observed using the Transformers library from

[1] All tests were performed on AWS g6.12xl EC2 Instances equipped with four NVidia L4 GPUs and without batch processing.

HuggingFace. The substantial difference between theoretical and practical processing times is mitigated by the fact that, during inference, the GPUs do not operate at their maximum Thermal Design Power (TDP). Consequently, although the processing time is significantly longer, the actual increase in energy consumption is less pronounced because the GPUs consume less power than their maximum capacity during extended inference periods.

These initial findings suggest that certain inefficiencies or bottlenecks prevent the full utilization of computational resources, leading to discrepancies between theoretical and actual performance. Research has identified several significant bottlenecks during the inference phase of large language models (LLMs). At a high level, these findings indicate that LLM inference is memory-bound, whereas LLM training is compute-bound. Specifically, these bottlenecks include memory bandwidth and I/O overheads, limited memory size and inefficient key-value cache management, control flow overhead, workload imbalance, and pre-filling time and latency. In the following chapter, these bottlenecks will be discussed in detail.

Memory Bandwidth

Memory bandwidth and I/O overheads pose significant challenges to the inference speed of transformer-based models [7], particularly LLMs. For each new token generated, the key values of the entire input sequence and the generated output tokens must be accessed and stored in GPU memory. While this explanation focuses on LLMs for simplification, the general concept also applies to other models using the transformer architecture [7].

The memory access pattern is highly intensive during inference. Initially, the model retrieves embeddings of the input tokens from memory, involving the encoder part of the transformer architecture [8]. These embeddings are processed through the model's layers, where intermediate results are stored and fetched multiple times. For each token, the model must fetch key-value pairs from the current and previous tokens to compute attention scores, crucial for generating contextually relevant output [128]. This read-compute-write cycle is repeated for each token, making the process highly resource-intensive.

The attention mechanism requires accessing all previous key-value pairs stored in the KV cache for every new token. As the sequence length increases, the KV cache size grows, resulting in more data movement between memory and processing units. This increase can significantly consume memory bandwidth and cause potential I/O bottlenecks, especially with long sequences [129]. The decoder processes the embeddings and generates the output tokens, involving further memory operations. Both encoded inputs and generated output tokens are stored in memory to maintain context and facilitate ongoing processing. Each newly generated token continues the cycle of memory access, maintaining context throughout the sequence.

During each inference step, the sequence of memory accesses typically occurs as follows:

1. **Input Embedding Retrieval:** The embeddings for the input tokens are read from memory.
2. **Layer Processing:** As data moves through each layer, intermediate results are continuously written to and read from memory.
3. **Key-Value Fetching:** For each token, the model fetches key-value pairs from previous tokens stored in the KV cache.
4. **Attention Computation:** The model computes attention scores using the fetched key-value pairs, involving multiple read-and-write operations.
5. **Output Generation:** The final output for the current token is computed and written to memory.

These memory operations are repeated for each token, leading to a high frequency of memory accesses. This intensive access pattern can quickly saturate memory bandwidth, causing delays and reducing

inference speed. Efficient memory management and optimization techniques are crucial to address these challenges and improve inference performance [130], [131].

Memory Size

Efficient memory management is essential, especially for the KV cache, which stores key-value pairs from previously processed tokens. Traditional systems pre-allocate memory for the maximum sequence length, often resulting in significant memory waste and inefficiency [129].

In typical transformer models, memory usage is primarily divided among three components: model parameters, the KV cache, and other operations. Approximately 65% of memory is occupied by model parameters, 30% by the KV cache, and the remaining 5% by other operations [128]. This highlights the substantial impact of the KV cache on overall memory usage. Pre-allocating memory for the maximum sequence length can lead to considerable inefficiency. For instance, if a model pre-allocates memory for a sequence length of 4000 tokens but frequently processes sequences of only 200 tokens, it wastes memory allocated for the unused 3800 tokens. This underutilization significantly reduces memory efficiency and can hinder performance [132]. Efficient KV cache management through dynamic memory allocation can mitigate this issue by minimizing memory waste and ensuring that memory is allocated based on actual needs rather than theoretical maximums [133].

When memory becomes insufficient, memory swapping occurs, freeing up memory space by writing data to disk. This significantly slows down computation due to the slower access times of disk storage compared to memory [134]. More available memory allows for more simultaneous processing, accommodating larger and more complex models, more intermediate results, and longer sequences without requiring memory swaps. This parallel processing capability directly translates to improved performance and reduced latency during inference [130].

To illustrate the impact of memory size on performance, consider two scenarios with different memory capacities:

- **Scenario with 50% more memory than required for model parameters:** This scenario provides ample space for the KV cache and other operations, reducing the likelihood of memory swaps and improving overall performance. However, this excess memory might not be fully utilized, leading to potential memory waste.
- **Scenario with only 10% more memory than required for model parameters:** This limited memory scenario increases the risk of memory swaps and bottlenecks, particularly when dealing with longer sequences. The system may frequently run out of memory, leading to increased latency and reduced inference speed.

Efficient memory management, particularly through dynamic allocation, is crucial for optimizing LLM performance. Properly managing the KV cache and ensuring adequate memory size can significantly enhance inference speed and computational efficiency [86]. Additionally, techniques such as efficient workload distribution and the use of CUDA graphs can help utilize available space more effectively (see Chapter 4.2). These methods can optimize parallel processing capabilities and reduce latency further, making inference more efficient and scalable. It is essential to balance memory size appropriately, ensuring neither underutilization nor excessive allocation that could lead to inefficiencies [131].

GPU Cache Size

The size of the GPU cache is another critical factor influencing transformer model performance. The GPU cache provides high-speed data access by temporarily storing frequently accessed data and instructions, significantly reducing latency and improving computational efficiency.

A larger GPU cache reduces the number of memory transfers required, alleviating bandwidth bottlenecks and improving overall performance. This reduction in memory transfer calls helps maintain computational efficiency by allowing the GPU to access data quickly without frequent trips to the main memory. For example, the NVIDIA A100 GPU features a 40 MB L2 cache, a substantial increase from the older V100's 6 MB L2 cache. NVIDIA's latest H100 GPU boasts an even larger 50 MB L2 cache [123], [124]. These increases in cache size enhance data access speeds and computational efficiency.

In addition to L1 and L2 caches, GPUs utilize register files and shared memory. Register files store temporary variables and results during computations, while shared memory is shared among threads in a block, facilitating faster data access within the GPU. These elements contribute to overall GPU processing efficiency by reducing the need to access slower main memory frequently [123], [124]. Technological advancements such as NVLink provide enhanced bandwidth for data transfer between GPUs, particularly beneficial for large-scale computations and deep learning tasks, where efficient data movement is critical. Although NVLink is significant in multi-GPU setups, it will not be discussed in detail in this thesis [135].

Control Flow Overhead, Workload Imbalance, and Prefilling

Additional performance bottlenecks in transformer models include control flow overhead, workload imbalance, and prefilling latency. Efficiently managing these factors is crucial for maximizing the effective utilization of computational resources and enhancing model performance.

Control flow overhead presents a significant bottleneck. During inference, non-computational operations such as branching and looping consume processing time without contributing to floating-point operations per second (FLOPS). These operations reduce the effective utilization of the GPU's computational power. Non-computational operations play a significant role in the lower-than-expected GPU utilization observed in practical scenarios. Optimizing control flow by minimizing unnecessary branches and loops and streamlining code paths can significantly enhance performance [129].

Workload imbalance during inference, particularly with complex models and varied input sizes, can lead to underutilization of some GPU cores, affecting overall efficiency. Balanced workload distribution across GPU cores is necessary to optimize performance. Dynamic load balancing, which distributes workloads based on current processing demands, and task scheduling algorithms that ensure even distribution of tasks are effective strategies to address this imbalance [128].

Prefilling, the process of encoding input tokens before generating the first token, introduces significant latency, especially as the input sequence lengthens. This latency directly impacts the effective throughput of the GPU. Optimizing prefilling processes to reduce latency can improve overall inference performance. Methods such as parallelizing the prefilling process, optimizing memory access patterns during prefilling, and reducing the complexity of initial token encoding can help mitigate this latency [136].

4.2 Optimizing Inference Performance with Serving Engines

Several inference serving engines, such as TensorRT [137], DeepSpeed [138], llama.cpp [139], and vLLM [140], are widely used to enhance the performance of large language models (LLMs). Among these, vLLM stands out due to its advanced features and open-source accessibility. It provides a comprehensive set of tools and techniques to address the numerous bottlenecks that hinder computational efficiency. The complexity of implementing these optimization techniques highlights the value of frameworks and engines like vLLM, which simplify the integration of advanced methodologies.

A detailed analysis by Goel and Borgohain [141] reveals that while TensorRT offers a lightly higher performance in terms of tokens per second, vLLM excels in ease of use and extensive feature sets. Unlike TensorRT, which requires significant implementation time and lacks comprehensive documentation, vLLM balances high performance and user-friendliness, making it a preferred choice among developers

and researchers. Furthermore, vLLM outperforms other alternatives such as HuggingFace Transformers, HuggingFace Text Generation Inference (TGI) [142], DeepSpeed Mii [143], and Triton with a vLLM backend [141]. Its capabilities include tensor parallelism, dynamic memory allocation, efficient scheduling algorithms, paged attention, a custom attention kernel, sophisticated load balancing, zero-copy data transfer, and optimized token encoding. These features collectively enhance computational efficiency, reduce latency, and ensure optimal resource utilization during inference.

Tensor parallelism is a technique used to fit larger models onto multiple GPUs by splitting the model's parameters across these GPUs [144]. Each GPU processes a slice of a tensor, allowing for parallel computation. By distributing the model's weight matrices column-wise across multiple GPUs, tensor parallelism facilitates handling larger models that would exceed the memory capacity of a single GPU. This method accelerates training and inference processes for extremely large models.

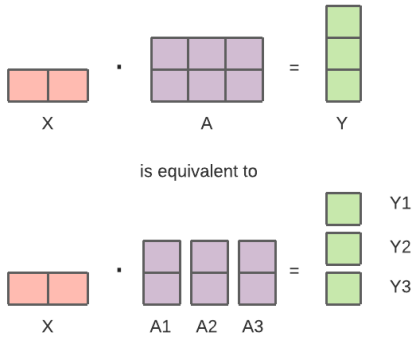

Figure 12: Simplified Illustration of Tensor Parallelism (Source: HuggingFace [145])

In tensor parallelism, the main building block is the fully connected layer followed by a nonlinear activation function, such as GELU. Denoting the input vector as X, the output vector as Y, and the weight matrix as A, the matrix multiplication in the transformer can be written as:

$$Y = \text{GELU}(XA).$$

When splitting the weight matrix A column-wise across N GPUs and performing matrix multiplications XA_1 through XA_N in parallel, N output vectors can be obtained $(Y_1, Y_2, \ldots, Y_N)$ which can be independently fed into the GELU activation function (see Figure 12). The equivalent operations are:

$$X \cdot A = Y \qquad \text{which is split into:} \qquad X \cdot \begin{bmatrix} A_1 & A_2 & A_3 \end{bmatrix} = \begin{bmatrix} Y_1 & Y_2 & Y_3 \end{bmatrix}.$$

Thus, tensor parallelism reduces the memory burden on each individual GPU and leverages parallel computations across multiple GPUs, significantly accelerating the training and inference processes. This allows for the training of very large models that would not fit into the memory of a single GPU.

Pipeline parallelism, on the other hand, divides the model's layers across different GPUs, enabling sequential processing of data through different segments of the model. Each GPU handles specific layers and processes data in stages, passing intermediate results to the next GPU in the pipeline. This technique allows for concurrent processing of different data batches, significantly improving throughput and computational efficiency. For instance, in a transformer model, one GPU might handle the initial layers of the encoder while another handles the subsequent layers, reducing computation time and enhancing scalability.

Studies have shown that using these techniques in combination can lead to substantial performance improvements. Specifically, the use of an interleaved pipelining schedule can improve throughput by over 10% compared to previously proposed schedules while maintaining a comparable memory footprint. This approach enables efficient scaling across thousands of GPUs, achieving significant improvements in training throughput and reducing overall training times [144].

Dynamic memory allocation allows memory resources to be allocated as needed during runtime rather than pre-allocating a fixed amount beforehand. This approach is beneficial when memory requirements vary significantly, such as during inference tasks with varying sequence lengths. Traditional static allocation often leads to substantial memory waste, as memory is allocated based on the maximum possible input tokens, which are rarely fully utilized. Dynamic allocation adjusts memory usage to the model's actual needs in real time, reducing memory waste and improving performance [144].

Paged attention builds upon dynamic memory allocation by managing the KV cache more efficiently. It divides the KV cache into smaller, manageable pages that can be loaded and accessed independently. In traditional attention mechanisms, the entire KV cache must be loaded into memory, which is inefficient for long sequences. Paged attention reduces the memory footprint by keeping only necessary pages in memory at any given time. For instance, dividing the KV cache into 100-token pages allows loading only the required pages for computation, significantly reducing memory usage and improving computational efficiency [144]. Research has shown that paged attention can reduce memory usage of sampling algorithms by up to 55%, resulting in inference throughput improvements of up to 2.2 times [142].

Automatic prefix caching (APC) reduces computational redundancy by caching results of previously computed prefixes. In many natural language processing tasks, input sequences often share common prefixes, leading to redundant calculations. APC stores computed results of these prefixes and reuses them when the same prefix appears again, significantly reducing computational load and speeding up inference, particularly during the prefilling phase [146]. For example, in a chatbot system where each query is preceded by a standard system prompt, APC retrieves cached results for this prompt, eliminating the need to recompute embeddings. APC primarily reduces the time for processing queries but does not impact the time for generating new tokens in the decoding phase. Therefore, its benefits are most pronounced when the length of answers is short and queries share common prefixes.

Zero-copy data transfer enhances data movement efficiency by using a call-by-reference approach. Traditional data transfer methods involve copying data between memory locations, often passing through the CPU, introducing latency and overhead. Zero-copy data transfer avoids this by transferring the reference to the memory address instead of the data itself, allowing direct access to the original data without intermediate copying. This approach minimizes latency and increases data throughput [147]. Technologies like NVIDIA's GPUDirect facilitate zero-copy transfer by enabling direct memory access between GPUs, reducing latency significantly [148].

NVLink is a high-bandwidth interconnect technology developed by NVIDIA for faster data transfer between GPUs. Unlike traditional PCIe connections with limited bandwidth, NVLink provides a direct and high-speed connection. The latest generation supports transfer rates over 600 GB/s, significantly outperforming PCIe Gen5's 128 GB/s [149]. This substantial bandwidth reduces communication overhead and latency in multi-GPU setups, beneficial for data-intensive applications like LLM inference [135].

Efficient scheduling algorithms optimize the execution order of operations within a computational system. These algorithms dynamically adjust the sequence of operations based on current computational demands and resource availability. In LLM inference, efficient scheduling ensures that GPUs are continuously engaged, minimizing idle times and maximizing resource utilization. By analyzing workloads and rearranging tasks dynamically, these algorithms prevent bottlenecks and maintain consistent performance, leading to higher throughput [144].

Load balancing algorithms distribute computational tasks evenly across available GPU cores. In complex models with varied input sizes and computational requirements, preventing any single GPU core from becoming a bottleneck is crucial. Load balancing algorithms adjust the distribution of tasks based on real-time analysis of workload and resource availability. This even distribution ensures effective utilization of all GPU cores, leading to smoother and faster processing [144].

Cuda Graphs

Cuda Graphs are a technique developed by NVidia to address inefficiencies in GPU operations. Traditional execution involves significant overhead due to repeated scheduling and launching of individual kernels. CUDA Graphs reduce runtime overhead by replaying the captured graph multiple times with minimal overhead, as operations do not need to be individually scheduled each time [150], [151].

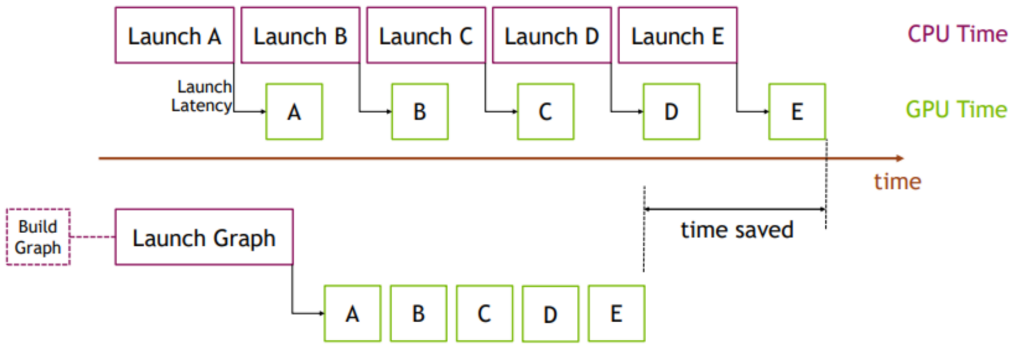

Figure 13: Simplified Illustration of CUDA Graphs (Source: Nguyen et. al. [150])

A key advantage of CUDA Graphs is their ability to optimize memory usage and execution order within the captured graph. Having a complete view of operations and dependencies allows intelligent decisions about memory allocation and reuse, reducing fragmentation and maximizing efficiency. Additionally, CUDA Graphs can fuse operations that would otherwise execute separately, further reducing kernel launches and improving performance [150], [151]. This approach enhances determinism and predictability of GPU workloads, ensuring consistent and repeatable execution, crucial for debugging and performance tuning.

In summary, the optimization techniques implemented in vLLM address the various bottlenecks identified in LLM inference (as discussed in the previous section), enhancing both execution performance and energy efficiency. By incorporating advanced methodologies such as tensor and pipeline parallelism, dynamic memory allocation, paged attention, automatic prefix caching, zero-copy data transfer, efficient scheduling and load balancing algorithms, and CUDA Graphs, vLLM provides a robust framework for efficient LLM inference. These optimizations collectively contribute to reduced latency, improved throughput, and optimal resource utilization, making the deployment of large language models more practical and sustainable.

5 RESULTS

This section presents the findings from performance and efficiency experiments on large language models. The focus is on serving engine optimization, model size, and token lengths, and their impact on output quality and energy consumption. The tests were conducted on AWS `g6` instances with L4 GPUs.

First, the vLLM serving engine was compared to the traditional Transformers library to evaluate improvements in prompt processing speed and energy efficiency. Next, different model sizes, ranging from 7B to 70B parameters in the Code LLaMA family, were assessed to determine their effect on computational efficiency. Then, the effects of varying input and output token lengths on energy usage were examined, noting nonlinear trends at higher token counts. Finally, quantization and prompt engineering, including Retrieval-Augmented Generation (RAG), were explored to optimize efficiency while maintaining or enhancing output quality.

5.1 Effects of an optimized Serving Engine

This subsection presents the results of performance and energy consumption tests comparing the optimized vLLM engine with the traditional Transformers library implementation. The tests were conducted on an AWS g6 12xl instance equipped with four L4 GPUs. A total of 7500 prompts were batch processed through an array[2]. The aim was to assess the efficiency and performance gains offered by the vLLM engine in comparison to the widely used Transformers library. For details of the testing methodology, refer to Section 3.

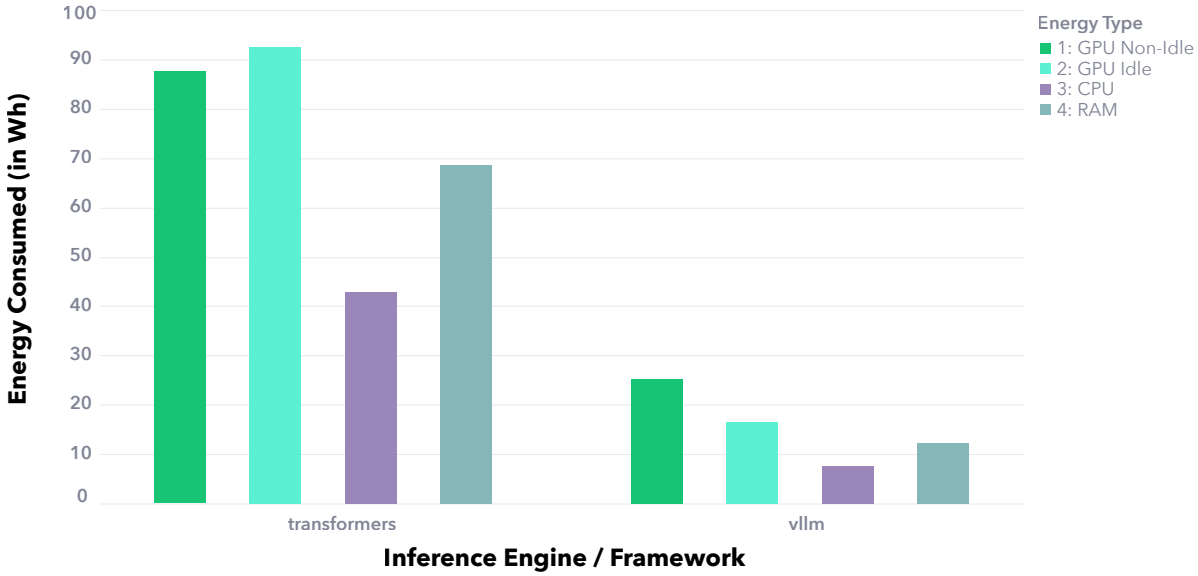

Figure 14: Comparing the Energy Consumption during batch Inference of LLaMA 3 8B Instruct using the Transformers Library and the vLLM Serving Engine

Figure 14 illustrates that vLLM offers a substantial reduction in energy consumption compared to Transformers. Table 4 provides detailed test results, including metrics such as average prompts processed per second, total time taken for inference, and energy consumption across different components.

[2]Detailed code for these tests can be found in the accompanying GitHub repository, specifically in the notebook: `notebooks/vllm_vs_transformers.ipynb`.

Metric	Transformers	vLLM
AVG. Prompts/s	2.52	14.16
Total Time	49min 33s	8min 50s
Energy Consumption: GPU (non-idle)	87.53 Wh	25.18 Wh
Energy Consumption: GPU (idle)	92.49 Wh	16.48 Wh
Energy Consumption: CPU	42.82 Wh	7.63 Wh
Energy Consumption: RAM	68.61 Wh	12.22 Wh
Total Energy Consumption	**291,45 Wh**	**61.51 Wh**

Table 4: Performance and Energy Consumption for 7500 Prompts using Transformers and vLLM

The tests revealed that the vLLM engine significantly reduces energy consumption and dramatically improves performance in terms of processing speed. The vLLM processed an average of 14.16 prompts per second, compared to just 2.52 prompts per second with the Transformers implementation. This improvement is also reflected in the total time required for inference. Using vLLM, the time to process 7500 prompts was reduced to 8 minutes and 50 seconds, a substantial decrease from the 49 minutes and 33 seconds required by the Transformers library.

Energy consumption metrics further highlight the advantages of vLLM. The total energy consumption for the Transformers setup was 291.45 Wh, whereas vLLM consumed only 61.51 Wh. This represents a 4.7x reduction in total energy consumption when using vLLM. Specifically, the energy consumption of non-idle GPUs was reduced from 87.53 Wh with Transformers to 25.18 Wh with vLLM, while idle GPU consumption dropped from 92.49 Wh to 16.48 Wh. Similarly, CPU and RAM energy consumption were significantly reduced, with CPU energy usage decreasing from 42.82 Wh to 7.63 Wh and RAM usage dropping from 68.61 Wh to 12.22 Wh.

Interestingly, the reduction in idle GPU, CPU, and RAM energy consumption is slightly more than 5.5x, whereas the reduction for non-idle GPU consumption is slightly less than 3.5x. This indicates that vLLM effectively reduces bottlenecks and utilizes the GPU more efficiently, leading to higher active energy consumption during inference and lower idle consumption.

While the Transformers library remains a well-established standard for large language models, these results demonstrate that adopting optimized engines like vLLM can lead to considerable performance and energy efficiency gains.

5.2 Model Size (Parameters)

The impact of model size on inference performance and energy consumption was evaluated by testing different parameter sizes within the Code LLaMA model class from Meta. Models ranging from 7 billion (7B) to 70 billion (70B) parameters were selected to explore how an increase in model size affects computational demands during batched inference.

The inference tests were conducted using 7500 batched queries on various AWS g6 instance types. The largest instance used was the g6.48xl, equipped with eight Nvidia L4 GPUs and a total of 192 GB VRAM. This configuration was just large enough to house the 70B parameter model in full 16-bit precision.

As shown in Figure 15, energy consumption decreased significantly as the model size was reduced, even when using the same instance size. For example, as presented in Table 5, the energy consumption for the 34B model on the g6.48xl instance was 678.15 Wh, nearly half of the 1308.01 Wh consumed by the 70B model on the same instance type. This significant reduction highlights the efficiency gains achievable by scaling down the model size while maintaining the same hardware configuration.

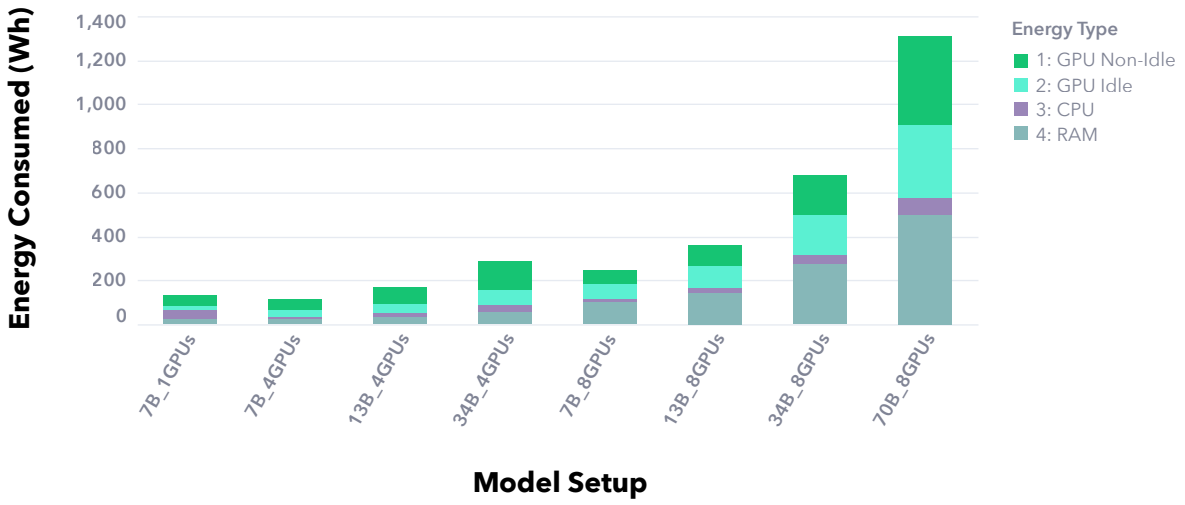

Figure 15: Comparing the Energy Consumption during batch Inference of different Code LLaMA Model Sizes

Model Size	Num. GPUs	Total Time	Energy Cons.
70B	8 GPUs	89 min	1308.01 Wh
34B	8 GPUs	49 min	678.15 Wh
34B	4 GPUs	38 min	285.53 Wh
13B	8 GPUs	26 min	359.18 Wh
13B	4 GPUs	22 min	166.88 Wh
7B	8 GPUs	18 min	246.78 Wh
7B	4 GPUs	16 min	113.76 Wh
7B	1 GPU	47 min	131.58 Wh

Table 5: Energy Consumption during the batched Inference Test with vLLM for different Code LLaMA Model Sizes[3]

Notable results were observed when the instance size was also adjusted to better align with the model's computational needs. Running the 34B model on a 4-GPU setup instead of the 8-GPU setup reduced energy consumption even further, from 678.15 Wh to 285.53 Wh. Moreover, when the model and its associated overhead fit within a certain instance configuration, scaling up the number of GPUs can introduce additional bottlenecks rather than advantages. To fully utilize all GPUs, the model weights are split across the GPUs in a tensor parallel manner, necessitating data transfers between the GPUs. Although Nvidia's NVLink allows for fast data transfers, these transfers still occur and can contribute to increased compute time. This explains the slight increase in processing time observed when using the 8-GPU setup compared to the 4-GPU setup. Interestingly, the relationship between model size, instance size, and energy consumption is not always linear. Testing the 7B model on a smaller setup with only one GPU resulted in an increase in energy consumption to 131.58 Wh, compared to 113.76 Wh on the 4-GPU setup. Although the increase in energy consumption was relatively modest, the time required to process the 7500 prompts increased drastically, from 16 minutes on the 4-GPU setup to 47 minutes on the

[3]The tests were done with 7500 Prompts each with the task to generate three short jokes.

1-GPU setup. These results highlight the importance of matching the model size with the appropriate instance size for optimal efficiency.

The findings indicate that an inference setup requires some overhead beyond the memory requirement for the model weights to maintain efficiency. According to the vLLM paper [128], approximately 60% of the memory used is occupied by model weights, with an additional 30% required for KV caching. While memory optimization can reduce the footprint of the cache, performance still benefits from sufficient memory to house the cache, especially in batched inference scenarios where the cache can be shared across multiple requests. Additionally, the use of CUDA graphs, which allocate 1–3 GB of GPU memory per GPU, further emphasizes the need for adequate memory beyond just the model weights.

For instance, the memory footprint of the model weights can be calculated using the following formula:

$$\text{Memory Footprint (GB)} = \frac{\text{Parameter Size} \times \text{Precision (bits)}}{8 \times 10^9}$$

For a 7 billion parameter model with 16-bit precision, this calculation yields:

$$\text{Memory Footprint} = \frac{7 \times 10^9 \times 16}{8 \times 10^9} = 14 \text{ GB}$$

Considering the need for a 30-40% overhead in memory, plus an additional 1-3 GB reserved for CUDA graphs, the required memory would be:

$$\text{Total Required Memory} = 14 \text{ GB} \times 1.4 + 3 \text{ GB} = 22.6 \text{ GB}$$

Despite the Nvidia L4 GPU having a theoretical memory size of 24 GB, the practical limit is around 21–22 GB, meaning that the 7B model's requirements slightly exceed this capacity. Additionally, it is assumed that the share of overhead needed for smaller models is more significant compared to larger models, given that the fixed overheads, such as CUDA graphs and memory allocation for caching, do not scale linearly with model size. Therefore, the optimal number of L4 GPUs for the inference of a 7B parameter model would be two. However, there was no AWS instance with a configuration of two L4 GPUs available for testing.

In conclusion, while reducing model size and adjusting instance size can lead to significant energy savings, careful consideration must be given to the memory requirements and processing time to optimize both energy consumption and inference performance.

5.3 Output Token Length

The impact of output token length on energy consumption during inference was examined by conducting experiments with varying numbers of generated output tokens. Table 6 presents the empirical data obtained from the experiments, showing the average number of output tokens per prompt and the corresponding total energy consumption for processing 10,000 prompts.

Analyzing the data reveals that energy consumption increases with the number of output tokens generated per prompt. Initially, the relationship between output token length and energy consumption appears nearly linear for shorter sequences. For example, increasing the average output tokens from approximately 18 to 70 results in a moderate increase in energy consumption from 52.59 Wh to 87.72 Wh. However, as the number of output tokens increases further, the relationship becomes more pronounced, with energy consumption rising significantly at higher token counts (see Figure 16).

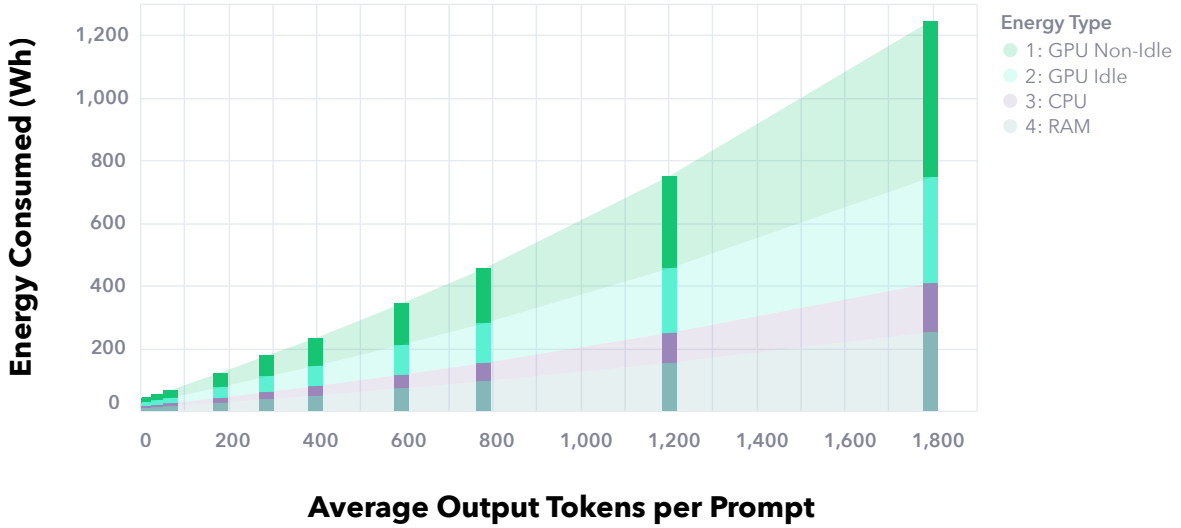

Figure 16: Comparing the Energy Consumption during batch Inference with the Number of generated Output Tokens utilizing an added Quadratic Regression

Average Output Tokens	**Energy Consumption**
18.17	52.59 Wh
40.86	70.99 Wh
69.86	87.72 Wh
182.24	161.19 Wh
398.26	308.04 Wh
594.77	456.19 Wh
1202.48	998.43 Wh
1796.30	1659.39 Wh

Table 6: Energy Consumption for Different Output Token Lengths in Batch Inference of 10,000 Prompts

This behavior arises from two factors. First, the computational effort required to calculate attention scores discussed earlier in this thesis (see section 2.6.1) becomes more significant. Casson broke down how the distribution of linear and quadratic terms within the total number of calculations of the transformer architecture changes with increasing context sizes [121]. Based on these findings, the share of quadratic terms becomes significant for context lengths of over 10.000 tokens. This aligns well with our findings.

Yet, another factor we have identified is also very likely contributing to this behavior: The autoregressive nature of LLMs during text generation.

Each new output token is generated based on the entire sequence of preceding tokens, which includes both the initial input tokens and the tokens generated so far. Consequently, with each additional output token, the model's context length increases by one token. The computational complexity for generating each token is proportional to the context length, as the model must process the entire sequence to predict the next token.

For shorter output sequences, the incremental increase in context length is relatively small, resulting in an approximately linear increase in computational effort and energy consumption. However, as the number

of output tokens grows, the cumulative effect of the increasing context length becomes significant. The energy consumption data reflects this trend, where increasing the average output tokens from around 1202 to 1796 leads to a substantial rise in energy consumption from 998.43 Wh to 1659.39 Wh.

These findings have practical implications for deploying large language models in energy-constrained environments. When designing systems that generate long outputs, it is essential to account for the disproportionately higher energy consumption associated with longer output sequences. Optimizing output lengths and employing techniques to reduce computational overhead, such as efficient caching mechanisms or guided output templates, can mitigate energy costs.

Furthermore, the results emphasize the importance of carefully selecting the desired output length based on the specific application requirements. In scenarios where shorter responses are sufficient, limiting the output token length can lead to significant energy savings without compromising the quality of the results.

In conclusion, the experiments demonstrate that while short output sequences result in a relatively modest increase in energy consumption, longer sequences lead to a much steeper increase due to the quadratic relationship between output token length and computational complexity. This behavior is inherent to the autoregressive generation process of large language models and highlights the need to consider output token length in energy optimization strategies.

5.4 Input Token Length

The influence of input token length on energy consumption during inference was investigated by conducting experiments with varying lengths of input text. The experiments utilized the LLaMA 3.1 8B model to summarize sequences from the text of *Moby-Dick* by Herman Melville. To isolate the impact of input token length, the energy consumption was standardized to 100,000 output tokens across all tests. Furthermore, the model was queried to summarize the same text sequence 1000 times.

Table 7 presents the empirical data obtained from the experiments, showing the average number of input tokens per prompt and the corresponding total energy consumption required to generate 100,000 output tokens.

Analyzing the data reveals that, in general, energy consumption increases with the average number of input tokens per prompt. With some exceptions a linear regression still yields the best fit. This linear relationship arises because each output token generation requires the model to process the entire input sequence. Therefore, if the input token length increases, the computational effort for each output token increases proportionally.

Average Input Tokens	**Energy Consumption**
441	8.76 Wh
748	7.01 Wh
1282	8.11 Wh
2220	10.22 Wh
3885	13.79 Wh
7759	26.63 Wh
14922	43.70 Wh
25457	57.46 Wh

Table 7: Energy Consumption for Different Input Token Lengths in Generating 100,000 Output Tokens

In the experiments discussed in Section 5.5, an increase of the input token length by a factor of approximately 6.4 led to an increase in energy consumption by a factor between 1.4 and 1.7, when using sufficiently large hardware instances. This indicates that, under ideal hardware conditions, the relationship between input token length and energy consumption is approximately linear.

However, additional overheads can occur if the available GPU memory is insufficient to handle the increased input length. The larger input requires more memory for storing intermediate computations. If the memory capacity is exceeded, the system may need to reduce parallel processing or perform memory swaps, leading to inefficiencies and greater increases in energy consumption. This was evident in the case of the 1-GPU fp8 test for the Mistral Nemo model, as shown in Table 11.

In summary, while energy consumption increases linearly with input token length under optimal conditions, hardware limitations can cause deviations from this relationship due to reduced efficiency.

5.5 RAG & Prompt Engineering

This experiment aimed to evaluate how the integration of Retrieval-Augmented Generation (RAG) and advanced prompt engineering could improve the performance of various LLMs on the arena-hard-auto benchmark, which consists of 500 particularly challenging questions. To automate the process of generating supplemental guidance, a complementary script was developed for the arena-hard-auto framework.[4] This script prompts a guide model—specifically GPT-4o—to produce detailed guidance for each benchmark question.[5] The guidance includes background information, step-by-step plans, useful tips, and additional considerations.

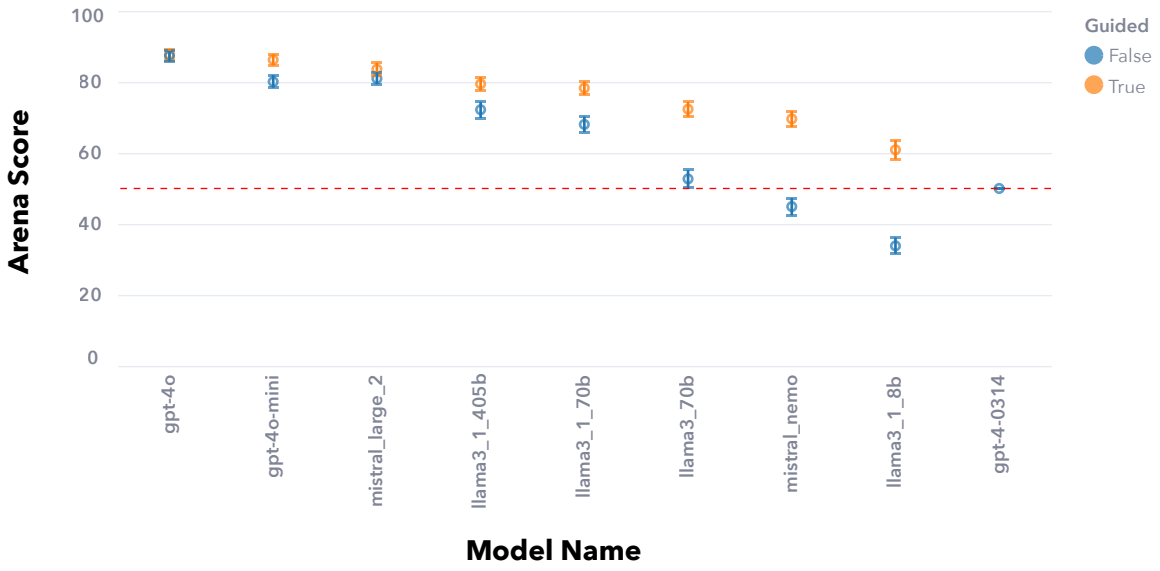

Figure 17: Model Quality (Arena Score) with and without the enriched System Prompt

[4]The code for automating the guidance generation process can be found in the public GitHub repository associated with this work [2], specifically within the Fork of the `arena-hard-auto` repository in the `/llm_judge` folder.

[5]The specific prompt template used to generate these guidances is available in Appendix B.

The guidance was designed to achieve the following goals:

1. Provide relevant background information necessary for solving the task.
2. Offer practical tips to approach the task effectively.
3. Present a step-by-step plan to systematically address the task.
4. Encourage deeper reasoning throughout the task-solving process.
5. Provide additional considerations that could influence the solution.

A critical aspect of this setup involved a significant increase in the number of input tokens to accommodate the enriched context. The average input tokens per prompt increased from 135.142 to 874.592 for the LLaMA 3 and 3.1 models, and from 143.19 to 921.216 for the Mistral models.

Embedding additional context and guidance in the system prompt resulted in significant performance gains, particularly for models with lower baseline capabilities. Figure 17 illustrates the relationship between model quality (as measured by Arena Scores) and the use of enriched prompts. Models with inherently lower performance, such as LLaMA 3.1 8B, showed the most significant improvements, with an increase of over 27 Arena Points. In contrast, models closer in performance to the guide model (GPT-4o) exhibited more modest gains.

Model Name	**Non-Guided Arena Score (CI)**	**Guided Arena Score (CI)**	**Arena Score Improvement**
GPT-4o	87.32 (-1.63, +1.41)	87.75 (-1.64, +1.53)	+0.43
GPT-4o-mini	80.09 (-1.68, +1.86)	86.27 (-1.58, +1.56)	+6.18
Mistral Large 2407	80.96 (-1.60, +1.84)	83.63 (-1.75, +1.85)	+2.67
LLaMA 3.1 405B	72.20 (-2.35, +2.20)	79.47 (-1.88, +1.93)	+7.27
LLaMA 3.1 70B	68.05 (-2.15, +2.23)	78.29 (-1.92, +1.87)	+10.24
LLaMA 3 70B	52.70 (-2.54, +2.51)	72.34 (-2.13, +2.08)	+19.64
GPT-4-0314	**50.00 (-0.00, +0.00)**	**N/A**	**N/A**
Mistral Nemo 2407	44.90 (-2.45, +2.38)	69.60 (-2.27, +2.21)	+24.70
LLaMA 3.1 8B	33.83 (-2.20, +2.34)	60.90 (-2.68, +2.66)	+27.07

Table 8: Arena Hard Auto Scores with and without System Prompt Guidance. GPT-4o was used to automatically create the guidance and judge the models. GPT-4-0314 serves as the baseline with a score of 50 points.[6]

Table 8 provides a detailed breakdown of Arena Scores for models with and without the enriched system prompt. The greatest improvements were observed in models with lower output quality, while the strongest model, GPT-4o, saw no increase beyond the confidence intervals.

While the enriched guidance led to performance improvements, it also resulted in a notable increase in energy consumption across all tested models. This increase ranged from 1.4 to 1.7 times the original consumption, depending on the model and the precision level used. Figure 18 highlights the trade-off between model performance and energy consumption for two representative models: LLaMA 3.1 70B Instruct and Mistral Nemo 2407 Instruct.

[6]All models used their Instruction-Tuned Versions if available.

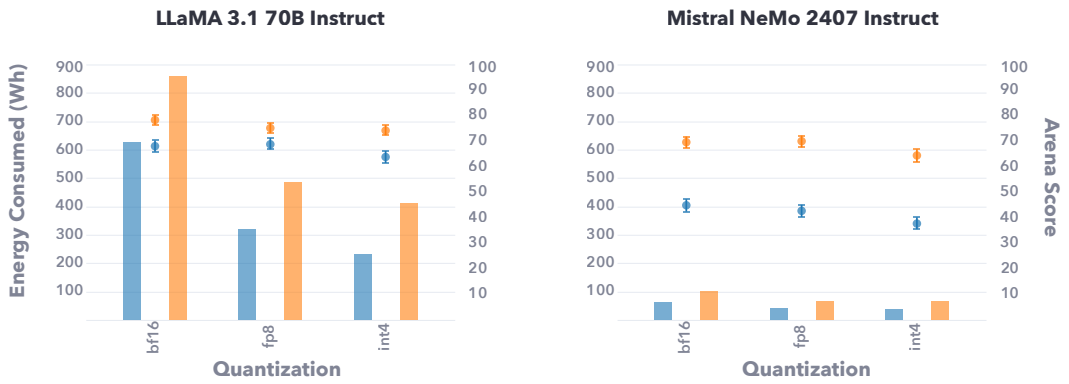

Figure 18: Model Quality (Arena Score) with and without the enriched System Prompt vs. Energy Consumption for LLaMA 3.1 70B Instruct and Mistral Nemo 2407 Instruct

Model Name	**Precision**	**Non-Guided Energy Cons.**	**Guided Energy Cons.**	**Increase**
LLaMA 3.1 70B	bf16	624,59 Wh	859.25 Wh	1.38×
LLaMA 3.1 70B	fp8	318.25 Wh	483.28 Wh	1.52×
LLaMA 3.1 70B	int4	232.03 Wh	411.30 Wh	1.77×
Mistral Nemo 2407	bf16	61.23 Wh	99.70 Wh	1.63×
Mistral Nemo 2407	fp8	39.19 Wh	65.14 Wh	1.66×
Mistral Nemo 2407	int4	38.42 Wh	64.03 Wh	1.67×
LLaMA 3.1 8B	bf16	62.13 Wh	88.07 Wh	1.42×
LLaMA 3.1 8B	fp8	38.49 Wh	58.48 Wh	1.52×
LLaMA 3.1 8B	int4	36.93 Wh	54.16 Wh	1.47×

Table 9: Energy Consumption during the Arena Hard Auto Benchmark with and without System Prompt Guidance, categorized by quantization precision.[7]

Table 9 details the energy consumption for models using different quantization precisions, emphasizing the balance between improved performance and increased resource usage. This experiment highlights the balance between model performance and energy efficiency, demonstrating that while enriched prompts can significantly enhance performance, they also come with increased energy costs.

5.6 Quantization

Reducing the precision of model weights from 16-bit floating point (bf16) to lower precision formats such as fp8 and int4 offers significant potential for improving the efficiency of model inference. As discussed in the theoretical section on quantization (Chapter 2.5.4), quantization can retain over 99% of model quality in standard multiple-choice benchmarks [33], [35]. This makes it an appealing approach for reducing computational demands without substantially compromising accuracy. However, these benchmarks primarily focus on structured tasks and may not fully reflect the complexities of real-world applications. Therefore, the Arena Hard Auto Benchmark was utilized to assess how quantization impacts model performance in more open-ended and practical scenarios.

[7]The results are shown for the largest instance configuration

This experiment tested the effects of quantization on both model performance and energy consumption, focusing on the 70B and 8B parameter versions of LLaMA 3.1 and the 12B parameter Mistral Nemo model. These models allowed for the investigation of how reducing precision, particularly from bf16 to fp8 and int4, affects both operational efficiency and model accuracy in realistic settings. Specifically, the AWQ quantization technique [33] was used for the 4-bit quantization, and the FP8 dynamic quantization technique [35] was used for the 8-bit quantization. These techniques were chosen because they provided the best support with vLLM at the time of the practical benchmarks.

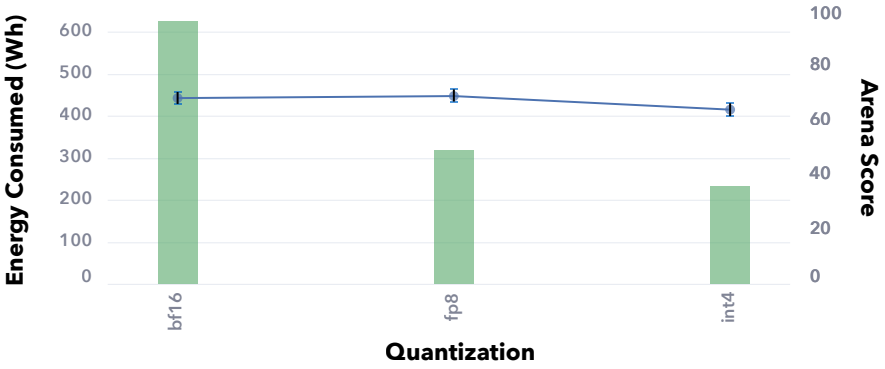

Figure 19: Model Quality (Arena Score) and Energy Consumption throughout different quantization levels for LLaMA 3.1 70B Instruct

Model Name	**Precision**	**Non-Guided Arena Score (CI)**	**Guided Arena Score (CI)**
LLaMA 3.1 70B	bf16	68.05 (-2.15, +2.23)	78.29 (-1.92, +1.87)
LLaMA 3.1 70B	fp8	68.79 (-2.00, +2.46)	75.14 (-2.02, +2.02)
LLaMA 3.1 70B	int4	63.82 (-2.36, +2.34)	74.17 (-1.85, +2.14)
GPT-4-0314	**-**	**50.00 (-0.00, +0.00)**	**N/A**
Mistral Nemo 2407	bf16	44.90 (-2.45, +2.38)	69.60 (-2.27, +2.21)
Mistral Nemo 2407	fp8	42.73 (-2.38, +2.36)	70.01 (-2.43, +2.18)
Mistral Nemo 2407	int4	37.74 (-2.18, +2.45)	64.41 (-2.53, +2.42)
LLaMA 3.1 8B	bf16	33.83 (-2.20, +2.34)	60.90 (-2.68, +2.66)
LLaMA 3.1 8B	fp8	33.65 (-2.41, +2.23)	54.02 (-2.43, +2.80)
LLaMA 3.1 8B	int4	29.40 (-2.07, +2.08)	52.27 (-2.98, +2.61)

Table 10: Arena Scores with and without System Prompt Guidance, categorized by Precision.

As seen in Table 10, quantization generally resulted in some loss of model quality when assessed using the Arena Hard Auto Benchmark. However, the retention of quality, especially when using fp8 precision, remained sufficiently high to justify the reduction in precision for most applications. For instance, the LLaMA 3.1 70B model's Arena Score did not decrease significantly when quantizing from bf16 to fp8.

The primary advantage of quantization, as shown in Table 11, was the considerable reduction in energy consumption. For instance, the LLaMA 3.1 70B model running in fp8 on an 8-GPU setup consumed 318.25 Wh, almost half of the 624.59 Wh consumed when using bf16 precision. When the model was further quantized to int4 and run on a 4-GPU setup, energy consumption dropped to 199.09 Wh. Similarly, Mistral Nemo 2407, when quantized from bf16 to int4 and moved from a 4-GPU to a 1-GPU setup, reduced energy consumption from 61.23 Wh to 24.31 Wh.

Model Name	Precision	Num. GPUs	Non-Guided Energy Cons.	Guided Energy Cons.
LLaMA 3.1 70B	8 GPUs	bf16	624,59 Wh	859.25 Wh
LLaMA 3.1 70B	8 GPUs	fp8	318.25 Wh	483.28 Wh
LLaMA 3.1 70B	8 GPUs	int4	232.03 Wh	411.30 Wh
LLaMA 3.1 70B	4 GPUs	fp8	250.20 Wh	424.81 Wh
LLaMA 3.1 70B	4 GPUs	int4	199.09 Wh	266.35 Wh
Mistral Nemo 2407	4 GPUs	bf16	61.23 Wh	99.70 Wh
Mistral Nemo 2407	4 GPUs	fp8	39.19 Wh	65.14 Wh
Mistral Nemo 2407	4 GPUs	int4	38.42 Wh	64.03 Wh
Mistral Nemo 2407	1 GPU	fp8	30.73 Wh	61.64 Wh
Mistral Nemo 2407	1 GPU	int4	24.31 Wh	42.67 Wh
LLaMA 3.1 8B	4 GPUs	bf16	62.13 Wh	88.07 Wh
LLaMA 3.1 8B	4 GPUs	fp8	38.49 Wh	58.48 Wh
LLaMA 3.1 8B	4 GPUs	int4	36.93 Wh	54.16 Wh
LLaMA 3.1 8B	1 GPU	fp8	26.07 Wh	33.96 Wh
LLaMA 3.1 8B	1 GPU	int4	22.47 Wh	34.14 Wh

Table 11: Energy Consumption during the Arena Hard Auto Benchmark with and without System Prompt Guidance, categorized by quantization precision and number of GPUs.

While the drop in model quality was more pronounced when evaluated with the Arena Hard Auto Benchmark, the overall quality retention, especially for fp8 quantization, remained high enough to make this trade-off worthwhile. Comparing the minimal degradation in quality with decreases in energy consumption by factors of 2x to 3x, it becomes evident that quantization is an effective solution to significantly increase the efficiency of model inference.

5.7 Considerations on Model Generation

Advancements in model generations offer a promising strategy to improve model quality without significantly increasing energy consumption. Historically, transitions between model generations have demonstrated substantial quality gains without corresponding increases in computational resource usage. This trend is evident in the shift from LLaMA-3 to LLaMA-3.1, where quality improvements occur while energy consumption remains relatively unchanged. Equivalent models from both generations maintain the same parameter size, making their energy consumption relatively comparable. However, LLaMA-3.1 exhibits a significant increase in output quality on the Arena Hard Auto Benchmark. As shown in Table 8, the LLaMA 3.1 70B Instruct model achieved a non-guided Arena Score of 68.05, whereas its predecessor, LLaMA 3 70B Instruct, reached only 52.70. The improvement of 15.35 points demonstrates the potential of model advancements to enhance quality with minimal energy impact.

This example underscores the importance of building dynamic and interchangeable Gen AI platforms. Platforms designed to easily swap foundation models enable organizations to utilize the latest advancements without significant delays. Maintaining flexibility in platform design allows dynamic switching to newer, more efficient models upon release. This adaptability should extend beyond models from the same provider—for instance, transitioning between LLaMA generations—to models from different families and providers. For example, if a new model from Mistral demonstrates superior quality over current LLaMA models, platforms should be equipped to transition seamlessly to this model. A well-architected, interchangeable platform facilitates rapid integration of superior models from different providers, ensuring continued state-of-the-art performance without significant changes to the underlying system.

6 DECISION FRAMEWORK

Implementing large language models (LLMs) efficiently and sustainably necessitates a systematic approach that balances performance, energy consumption, and cost. Building upon the empirical findings from previous experiments, this section introduces a comprehensive decision framework to guide practitioners in optimizing LLM deployments. The discussion begins by outlining the key factors—both customization and implementation—that significantly impact energy efficiency and model quality. A step-by-step methodology is then provided for informed decision-making, covering aspects such as hosting strategies, hardware selection, and model customization. Additionally, an interactive tool is presented to assist in calculating potential energy savings, enabling organizations to tailor LLM implementations to specific requirements and constraints.

6.1 Key Findings

The conducted experiments revealed several crucial factors influencing the energy efficiency and performance of LLMs. These findings are categorized into customization factors and implementation factors, each playing a significant role in optimizing LLM deployments. The customization factors pertain to model-specific adjustments that can enhance efficiency, while implementation factors involve strategic decisions about hosting and resource utilization. Understanding these factors is essential for organizations aiming to balance performance with sustainability in their AI solutions.

6.1.1 Customization Factors

The experimental results highlight several key customization factors that substantially influence the energy efficiency and performance of LLMs. These factors are critical considerations in the decision-making process for implementing and deploying LLMs in practical applications.

Firstly, model size, defined by the number of parameters, directly impacts energy consumption and computational demands. Experiments with Code LLaMA models ranging from 7B to 70B parameters revealed that reducing model size significantly decreases energy consumption. For instance, downsizing from a 70B to a 34B parameter model resulted in nearly a 50% reduction in energy usage when using the same hardware configuration. Aligning the hardware setup with the model's computational and memory requirements minimizes unnecessary overhead and maximizes performance. However, over-provisioning hardware can introduce inefficiencies due to increased data transfer between GPUs and underutilized resources.

Secondly, the length of input and output tokens affects energy consumption during inference. While the impact of increased input token length is relatively modest, longer output sequences lead to a nonlinear rise in energy usage. This nonlinear relationship intensifies as output token lengths extend into the thousands, owing to the sequential and recursive nature of token generation in LLMs. Understanding the application's token length requirements allows for better estimation of energy costs and informs decisions on model configuration and resource allocation.

Thirdly, the incorporation of advanced prompt engineering techniques and Retrieval-Augmented Generation (RAG) can enhance model performance but may also increase energy consumption. Enriching the system prompt with additional guidance significantly improves model outputs, particularly for models with lower baseline performance. However, this improvement comes with a 1.4 to 1.7-fold increase in energy consumption, primarily due to the increased input token length required to provide the additional context. Decision-makers must balance the benefits of improved performance against the higher energy costs associated with these techniques.

Fourthly, quantization emerges as an effective strategy to reduce energy consumption without substantial loss of model quality. Reducing precision from 16-bit to lower precisions such as 8-bit (fp8) or 4-bit (int4)

significantly decreases energy usage and hardware requirements. For example, quantizing the LLaMA 3.1 70B model to fp8 reduced energy consumption by nearly 50% with minimal impact on performance in practical benchmarks. Quantization techniques enable the deployment of large models on smaller hardware configurations, enhancing efficiency and reducing operational costs.

Lastly, advancements in model generation can yield significant improvements in performance without increasing energy consumption. The transition from LLaMA 3 to LLaMA 3.1 resulted in substantial gains in model quality while maintaining similar energy requirements. Keeping models up-to-date with the latest versions can enhance performance efficiency. Building flexible and dynamic platforms that allow for seamless integration of new models can capitalize on these advancements without necessitating significant infrastructure changes.

In summary, the identified customization factors—model size selection, token length management, advanced prompt engineering and RAG, quantization, and model updates—are critical considerations for optimizing the deployment of LLMs. By carefully evaluating and adjusting these factors, organizations can achieve a balance between performance and energy efficiency, leading to more sustainable and cost-effective AI solutions.

6.1.2 Implementation Factors

Implementation choices significantly influence the energy efficiency and performance of LLMs. Key factors include the hosting strategy, the availability of specialized hardware, the selection of hosting regions, and the choice of serving engine. These factors impact computational efficiency, resource utilization, and environmental sustainability.

Firstly, the hosting strategy plays a pivotal role in energy consumption. Deploying and self-hosting a model may lead to considerable energy waste if the model is underutilized. LLMs require substantial computational resources, and if the inference demand does not fully occupy these resources, the energy consumed during idle periods contributes to inefficiency. Self-hosting is advisable from an efficiency standpoint only when the model is fully utilized during its uptime. This scenario occurs when large batches of queries are processed frequently or when there is a sufficiently large user base to keep the model continuously engaged.

Alternatively, utilizing models through application programming interfaces (APIs) provided by external services can be more energy-efficient. API-based models operate on shared infrastructure, where computational resources are pooled across multiple customers. This shared usage leads to higher hardware utilization rates and reduces the per-request energy consumption. Additionally, costs are typically incurred on a per-request basis, aligning expenses directly with usage and avoiding the overhead of maintaining dedicated hardware.

However, using external APIs introduces limitations, such as reduced transparency regarding the model's hosting environment. Information about the specific data center location, regional energy mix, and sustainability practices may not be readily available. This lack of transparency can hinder efforts to assess and minimize the carbon footprint associated with model usage.

Conversely, self-hosting provides the opportunity to select hosting regions with low-carbon energy mixes to reduce the environmental impact of LLM deployments. Regions such as Northern Europe (e.g., Norway and Sweden), Switzerland, Iceland, West and East Canada, Brazil, and New Zealand rely heavily on renewable energy sources like hydropower and geothermal power, leading to some of the lowest carbon intensities globally [55]. Hosting models in these regions can significantly decrease the carbon footprint associated with energy consumption. For example, Iceland's energy grid is almost exclusively powered by geothermal and hydropower, making it one of the most sustainable options for data center operations.

However, this choice requires careful planning and may involve considerations regarding data sovereignty, latency, and regulatory compliance.

Secondly, the availability of specialized hardware for model inference impacts the decision to self-host or use external services. Recent investigations into the inference speed of different API providers have identified three leading providers: SambaNova [152], Groq [152], and Cerebras [153]. These providers offer significantly faster model APIs due to custom chips specifically designed for model inference [154]. These specialized chips not only enhance performance but also save substantial amounts of energy [155]. Consequently, deployment on common GPUs may be less efficient in terms of performance and energy consumption. This suggests that deploying models for inference might become less competitive without access to such specialized hardware. In the future, it may not be justifiable to self-host models on standard hardware from both energy and cost perspectives.

Lastly, for those opting to self-host models, the choice of serving engine is crucial. The comparison between the optimized vLLM serving engine and the traditional Transformers library revealed that vLLM offers substantial improvements in processing speed and energy efficiency. Specifically, vLLM reduced total energy consumption by a factor of 4.7 and increased prompt processing speed nearly sixfold compared to the Transformers implementation. These gains are attributed to vLLM's advanced optimization techniques, which alleviate bottlenecks and enhance GPU utilization. Selecting an optimized serving engine like vLLM can markedly improve the operational efficiency of self-hosted LLM deployments.

In summary, implementation factors—including the decision between self-hosting and using external APIs, the selection of hosting regions, the availability of specialized hardware, and the choice of an optimized serving engine—are critical considerations for optimizing the deployment of LLMs. By carefully evaluating these factors, organizations can balance performance, cost, and energy efficiency, contributing to more sustainable AI solutions.

6.2 Developing the Decision Framework

Implementing Large Language Models (LLMs) efficiently requires a systematic approach that aligns model capabilities with specific application needs while balancing performance, energy consumption, and cost. Based on the findings from the experiments in Sections 5.2 to 5.7, a revised decision framework is proposed. This framework consists of nine steps: defining usage patterns, defining LLM tasks, determining quality requirements and designing benchmarks, evaluating candidate models, applying quality-enhancing customizations, assessing models against comprehensive criteria, optimizing for efficiency, implementing and testing models, and selecting the optimal model configuration.

Step 1: Define Usage Patterns

The first step involves clearly defining the usage patterns for the LLM application. Understanding user access patterns and processing requirements is crucial for selecting an appropriate deployment strategy. Key considerations include:

- **Batch Processing Needs**: Will the application process large volumes of requests in fixed batches without strict uptime requirements? For example, a data analysis tool that processes datasets overnight.
- **High Concurrency Levels**: Will the application serve hundreds or thousands of users concurrently, especially during specific peak hours (e.g., 9 AM to 5 PM)? An instance of this is a customer service chatbot used by many users during business hours.

- **Irregular or Low-Frequency Access**: Will the application cater to a small number of users accessing the system irregularly? For instance, a specialized tool used by a few experts sporadically.

Defining these patterns is essential to determine whether self-hosting an LLM is justifiable from an efficiency perspective or if utilizing external APIs is more appropriate. High and consistent usage may warrant self-hosting to optimize performance and control, whereas irregular usage patterns might benefit from the flexibility and cost-effectiveness of external APIs.

Step 2: Define LLM Tasks

The second step is to precisely define the tasks the LLM is expected to perform, which involves identifying typical prompts and the nature of interactions with the model. Examples include:

- **Text Summarization**: Summarizing existing text documents, such as summarizing articles, reports, or legal documents.
- **Document-Based Question Answering**: Answering questions strictly based on existing documentation, such as an FAQ bot that provides answers sourced from company manuals.
- **Agentic Tasks with Retrieval Pipelines**: Operating within an agentic framework that involves retrieving information from various tools or databases, like financial analysis tools that pull data from multiple sources.
- **Domain-Specific Query Resolution**: Addressing queries requiring niche expertise, such as providing medical advice based on clinical guidelines.
- **Complex Problem Solving**: Solving tasks that require advanced reasoning, for instance, legal reasoning or scientific research assistance.

Determining the specific tasks aids in assessing the necessary model capabilities, such as the required knowledge base, reasoning ability, and ability to follow complex instructions.

Step 3: Determine Quality Requirements and Design Benchmarks

Based on the defined usage patterns and tasks, the next step is to establish explicit quality requirements for the LLM. This includes:

- **Defining Performance Metrics**: Establish acceptable thresholds for accuracy, relevance, coherence, and response time.
- **Designing Evaluation Benchmarks**: Create a set of representative test questions or scenarios that reflect typical user interactions. For each question, develop ideal answers that exemplify the expected quality and content.
- **Selecting Evaluation Methods**: Choose appropriate evaluation frameworks, such as LLM-as-a-Judge models (see Section 2.4) or human evaluation, to assess model outputs against the ideal answers.

For example, if the application is an FAQ bot, compile a list of frequently asked questions along with authoritative answers sourced from official documentation. These will serve as the benchmark for evaluating model responses.

Step 4: Evaluate Candidate Models

With benchmarks in place, various models can be evaluated to assess their suitability for the defined tasks. This involves:

- **Testing Multiple Models**: Test both proprietary models (e.g., GPT-4o, Anthropic's Claude) and open-source models (e.g., LLaMA, Mistral) in their default configurations.
- **Assessing Performance**: Evaluate model outputs against the established benchmarks, measuring how closely the responses match the ideal answers in terms of accuracy, completeness, and style.
- **Considering Task-Specific Strengths**: Recognize that some models may perform better on specific tasks despite lower overall rankings in general-purpose benchmarks. For instance, a smaller, specialized model might outperform larger models on domain-specific tasks.

Regular re-evaluation is important, as models receive updates that may alter their performance. Automating the evaluation pipeline ensures consistency and efficiency. Tools such as continuous integration systems can facilitate ongoing testing as models evolve.

Step 5: Apply Quality-Enhancing Customizations

If initial evaluations indicate that candidate models meet the basic requirements, quality-enhancing techniques (see Section 2.5.5) should be applied to further improve performance. These include:

- **Retrieval-Augmented Generation (RAG)**: Implement RAG setups that integrate external knowledge bases or document repositories to provide up-to-date and context-specific information.
- **Advanced Prompt Engineering**: Design custom prompts that employ techniques like Chain-of-Thought prompting, role prompting, or XML tags to guide the model's responses.
- **Tool Usage and Multi-Agent Pipelines**: Utilize frameworks that enable the model to interact with external tools or other specialized agents to perform complex tasks.

Building the application to be model-agnostic at this stage is beneficial, allowing flexibility in swapping models without significant changes to the system. Utilizing standard APIs and interfaces, such as the OpenAI API standard, can facilitate this flexibility.

Step 6: Assess Models Against Comprehensive Criteria

After applying customizations, models should be re-evaluated, focusing on:

- **Quality Improvements**: Measure the impact of customizations using the benchmarks. Determine if the enhancements meet the quality requirements established earlier.
- **Operational Costs**: Analyze costs related to computational resources, including energy consumption (see Section 3.2), hardware acquisition or rental fees, and any licensing fees for proprietary models.
- **Environmental Impact**: Consider emissions and environmental impact, taking into account factors such as the energy mix of the hosting region (see Table 12).
- **Governance and Compliance**: Evaluate models based on governance factors, including open-source availability, data privacy considerations, control over hosting environments, and the ethical stance and practices of the model providers.

This comprehensive assessment enables the selection of a subset of models that not only perform well but also align with organizational values and constraints.

Step 7: Optimize Selected Models for Efficiency

For self-hosted models that have progressed through the previous steps, further optimizations can be applied to reduce costs and energy consumption. These optimizations include:

- **Quantization Techniques**: Apply quantization methods to reduce model precision from 16-bit to 8-bit or 4-bit representations, significantly decreasing energy consumption with minimal loss of quality (see Section 5.6).
- **Model Pruning and Compression**: Remove unnecessary parameters from the model to reduce complexity and improve efficiency without substantially affecting performance.
- **Token Length Optimization**: Restrict input and output token lengths by eliminating unnecessary verbosity or implementing summary compression to reduce computational load (see Sections 5.3 and 5.4).
- **Optimized Serving Engines**: Utilize optimized serving engines like vLLM to enhance inference efficiency by reducing bottlenecks and improving hardware utilization (see Section 5.1).

Potential energy savings and quality impacts can be estimated using the tool presented in Section 6.3. These estimations help in making informed decisions about the trade-offs between efficiency and potential quality drawbacks.

Step 8: Implement and Test Models

Depending on the application's scope, multiple models or configurations may be implemented for testing in a live environment. Key considerations include:

- **A/B Testing**: Conduct A/B testing by exposing subsets of users to different models or configurations to compare performance and user satisfaction.
- **User Feedback Collection**: Gather qualitative and quantitative feedback from users regarding the quality of responses, response times, and overall satisfaction.
- **Performance Monitoring**: Monitor key performance indicators (KPIs), such as latency, throughput, error rates, and energy consumption during real-world usage.
- **Scalability Assessment**: Evaluate how models perform under varying loads to ensure they can handle expected traffic without degradation in performance.

Building the platform or application to be model-agnostic, possibly by using standard APIs and modular architectures, facilitates easy switching between models and rapid iteration.

Step 9: Select Optimal Model Configuration

After thorough testing and evaluation, the final step is to select the model or models that best meet the application's requirements. This involves:

- **Balancing Performance and Efficiency**: Choose models that provide sufficient performance for the specific tasks while offering the most sustainable and cost-effective solutions.
- **Task-Specific Model Selection**: Consider using different models for specialized sub-tasks within the application if it enhances overall efficiency and performance.
- **Future-Proofing**: Select models and configurations that allow for scalability and adaptability to future needs, including the integration of new models as they become available.

The selected model configuration should align with the organization's goals regarding performance, cost, sustainability, and ethical considerations.

Hosting Considerations

An important aspect of implementing LLMs is deciding on the hosting strategy, which affects both performance and environmental impact. If self-hosting is chosen, selecting a hosting region with a low-carbon energy mix can reduce emissions associated with energy consumption. Table 12 provides an overview of potential hosting regions based on their carbon intensity.

Region	Country	Carbon Intensity	Primary Energy Source
EMEA	Iceland	28 (gCO_2/kWh)	Geothermal, Hydropower
	Sweden	25 (gCO_2/kWh)	Hydropower, Nuclear
	Switzerland	68 (gCO_2/kWh)	Hydropower, Nuclear
	France[8]	53 (gCO_2/kWh)	Nuclear
	Germany	400 (gCO_2/kWh)	Coal, Wind
Americas	East Canada	31 (gCO_2/kWh)	Hydropower
	West Canada	72 (gCO_2/kWh)	Hydropower
	Brazil	82 (gCO_2/kWh)	Hydropower
	USA[9]	411 (gCO_2/kWh)	Gas
APAC	New Zealand	97 (gCO_2/kWh)	Hydropower
	Japan	460 (gCO_2/kWh)	Gas, Coal
	Australia	492 (gCO_2/kWh)	Coal
	India	660 (gCO_2/kWh)	Coal

Table 12: Carbon intensity of potential hosting regions (2023 data from electricityMaps [55])

Choosing regions with low-carbon energy sources, such as Iceland, Norway, or Sweden, can significantly reduce the environmental impact of self-hosted LLM deployments. Additionally, considering data sovereignty laws, latency requirements, and availability of specialized hardware in these regions is essential.

Continuous Monitoring and Adaptation

After deployment, continuous monitoring of model performance, user satisfaction, and energy consumption is essential. Key activities include:

- **Regular Performance Reviews**: Periodically re-evaluate models using the benchmarks to ensure consistent performance over time.
- **Monitoring Technological Advances**: Stay informed about new models, updates, and optimization techniques that could enhance performance or efficiency.
- **User Feedback Integration**: Incorporate user feedback into ongoing improvements, adapting the model or prompts as necessary.
- **Scalability Planning**: Prepare for changes in usage patterns, ensuring that the infrastructure can scale accordingly without compromising efficiency.

[8] It is important to highlight that France uses an extremely high share of nuclear power to produce its energy. While nuclear power is a low-carbon energy source, it cannot be counted as sustainable when factoring in the nuclear waste it creates.

[9] The consumption varies based on the reviewed state. Generally, the central states have the worst energy mix, while the western states show the best energy mix.

Adapting to new models or updates and re-evaluating configurations in light of technological advancements ensures that the application remains efficient, effective, and aligned with sustainability objectives over time.

6.3 Calculating Energy Savings for Existing Implementations

Estimating energy consumption and emissions during the inference phase of LLMs is inherently challenging due to the significant dependence on hardware configurations. Inference efficiency is profoundly influenced by hardware-specific factors such as memory bandwidth, memory size, and processing overhead (as discussed in Section 4). Consequently, providing precise energy consumption estimates without specific information about the hardware setup is not feasible. Offering estimates for particular hardware configurations is also impractical given the rapidly evolving hosting landscape and the multitude of available hardware options. Advances in hardware technology and the emergence of specialized inference accelerators continuously change the performance characteristics of potential setups, making it difficult to generate estimates that are both accurate and broadly applicable.

To address this challenge, an interactive framework has been developed to assist practitioners in calculating relative energy savings when applying specific customization approaches. This framework is implemented as a Streamlit dashboard[10], providing a user-friendly interface to explore the impact of various optimization strategies on energy consumption.

The quantitative results from this research have been consolidated into regression models that capture the relationships between energy consumption and key factors affecting LLM inference. These regressions enable the calculation of relative energy changes when specific customization approaches are applied, offering actionable insights without requiring exact hardware specifications.

For **model size**, an exponential regression was employed, as it best fit the observed energy consumption data in the most energy-optimal test setups (see Section 5.2). The exponential model reflects how energy usage increases disproportionately with the number of model parameters due to the increasing hardware demands.

Regarding **input token length**, a linear regression was used to model the energy consumption (refer to Section 5.4). Experiments demonstrated that energy usage scales linearly with the number of input tokens. This linear relationship arises because longer input sequences demand more memory and processing power, especially due to the substantial impact on the Key-Value (KV) cache size and management.

For **output token length**, a quadratic regression was utilized (see Section 5.3). The quadratic nature of the relationship becomes significant when generating longer outputs. For shorter sequences, the relationship between output token length and energy consumption is approximately linear, but as the number of generated tokens increases into the thousands, energy consumption accelerates more rapidly due to the growing context length with each generated token.

For **quantization levels**, fixed energy change factors observed during the tests were used (refer to Section 5.6). These factors represent the relative changes in energy consumption when transitioning between different precision levels. The specific multipliers applied are:

- Converting from `bf16` (16-bit floating point) to `fp8` (8-bit floating point) reduces energy consumption to approximately 40% of the original.

[10] Accessible on the Framework Page of the Streamlit Dashboard at `https://thesis.d-wetzel.de` [1]

- Converting from `bf16` to `int4` (4-bit integer) reduces energy consumption to approximately 31.88% of the original.
- Converting from `fp8` to `bf16` increases energy consumption by approximately 2.5 times.
- Converting from `int4` to `bf16` increases energy consumption by approximately 3.14 times.

These factors were derived from empirical observations and are implemented in the interactive framework to facilitate energy savings estimations for different quantization scenarios.

In terms of **serving engine optimization**, a fixed factor based on practical observations was applied. Utilizing an optimized serving engine like vLLM, as opposed to a standard implementation with the Transformers library, resulted in an approximate 4.7-fold reduction in energy consumption (see Section 5.1). In the framework, this is represented as a binary choice—either employing an optimized serving engine or not—allowing users to assess the potential energy savings from this optimization.

Additionally, the interactive framework includes a module that examines potential emission reductions across different hosting regions. Given that the carbon intensity of electricity production varies by location, selecting a hosting region with a lower carbon footprint can significantly decrease overall emissions (refer to Section 2.3). The framework provides visualizations of energy emission data changes by time of day and month of the year, enabling users to make informed decisions about hosting locations and operational strategies.

By inputting parameters such as model size, quantization levels, token lengths, and serving engine choices into the interactive dashboard, users can visualize the impact of these factors on energy consumption. This tool allows practitioners to evaluate and compare different optimization strategies, facilitating more energy-efficient and sustainable LLM deployments.

In summary, while precise energy consumption estimates for LLM inference require detailed hardware information, the regression models and observed factors from this research provide valuable means to assess relative energy savings. By applying these insights through the interactive framework, organizations can make informed decisions to optimize their LLM implementations for both performance and energy efficiency.

7 DISCUSSION

This section interprets the study's findings and discusses their implications for the sustainable deployment of large language models. The evaluated customization techniques are analyzed in terms of their impact on energy efficiency and model quality. The research's limitations are acknowledged, and avenues for future work are proposed. Finally, the conclusion emphasizes on the potential for optimizing LLM deployments to mitigate environmental impact while maintaining high output quality.

7.1 Summary

This research investigated strategies to reduce energy consumption during LLM inference without compromising model quality. The study focused on evaluating customization techniques, including model quantization, advanced prompt engineering with Retrieval-Augmented Generation (RAG), and serving engine optimization. Experiments were conducted using various models, such as LLaMA 3.1 and Mistral Nemo, across different quantization levels and prompt configurations.

Key findings demonstrate that quantization can significantly reduce energy consumption while maintaining acceptable levels of model quality. For instance, quantizing the LLaMA 3.1 70B model from 16-bit floating point (`bf16`) to 8-bit floating point (`fp8`) resulted in energy savings of approximately 60% with minimal impact on output quality, as evidenced by the Arena Hard Auto Benchmark scores. Further reducing precision to 4-bit integer (`int4`) led to additional energy savings but introduced a more noticeable decrease in model quality.

Advanced prompt engineering and the integration of RAG techniques enhanced model performance, particularly for models with lower baseline capabilities. Embedding enriched guidance into the system prompts improved Arena Scores by up to 27 points for certain models. However, this improvement was accompanied by an increase in energy consumption ranging from 1.4 to 1.7 times the original usage, primarily due to longer input sequences.

Serving engine optimization, specifically the use of the vLLM engine, demonstrated substantial improvements in both processing speed and energy efficiency compared to the traditional Transformers library implementation. The vLLM engine reduced total energy consumption by a factor of 4.7 and increased prompt processing speed by approximately 5.6 times.

Selecting appropriate model sizes was also critical in optimizing energy consumption and inference performance. Smaller models exhibited significant reductions in energy usage but at the potential cost of decreased output quality. Aligning the hardware configuration with the model's computational and memory requirements further enhanced efficiency by minimizing unnecessary overhead.

Collectively, these findings contribute to the development of a comprehensive decision framework for optimizing LLM deployments. By balancing factors such as model size, quantization level, prompt construction, and serving infrastructure, it is possible to achieve substantial energy savings while maintaining or improving model performance. The proposed framework provides practical guidelines for practitioners to implement sustainable and efficient LLM applications, addressing the environmental concerns associated with the growing adoption of AI technologies.

7.2 Implications of the Results

The findings of this research have important implications for the deployment and sustainability of Large Language Models. They demonstrate that specific customizations can effectively reduce energy consumption during inference without significantly compromising model quality, thereby providing practical solutions to the research question:

How can customizations to Large Language Models reduce energy consumption during inference without compromising model quality?

Firstly, the effectiveness of quantization techniques suggests that lowering the precision of model weights is a viable strategy for enhancing the energy efficiency of LLM deployments. By reducing computational demands, quantization decreases operational costs and environmental impact. This finding implies that organizations can adopt quantization to deploy large models more sustainably, especially when energy resources or budgets are constrained.

Secondly, the benefits of advanced prompt engineering and RAG knowledge embedding highlight the potential for improving model output quality without relying on larger, more resource-intensive models. By providing enriched context and guidance within prompts, even smaller models can achieve enhanced performance. This approach enables more efficient use of computational resources, suggesting that emphasis on prompt design and external knowledge integration can be a key strategy in sustainable AI practices.

Thirdly, the substantial gains observed through serving engine optimization, particularly with the vLLM engine, underscore the importance of infrastructure-level enhancements in AI deployments. This indicates that software optimizations can play a significant role in reducing energy consumption and improving performance, independent of model architecture. It suggests that organizations should consider investing in optimized serving infrastructures as part of their sustainability strategies.

Moreover, the results imply that careful selection of model sizes based on specific application requirements is critical for balancing performance and energy efficiency. The trade-offs between model size, energy consumption, and output quality necessitate thoughtful evaluation to determine the most suitable model for a given task. This finding encourages practitioners to assess the necessity of larger models against their operational costs and environmental impact.

Overall, these implications contribute to advancing sustainable AI practices by providing actionable insights into how LLMs can be optimized for energy efficiency without sacrificing performance. They emphasize that targeted customizations and optimizations can address the environmental concerns associated with LLM deployments, supporting broader efforts to reduce the carbon footprint of artificial intelligence technologies.

The development of the decision framework, based on these findings, offers a practical tool for practitioners to systematically implement these strategies. By considering factors such as quantization, prompt engineering, model size selection, and serving infrastructure, organizations can make informed decisions that align with both performance objectives and sustainability goals.

In conclusion, this research provides evidence that energy consumption during LLM inference can be significantly reduced through specific customizations without compromising model quality. These insights directly address the research question and offer a foundation for future work in developing more sustainable AI systems.

7.3 Limitations

Although this research provides valuable insights into reducing energy consumption during LLM inference through various customization techniques, some limitations should be acknowledged.

Firstly, the study did not quantitatively investigate model fine-tuning or training procedures. Including these aspects could have offered a more comprehensive understanding of how training strategies impact energy efficiency during inference.

Secondly, the research encompassed multiple topics, including quantization, prompt engineering, and Retrieval-Augmented Generation (RAG). Due to the breadth of these subjects, the depth of exploration within each area was necessarily constrained. A more focused investigation of individual techniques could have provided more detailed insights.

Thirdly, the number of tests conducted was limited. A test across a larger magnitude of models, quantization techniques, and multiple different prompt engineering and knowledge embedding techniques could have been interesting. Again, due to the necessary constraint of this thesis, a multitude of methods had to be investigated which resulted in the depth of this work.

Fourthly, the study focused exclusively on LLMs and did not extend to other generative AI models, such as diffusion models. This limitation may affect the applicability of the findings to a broader range of AI technologies. In addition to that, a large number of foundation models got released within the span of this research. While this thesis could shift towards LLaMA 3.1 [21] for a majority of the tests, it was not feasible to conduct further tests on LLaMA 3.2 [22] (which was released on September 25th).

Lastly, while energy consumption was thoroughly measured and can be largely translated to operational costs, the research did not include a detailed cost assessment. Incorporating an economic analysis could have strengthened the practical relevance of the findings by allowing organizations to consider both energy and financial implications when adopting the recommended optimization strategies.

7.4 Future Research

Building upon the findings of this study, several avenues for future research are identified.

Firstly, deeper exploration into each customization technique is warranted. In the context of Retrieval-Augmented Generation (RAG), investigating data curation methods—such as comparing standard vector similarity searches with knowledge graphs or exploring novel approaches—could optimize information retrieval for energy efficiency and model performance.

Secondly, exploring more advanced quantization techniques that reduce precision below 4 bits could reveal additional opportunities for energy savings. Such investigations should carefully consider the impact on model quality, as extreme quantization may introduce significant degradation in performance.

Thirdly, evaluating the energy consumption implications of multi-agent pipeline architectures presents another potential research direction. Assessing whether utilizing multiple smaller agents in a pipeline instead of a single large model can reduce overall energy consumption would provide valuable insights for optimizing LLM deployments. While smaller agents might reduce individual energy usage, managing multiple models simultaneously could increase complexity and resource demands.

Fourthly, extending the research to include other generative AI models, such as diffusion models, could broaden the applicability of the findings and contribute to a more holistic understanding of energy efficiency in AI deployments.

Fifthly, a detailed assessment of specialized serverless cloud inference services in terms of emissions and energy consumption would be beneficial. Comparing different cloud providers could offer insights into how infrastructure choices impact energy efficiency and carbon footprint, aiding practitioners in making informed deployment decisions.

Moreover, incorporating an economic analysis in future studies could strengthen the practical relevance of the findings by allowing organizations to consider both energy and financial implications when adopting optimization strategies.

Additionally, near the end of this research, the team behind the arena-hard-auto benchmark introduced an advancement by disentangling style and substance in the Chatbot Arena [156]. They identified that models prioritizing length, style, and markdown formatting in their outputs scored higher in side-by-side comparisons, potentially skewing the accuracy of the scores towards models that focus on presentation over content. This concern arose due to suspicions that some foundation model providers might fine-tune their models to produce longer, more elaborately formatted responses to achieve better results in evaluations. Considering findings related to overfitted models on multiple-choice benchmarks [66], [67], this possibility warrants attention. To address this issue, the team developed a method to mitigate such biases, resulting in significant changes in the arena scores of different models. Retesting models with these adjustments could provide more accurate assessments of model quality and energy efficiency. However, it is important to recognize that widely adopted benchmarks may incentivize efforts to game the system, highlighting the need for continuous refinement of evaluation methodologies.

Furthermore, practical applications of hyper-optimized inference setups in production systems, such as Apple's recent showcase of their machine learning models branded as Apple Intelligence, demonstrate the potential for advancing model inference efficiency [97], [98]. The use of mixed-precision quantization and other optimization techniques resulted in impressive efficiency gains, enabling a 3 billion parameter model to run at 30 tokens per second with a time-to-first-token of 0.6 milliseconds on edge devices. Investigating these techniques further could provide insights into achieving substantial memory footprint reductions without significant performance decreases.

Lastly, examining the environmental impact of generative AI beyond energy consumption, such as water usage associated with cooling data centers, could contribute to a more comprehensive understanding of sustainable AI practices. Considering factors like water consumption in data centers adds another dimension to assessing the environmental footprint of AI technologies. Recent research has revealed the significant impact that energy consumption has on water usage and waste [157].

Exploring the social implications of generative AI is also a pressing area for future research. Issues related to the ownership and copyright of training data, the form of consent required for using user data to train models, and the implications of predictive algorithms on social media and other services need thorough investigation. While this research focused on sustainability, ethical and social considerations are equally important and warrant dedicated study.

In addition, some academic work suggests that increasing the use of carbon-free energy sources negates the need to worry about emissions associated with AI [158]. However, it is crucial to acknowledge that energy demand is increasing dramatically. Sustainability reports, such as Google's Sustainability Report for 2023 [28], indicate that even with advancements in renewable energy adoption, the rising energy requirements present challenges. Achieving 100% renewable energy sourcing does not guarantee that energy supply will meet future demands, especially with the rapid adoption of resource-intensive technologies like generative AI. Therefore, continued research on reducing the energy consumption of generative AI is essential to ensure that this technology does not significantly contribute to climate change.

In summary, future research should delve deeper into optimization techniques, explore additional dimensions of environmental impact, and consider the broader ethical, social, and infrastructural implications of deploying large-scale AI models. Such comprehensive investigations will contribute to the development of sustainable and responsible AI practices.

7.5 Conclusion

The unprecedented growth in generative AI technologies has led to a dramatic surge in global energy demands, highlighting urgent concerns about environmental sustainability. Recent developments underscore the scale of this issue. OpenAI has proposed that the U.S. government permit the construction of data centers with capacities of 5 gigawatts each, expressing intentions to build five to seven such centers across multiple states [159]. This expansion would result in an additional 25 to 35 gigawatts of energy consumption. To contextualize, only the largest nuclear power plants are capable of producing 5 gigawatts. In Germany, nuclear power plants had capacities of up to 1.5 gigawatts.

Similarly, Amazon Web Services (AWS) has acquired a data center directly powered by a nuclear power plant [31]. These developments indicate a foreseeable massive increase in energy demand attributable to AI technologies. While utilizing low-carbon energy sources can mitigate carbon emissions, meeting such enormous energy demands sustainably remains a significant challenge.

Major technology corporations possess the financial resources and flexibility to invest in renewable (or at least low-carbon) energy regardless of its price. With the social pressure and the very optimistic sustainability goals these companies set for themselves, it is also very likely that they will purchase as much low-carbon electricity as they can. However, basic market dynamics suggest that increasing demand from these entities would lead to price inflation for renewable energy, affecting other sectors that may not have the same financial flexibility. Consequently, it is not equitable to expect smaller companies or consumers to absorb higher energy costs driven by the substantial demands of large corporations. If the demand and price for renewable energy rise, the feasibility of choosing these types of energy decreases for a majority of consumers.

Corporations responsible for rising energy demands should also be accountable for optimizing energy efficiency in their operations. They should investigate every possible implementation decision to save energy and reduce emissions before adopting new, energy-intensive technologies. While this thesis cannot single-handedly reduce the energy consumption in the burgeoning AI market, it provides relevant research for practitioners. It highlights the immense energy-saving potentials achievable when critically assessing the implementation of Large Language Models (LLMs).

This thesis offers practical frameworks and tools to estimate potential energy savings, guiding decision-making in the deployment of LLMs. By adopting the strategies explored in this research, significant energy reductions can be realized without compromising model quality. This work succeeds in providing the necessary knowledge and tools for conscious evaluation of LLM implementation decisions.

In the face of rapidly increasing energy demands, it is imperative for both industry and researchers to prioritize energy efficiency alongside performance. The environmental impact associated with the growth of AI technologies must be addressed proactively. Through targeted optimizations and responsible implementation strategies, it is possible to harness the benefits of AI sustainably. Continued efforts in this area are essential to ensure that technological advancements contribute positively to society without exacerbating environmental challenges.

REFERENCES

[1] Daniel Wetzel. "Interactive streamlit dashboard." (2024), [Online]. Available: https://llm-emissions.streamlit.app (visited on 09/28/2024).

[2] Daniel Wetzel. "Accompanying GitHub code repository." (2024), [Online]. Available: https://github.com/danielwetzel/llm-customization (visited on 09/28/2024).

[3] Deloitte, "State of gen AI report," Deloitte, 2023. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/de/Documents/Innovation/deloitte_state-of-gen-ai-report.pdf.

[4] MIT-CIO and Databricks, "MIT-CIO generative AI report," MIT-CIO and Databricks, 2023. [Online]. Available: https://www.databricks.com/sites/default/files/2023-07/ebook_mit-cio-generative-ai-report.pdf.

[5] Gartner, "Understand and exploit generative AI with gartner's new impact radar," *Gartner*, 2023. [Online]. Available: https://www.gartner.com/en/articles/understand-and-exploit-gen-ai-with-gartner-s-new-impact-radar.

[6] McKinsey & Company, "The economic potential of generative AI: The next productivity frontier," McKinsey & Company, 2023. [Online]. Available: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier.

[7] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, Version Number: 7, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762 (visited on 07/01/2024).

[8] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language models are few-shot learners*, Jul. 22, 2020. arXiv: 2005.14165[cs]. [Online]. Available: http://arxiv.org/abs/2005.14165 (visited on 06/27/2024).

[9] OpenAI. "GPT-4." (2024), [Online]. Available: https://openai.com/index/gpt-4/.

[10] OpenAI. "GPT-4o mini: Advancing cost-efficient intelligence." (2024), [Online]. Available: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

[11] OpenAI. "Introducing OpenAI o1 preview." (2024), [Online]. Available: https://openai.com/index/introducing-openai-o1-preview/.

[12] OpenAI. "Hello GPT-4o." (2024), [Online]. Available: https://openai.com/index/hello-gpt-4o/.

[13] Anthropic. "Claude 3 family." (2024), [Online]. Available: https://www.anthropic.com/news/claude-3-family.

[14] Anthropic. "Claude 3.5: Sonnet." (2024), [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet.

[15] DeepMind. "Gemini models: Breakthrough AI with faster models, longer context, and AI agents." (2024), [Online]. Available: https://deepmind.google/technologies/gemini/.

[16] G. Gemini Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," Google, 2024. [Online]. Available: https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf.

[17] G. Team, R. Anil, S. Borgeaud, *et al.*, *Gemini: A family of highly capable multimodal models*, Jun. 17, 2024. arXiv: 2312.11805[cs]. [Online]. Available: http://arxiv.org/abs/2312.11805 (visited on 09/26/2024).

[18] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, *Estimating the carbon footprint of BLOOM, a 176b parameter language model*, Version Number: 1, 2022. DOI: 10.48550/ARXIV.2211.02001. [Online]. Available: https://arxiv.org/abs/2211.02001 (visited on 06/27/2024).

[19] H. Touvron, T. Lavril, G. Izacard, *et al.*, *LLaMA: Open and efficient foundation language models*, Feb. 27, 2023. arXiv: 2302.13971[cs]. [Online]. Available: http://arxiv.org/abs/2302.13971 (visited on 06/27/2024).

[20] H. Touvron, L. Martin, K. Stone, *et al.*, *Llama 2: Open foundation and fine-tuned chat models*, Jul. 19, 2023. arXiv: 2307.09288[cs]. [Online]. Available: http://arxiv.org/abs/2307.09288 (visited on 06/27/2024).

[21] Meta. "Llama 3.1 prompt template," LLaMA Model Cards & Prompt Formats. (2024), [Online]. Available: https://llama.meta.com/docs/model-cards-and-prompt-formats/llama3_1 (visited on 07/29/2024).

[22] Meta. "Llama 3.2: Connect 2024 vision for edge and mobile devices." (2024), [Online]. Available: https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

[23] Mistral AI. "Mistral AI technology." (2024), [Online]. Available: https://mistral.ai/technology/.

[24] Mistral AI. "Large enough - mistral large 2407." (2024), [Online]. Available: https://mistral.ai/news/mistral-large-2407/.

[25] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, *Scaling laws for neural language models*, Jan. 22, 2020. arXiv: 2001.08361[cs,stat]. [Online]. Available: http://arxiv.org/abs/2001.08361 (visited on 06/27/2024).

[26] J. Hoffmann, S. Borgeaud, A. Mensch, *et al.*, *Training compute-optimal large language models*, Mar. 29, 2022. arXiv: 2203.15556[cs]. [Online]. Available: http://arxiv.org/abs/2203.15556 (visited on 09/14/2024).

[27] I. Rahman-Jones. "AI drives 48% increase in google emissions," BBC News. (2024), [Online]. Available: https://www.bbc.com/news/articles/c51yvz51k2xo.

[28] Google, "Google 2024 environmental report," Google, 2024. [Online]. Available: https://www.gstatic.com/gumdrop/sustainability/google-2024-environmental-report.pdf.

[29] Microsoft, "Microsoft sustainability report for fiscal year 2023," Microsoft, 2024. [Online]. Available: https://aka.ms/SustainabilityReport2024.

[30] Amazon, "Amazon 2023 sustainability report," Amazon, 2023. [Online]. Available: https://sustainability.aboutamazon.com/content/dam/sustainability-marketing-site/pdfs/reports-docs/2023-amazon-sustainability-report.pdf.

[31] L. Harris, "Amazon presses the nuclear button," *Financial Times*, 2024. [Online]. Available: https://www.ft.com/content/f073b54d-9290-49b4-8ee7-56b4fb3d8177.

[32] W. Huang, X. Zheng, X. Ma, *et al.*, *An empirical study of LLaMA3 quantization: From LLMs to MLLMs*, Jul. 19, 2024. arXiv: 2404.14047[cs]. [Online]. Available: http://arxiv.org/abs/2404.14047 (visited on 08/04/2024).

[33] J. Lin, J. Tang, H. Tang, *et al.*, *AWQ: Activation-aware weight quantization for LLM compression and acceleration*, Jul. 18, 2024. arXiv: 2306.00978[cs]. [Online]. Available: http://arxiv.org/abs/2306.00978 (visited on 08/04/2024).

[34] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, *GPTQ: Accurate post-training quantization for generative pre-trained transformers*, Mar. 22, 2023. arXiv: 2210.17323[cs]. [Online]. Available: http://arxiv.org/abs/2210.17323 (visited on 08/04/2024).

[35] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort, *FP8 quantization: The power of the exponent*, Feb. 23, 2024. arXiv: 2208.09225[cs]. [Online]. Available: http://arxiv.org/abs/2208.09225 (visited on 09/10/2024).

[36] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[37] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. Cambridge, Mass: MIT Press, 1999, 680 pp., ISBN: 978-0-262-13360-9.

[38] R. Sennrich, B. Haddow, and A. Birch, *Neural machine translation of rare words with subword units*, Jun. 10, 2016. arXiv: 1508.07909[cs]. [Online]. Available: http://arxiv.org/abs/1508.07909 (visited on 09/24/2024).

[39] D. Operationnelle, Y. Bengio, R. Ducharme, P. Vincent, and C. Mathematiques, "A neural probabilistic language model," Oct. 2001.

[40] G. Developers, *Embedding space - machine learning crash course*, 2024. [Online]. Available: https://developers.google.com/machine-learning/crash-course/embeddings/embedding-space?hl=de.

[41] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient estimation of word representations in vector space*, Sep. 6, 2013. arXiv: 1301.3781[cs]. [Online]. Available: http://arxiv.org/abs/1301.3781 (visited on 09/24/2024).

[42] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *EMNLP*, vol. 14, Jan. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[43] Z. S. Harris, "Distributional structure," *WORD*, vol. 10, no. 2, pp. 146–162, Aug. 1954, ISSN: 0043-7956. DOI: 10.1080/00437956.1954.11659520. [Online]. Available: http://dx.doi.org/10.1080/00437956.1954.11659520 (visited on 03/31/2017).

[44] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, L. Vanderwende, H. Daumé III, and K. Kirchhoff, Eds., Atlanta, Georgia: Association for Computational Linguistics, Jun. 2013, pp. 746–751. [Online]. Available: https://aclanthology.org/N13-1090.

[45] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," Jan. 1, 2019.

[46] B. Courty, V. Schmidt, S. Luccioni, *et al.*, *Mlco2/codecarbon: V2.4.1*, version v2.4.1, May 2024. DOI: 10.5281/zenodo.11171501. [Online]. Available: https://doi.org/10.5281/zenodo.11171501.

[47] L. F. W. Anthony, B. Kanding, and R. Selvan, *Carbontracker: Tracking and predicting the carbon footprint of training deep learning models*, Jul. 6, 2020. arXiv: 2007.03051[cs, eess, stat]. [Online]. Available: http://arxiv.org/abs/2007.03051 (visited on 06/27/2024).

[48] L. B. Heguerte, A. Bugeau, and L. Lannelongue, "How to estimate carbon footprint when training deep learning models? a guide and review," *Environmental Research Communications*, vol. 5, no. 11, p. 115 014, Nov. 1, 2023, ISSN: 2515-7620. DOI: 10.1088/2515-7620/acf81b. arXiv: 2306.08323[cs]. [Online]. Available: http://arxiv.org/abs/2306.08323 (visited on 09/17/2024).

[49] Benjamin Davy. "Building an AWS EC2 carbon emissions dataset," Teads Engineering. (Sep. 23, 2021), [Online]. Available: https://medium.com/teads-engineering/building-an-aws-ec2-carbon-emissions-dataset-3f0fd76c98ac (visited on 07/09/2024).

[50] S. Graham and O. Yashkova, "Energy and carbon efficiency benefits of public cloud computing over enterprise datacenters," IDC, Sponsored by AWS, United Kingdom, InfoBrief, Apr. 2024. [Online]. Available: https://www.idc.com/.

[51] Amazon Web Services. "Tips from IDC to improve the environmental sustainability of your IT workloads." (Apr. 2024), [Online]. Available: https://aws.amazon.com/de/blogs/aws-insights/tips-from-idc-to-improve-the-environmental-sustainability-of-your-it-workloads/.

[52] Microsoft Azure. "Sharing the latest improvements to efficiency in microsoft's datacenters." (Aug. 2023), [Online]. Available: https://azure.microsoft.com/en-us/blog/sharing-the-latest-improvements-to-efficiency-in-microsoft-s-datacenters/.

[53] Google. "Efficiency: How we do it." (2024), [Online]. Available: https://www.google.com/about/datacenters/efficiency/.

[54] Google. "Google cloud region carbon information." (2024), [Online]. Available: https://cloud.google.com/sustainability/region-carbon.

[55] electricityMaps, *electricityMaps: Live visualization of electricity CO2 emissions*, 2024. [Online]. Available: https://app.electricitymaps.com/map.

[56] Tianle Li, Wei-Lin Chiang, Evan Frick, *et al.* "LMSYS chatbot arena: Benchmarking LLMs in the wild," LMSYS Chatbot Arena. (2024), [Online]. Available: https://lmarena.ai/ (visited on 08/27/2024).

[57] W.-L. Chiang, L. Zheng, Y. Sheng, *et al.*, *Chatbot arena: An open platform for evaluating LLMs by human preference*, Mar. 6, 2024. arXiv: 2403.04132[cs]. [Online]. Available: http://arxiv.org/abs/2403.04132 (visited on 06/27/2024).

[58] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3, p. 324, Dec. 1952, ISSN: 00063444. DOI: 10.2307/2334029. [Online]. Available: https://www.jstor.org/stable/2334029?origin=crossref (visited on 08/27/2024).

[59] V. Vovk and R. Wang, "E-values: Calibration, combination, and applications," *The Annals of Statistics*, vol. 49, no. 3, Jun. 1, 2021, ISSN: 0090-5364. DOI: 10.1214/20-AOS2020. arXiv: 1912.06116[math,stat]. [Online]. Available: http://arxiv.org/abs/1912.06116 (visited on 08/27/2024).

[60] Y. Wang, X. Ma, G. Zhang, *et al.*, *MMLU-pro: A more robust and challenging multi-task language understanding benchmark*, Jun. 23, 2024. arXiv: 2406.01574[cs]. [Online]. Available: http://arxiv.org/abs/2406.01574 (visited on 08/26/2024).

[61] D. Hendrycks, C. Burns, S. Basart, *et al.*, *Measuring massive multitask language understanding*, Jan. 12, 2021. arXiv: 2009.03300[cs]. [Online]. Available: http://arxiv.org/abs/2009.03300 (visited on 08/30/2024).

[62] D. Hendrycks, C. Burns, S. Kadavath, *et al.*, *Measuring mathematical problem solving with the MATH dataset*, Nov. 8, 2021. arXiv: 2103.03874[cs]. [Online]. Available: http://arxiv.org/abs/2103.03874 (visited on 08/26/2024).

[63] A. Wang, Y. Pruksachatkun, N. Nangia, *et al.*, *SuperGLUE: A stickier benchmark for general-purpose language understanding systems*, Feb. 12, 2020. arXiv: 1905.00537[cs]. [Online]. Available: http://arxiv.org/abs/1905.00537 (visited on 08/26/2024).

[64] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, *GLUE: A multi-task benchmark and analysis platform for natural language understanding*, Feb. 22, 2019. arXiv: 1804.07461[cs]. [Online]. Available: http://arxiv.org/abs/1804.07461 (visited on 08/30/2024).

[65] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, *HellaSwag: Can a machine really finish your sentence?* May 19, 2019. arXiv: 1905.07830[cs]. [Online]. Available: http://arxiv.org/abs/1905.07830 (visited on 08/26/2024).

[66] V. Gupta, D. Pantoja, C. Ross, A. Williams, and M. Ung, *Changing answer order can decrease MMLU accuracy*, Jun. 27, 2024. arXiv: 2406.19470[cs]. [Online]. Available: http://arxiv.org/abs/2406.19470 (visited on 08/13/2024).

[67] P. Pezeshkpour and E. Hruschka, *Large language models sensitivity to the order of options in multiple-choice questions*, Aug. 22, 2023. arXiv: 2308.11483[cs]. [Online]. Available: http://arxiv.org/abs/2308.11483 (visited on 08/13/2024).

[68] Edward Beeching, Clémentine Fourrier, Thomas Wolf, *et al.* "Open LLM leaderboard," Hugging Face. (2023), [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard-old/open_llm_leaderboard (visited on 08/27/2024).

[69] Clémentine Fourrier, Thomas Wolf, Nathan Habib, and Julien Launay. "What's going on with the open LLM leaderboard?" Hugging Face. (Jun. 23, 2023), [Online]. Available: https://huggingface.co/blog/open-llm-leaderboard-mmlu (visited on 08/27/2024).

[70] Clémentine Fourrier, Thomas Wolf, Nathan Habib, Alex Cabrera, and Stella Biderman. "Open LLM leaderboard: DROP deep dive," Hugging Face. (Jan. 12, 2023), [Online]. Available: https://huggingface.co/blog/open-llm-leaderboard-drop (visited on 08/27/2024).

[71] F. Chollet, *On the measure of intelligence*, Nov. 25, 2019. arXiv: 1911.01547[cs]. [Online]. Available: http://arxiv.org/abs/1911.01547 (visited on 08/30/2024).

[72] Clementine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. "Performances are plateauing, let's make the leaderboard steep again," Hugging Face. (Jun. 26, 2024), [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/blog (visited on 08/27/2024).

[73] Clementine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, and Thomas Wolf. "Open LLM leaderboard v2," Hugging Face. (Jun. 26, 2024), [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard (visited on 08/27/2024).

[74] D. Rein, B. L. Hou, A. C. Stickland, *et al.*, *GPQA: A graduate-level google-proof q&a benchmark*, Nov. 20, 2023. arXiv: 2311.12022[cs]. [Online]. Available: http://arxiv.org/abs/2311.12022 (visited on 08/27/2024).

[75] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, and G. Durrett, *MuSR: Testing the limits of chain-of-thought with multistep soft reasoning*, Mar. 23, 2024. arXiv: 2310.16049[cs]. [Online]. Available: http://arxiv.org/abs/2310.16049 (visited on 08/27/2024).

[76] T. Li, W.-L. Chiang, E. Frick, *et al.*, *From crowdsourced data to high-quality benchmarks: Arena-hard and BenchBuilder pipeline*, Jun. 17, 2024. arXiv: 2406.11939[cs]. [Online]. Available: http://arxiv.org/abs/2406.11939 (visited on 08/04/2024).

[77] Tianle Li*, Wei-Lin Chiang*, Evan Frick, *et al.* "From live data to high-quality benchmarks: The arena-hard pipeline," lmsys Blog. (Apr. 19, 2024), [Online]. Available: https://lmsys.org/blog/2024-04-19-arena-hard/ (visited on 08/04/2024).

[78] Tianle Li*, Wei-Lin Chiang*, Evan Frick, *et al.* "Arena-hard-auto github repository," Github. (2024), [Online]. Available: https://github.com/lm-sys/arena-hard-auto (visited on 08/27/2024).

[79] J. Wei, X. Wang, D. Schuurmans, *et al.*, *Chain-of-thought prompting elicits reasoning in large language models*, Jan. 10, 2023. arXiv: 2201.11903[cs]. [Online]. Available: http://arxiv.org/abs/2201.11903 (visited on 08/30/2024).

[80] B. Efron and R. Tibshirani, "The bootstrap method for assessing statistical accuracy," *Behaviormetrika*, vol. 12, no. 17, pp. 1–35, Jan. 1985, ISSN: 0385-7417, 1349-6964. DOI: 10.2333/bhmk.12.17_1. [Online]. Available: http://link.springer.com/10.2333/bhmk.12.17_1 (visited on 08/26/2024).

[81] R. Stine, "An introduction to bootstrap methods: Examples and ideas," *Sociological Methods & Research*, vol. 18, no. 2, pp. 243–291, Nov. 1989, ISSN: 0049-1241, 1552-8294. DOI: 10.1177/0049124189018002003. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0049124189018002003 (visited on 08/26/2024).

[82] D. Kalajdzievski, *Scaling laws for forgetting when fine-tuning large language models*, Jan. 10, 2024. arXiv: 2401.05605[cs]. [Online]. Available: http://arxiv.org/abs/2401.05605 (visited on 09/16/2024).

[83] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, *Parameter-efficient fine-tuning for large models: A comprehensive survey*, Sep. 15, 2024. arXiv: 2403.14608[cs]. [Online]. Available: http://arxiv.org/abs/2403.14608 (visited on 09/17/2024).

[84] N. Houlsby, A. Giurgiu, S. Jastrzebski, *et al.*, *Parameter-efficient transfer learning for NLP*, Jun. 13, 2019. arXiv: 1902.00751[cs,stat]. [Online]. Available: http://arxiv.org/abs/1902.00751 (visited on 09/17/2024).

[85] B. Lester, R. Al-Rfou, and N. Constant, *The power of scale for parameter-efficient prompt tuning*, Sep. 2, 2021. arXiv: 2104.08691[cs]. [Online]. Available: http://arxiv.org/abs/2104.08691 (visited on 09/14/2024).

[86] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *LoRA: Low-rank adaptation of large language models*, Oct. 16, 2021. arXiv: 2106.09685[cs]. [Online]. Available: http://arxiv.org/abs/2106.09685 (visited on 06/27/2024).

[87] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *QLoRA: Efficient finetuning of quantized LLMs*, May 23, 2023. arXiv: 2305.14314[cs]. [Online]. Available: http://arxiv.org/abs/2305.14314 (visited on 06/27/2024).

[88] S.-Y. Liu, C.-Y. Wang, H. Yin, *et al.*, *DoRA: Weight-decomposed low-rank adaptation*, Jun. 3, 2024. arXiv: 2402.09353[cs]. [Online]. Available: http://arxiv.org/abs/2402.09353 (visited on 06/27/2024).

[89] B. Zhang, Z. Liu, C. Cherry, and O. Firat, *When scaling meets LLM finetuning: The effect of data, model and finetuning method*, Feb. 26, 2024. arXiv: 2402.17193[cs]. [Online]. Available: http://arxiv.org/abs/2402.17193 (visited on 09/16/2024).

[90] G. Hinton, O. Vinyals, and J. Dean, *Distilling the knowledge in a neural network*, Mar. 9, 2015. arXiv: 1503.02531[cs,stat]. [Online]. Available: http://arxiv.org/abs/1503.02531 (visited on 06/27/2024).

[91] Llama Team, AI @ Meta, *The llama 3 herd of models*, Jul. 23, 2024. [Online]. Available: https://ai.meta.com/research/publications/the-llama-3-herd-of-models/ (visited on 07/29/2024).

[92] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter*, Feb. 29, 2020. arXiv: 1910.01108[cs]. [Online]. Available: http://arxiv.org/abs/1910.01108 (visited on 08/20/2024).

[93] Y. Gu, L. Dong, F. Wei, and M. Huang, *MiniLLM: Knowledge distillation of large language models*, Apr. 9, 2024. arXiv: 2306.08543[cs]. [Online]. Available: http://arxiv.org/abs/2306.08543 (visited on 06/27/2024).

[94] X. Ma, G. Fang, and X. Wang, *LLM-pruner: On the structural pruning of large language models*, Sep. 27, 2023. arXiv: 2305.11627[cs]. [Online]. Available: http://arxiv.org/abs/2305.11627 (visited on 06/27/2024).

[95] T. Jiang, D. Wang, F. Zhuang, R. Xie, and F. Xia, *Pruning pre-trained language models without fine-tuning*, May 16, 2023. arXiv: 2210.06210[cs]. [Online]. Available: http://arxiv.org/abs/2210.06210 (visited on 08/20/2024).

[96] vLLM Team. "vLLM documentation." (2024), [Online]. Available: https://docs.vllm.ai/en/stable/ (visited on 06/28/2024).

[97] Artem Dinaburg. "Understanding apple's on-device and server foundation models release," Trail of Bits Blog. (Jun. 14, 2024), [Online]. Available: https://blog.trailofbits.com/2024/06/14/understanding-apples-on-device-and-server-foundations-model-release/ (visited on 07/29/2024).

[98] Apple. "Introducing apple's on-device and server foundation models," Apple - Machine Learning Research. (Jun. 10, 2024), [Online]. Available: https://machinelearning.apple.com/research/introducing-apple-foundation-models (visited on 07/29/2024).

[99] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, *Unleashing the potential of prompt engineering in large language models: A comprehensive review*, Sep. 5, 2024. arXiv: 2310.14735[cs]. [Online]. Available: http://arxiv.org/abs/2310.14735 (visited on 09/12/2024).

[100] Anthropic. "Be clear, direct, and detailed - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags (visited on 07/29/2024).

[101] Mistral. "Mistral prompting capabilities," Mistral Docs. (2024), [Online]. Available: https://docs.mistral.ai/guides/prompting_capabilities/ (visited on 07/29/2024).

[102] Anthropic. "Use XML tags to structure your prompts - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags (visited on 07/29/2024).

[103] Anthropic. "Giving claude a role with a system prompt - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags (visited on 07/29/2024).

[104] S. Yao, D. Yu, J. Zhao, *et al.*, *Tree of thoughts: Deliberate problem solving with large language models*, Dec. 3, 2023. arXiv: 2305.10601[cs]. [Online]. Available: http://arxiv.org/abs/2305.10601 (visited on 09/12/2024).

[105] Anthropic. "Prefill claude's response for greater output control - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/prefill-claudes-response#example-maintaining-character-with-role-prompting (visited on 07/29/2024).

[106] Anthropic. "Use examples (multishot prompting) to guide claude's behavior - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags (visited on 07/29/2024).

[107] B. Paranjape, S. Lundberg, S. Singh, H. Hajishirzi, L. Zettlemoyer, and M. T. Ribeiro, *ART: Automatic multi-step reasoning and tool-use for large language models*, Mar. 15, 2023. arXiv: 2303.09014[cs]. [Online]. Available: http://arxiv.org/abs/2303.09014 (visited on 09/12/2024).

[108] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, *Large language models are zero-shot reasoners*, Jan. 29, 2023. arXiv: 2205.11916[cs]. [Online]. Available: http://arxiv.org/abs/2205.11916 (visited on 09/12/2024).

[109] Anthropic. "Let claude think (chain of thought prompting) to increase performance - prompt engineering," Anthropic Developer Documentation. (2024), [Online]. Available: https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/use-xml-tags (visited on 07/29/2024).

[110] O. Yoran, T. Wolfson, B. Bogin, U. Katz, D. Deutch, and J. Berant, *Answering questions by meta-reasoning over multiple chains of thought*, Aug. 2, 2024. arXiv: 2304.13007[cs]. [Online]. Available: http://arxiv.org/abs/2304.13007 (visited on 09/12/2024).

[111] D. Zhou, N. Schärli, L. Hou, *et al.*, *Least-to-most prompting enables complex reasoning in large language models*, Apr. 16, 2023. arXiv: 2205.10625[cs]. [Online]. Available: http://arxiv.org/abs/2205.10625 (visited on 09/12/2024).

[112] L. Yang, Z. Yu, T. Zhang, *et al.*, *Buffer of thoughts: Thought-augmented reasoning with large language models*, Jun. 6, 2024. arXiv: 2406.04271[cs]. [Online]. Available: http://arxiv.org/abs/2406.04271 (visited on 09/12/2024).

[113] A. Srivastava, A. Rastogi, A. Rao, *et al.*, *Beyond the imitation game: Quantifying and extrapolating the capabilities of language models*, Jun. 12, 2023. arXiv: 2206.04615[cs, stat]. [Online]. Available: http://arxiv.org/abs/2206.04615 (visited on 09/12/2024).

[114] X. Wang and D. Zhou, *Chain-of-thought reasoning without prompting*, May 23, 2024. arXiv: 2402.10200[cs]. [Online]. Available: http://arxiv.org/abs/2402.10200 (visited on 09/12/2024).

[115] A. Ajith, C. Pan, M. Xia, A. Deshpande, and K. Narasimhan, *InstructEval: Systematic evaluation of instruction selection methods*, Jul. 16, 2023. arXiv: 2307.00259[cs]. [Online]. Available: http://arxiv.org/abs/2307.00259 (visited on 09/12/2024).

[116] Y. Qin, S. Liang, Y. Ye, *et al.*, *ToolLLM: Facilitating large language models to master 16000+ real-world APIs*, Oct. 3, 2023. arXiv: 2307.16789[cs]. [Online]. Available: http://arxiv.org/abs/2307.16789 (visited on 09/12/2024).

[117] S. Huang, W. Zhong, J. Lu, *et al.*, *Planning, creation, usage: Benchmarking LLMs for comprehensive tool utilization in real-world complex scenarios*, Jun. 3, 2024. arXiv: 2401.17167[cs]. [Online]. Available: http://arxiv.org/abs/2401.17167 (visited on 09/12/2024).

[118] T. Schick, J. Dwivedi-Yu, R. Dessì, *et al.*, *Toolformer: Language models can teach themselves to use tools*, Feb. 9, 2023. arXiv: 2302.04761[cs]. [Online]. Available: http://arxiv.org/abs/2302.04761 (visited on 09/12/2024).

[119] OpenAI. "Function calling," OpenAI Developer Documentation. (2024), [Online]. Available: https://platform.openai.com/docs/guides/function-calling (visited on 07/29/2024).

[120] Dzmitry Bahdanau. "The FLOPs calculus of language model training," Medium. (Sep. 1, 2022), [Online]. Available: https://medium.com/@dzmitrybahdanau/the-flops-calculus-of-language-model-training-3b19c1f025e4 (visited on 05/15/2024).

[121] Adam Casson. "Transformer FLOPs," Adam Casson Blog. (May 16, 2023), [Online]. Available: https://www.adamcasson.com/posts/transformer-flops#user-content-fnref-remat (visited on 05/15/2024).

[122] A. Chowdhery, S. Narang, J. Devlin, *et al.*, *PaLM: Scaling language modeling with pathways*, Oct. 5, 2022. arXiv: 2204.02311[cs]. [Online]. Available: http://arxiv.org/abs/2204.02311 (visited on 09/15/2024).

[123] NVidia, *NVIDIA a100 tensor core GPU architecture*, 2020. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf (visited on 07/01/2024).

[124] NVidia, *Architecture - NVIDIA h100 tensor core GPU*, 2023. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper?ncid=no-ncid (visited on 07/01/2024).

[125] L. Biewald, *Experiment tracking with weights and biases*, 2020. [Online]. Available: https://www.wandb.com/.

[126] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6 (visited on 06/28/2024).

[127] NVidia, *L4 tensor core GPU - datasheet*, 2023. [Online]. Available: https://nvdam.widen.net/s/rvq98gbwsw/l4-datasheet-2595652 (visited on 07/05/2024).

[128] W. Kwon, Z. Li, S. Zhuang, *et al.*, "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, Koblenz Germany: ACM, Oct. 23, 2023, pp. 611–626. DOI: 10.1145/3600006.3613165. [Online]. Available: https://dl.acm.org/doi/10.1145/3600006.3613165 (visited on 06/28/2024).

[129] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, *Towards the systematic reporting of the energy and carbon footprints of machine learning*, Nov. 29, 2022. arXiv: 2002.05651[cs]. [Online]. Available: http://arxiv.org/abs/2002.05651 (visited on 06/27/2024).

[130] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, *Data movement is all you need: A case study on optimizing transformers*, Nov. 8, 2021. arXiv: 2007.00072[cs,stat]. [Online]. Available: http://arxiv.org/abs/2007.00072 (visited on 06/27/2024).

[131] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, *Green AI*, Aug. 13, 2019. arXiv: 1907.10597[cs,stat]. [Online]. Available: http://arxiv.org/abs/1907.10597 (visited on 06/27/2024).

[132] Y. Sun, L. Dong, Y. Zhu, *et al.*, *You only cache once: Decoder-decoder architectures for language models*, May 9, 2024. arXiv: 2405.05254[cs]. [Online]. Available: http://arxiv.org/abs/2405.05254 (visited on 06/27/2024).

[133] J. Dodge, T. Prewitt, R. T. D. Combes, *et al.*, *Measuring the carbon intensity of AI in cloud instances*, Jun. 10, 2022. arXiv: 2206.05229[cs]. [Online]. Available: http://arxiv.org/abs/2206.05229 (visited on 06/27/2024).

[134] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, *Toward sustainable GenAI using generation directives for carbon-friendly large language model inference*, Mar. 19, 2024. arXiv: 2403.12900[cs]. [Online]. Available: http://arxiv.org/abs/2403.12900 (visited on 06/27/2024).

[135] D. Foley and J. Danskin, "Ultra-performance pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, Mar. 2017, ISSN: 0272-1732, 1937-4143. DOI: 10.1109/MM.2017.37. [Online]. Available: https://ieeexplore.ieee.org/document/7924274/ (visited on 07/01/2024).

[136] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the carbon impact of generative AI inference (today and in 2035)," in *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, Boston MA USA: ACM, Jul. 9, 2023, pp. 1–7. DOI: 10.1145/3604930.3605705. [Online]. Available: https://dl.acm.org/doi/10.1145/3604930.3605705 (visited on 06/27/2024).

[137] NVidia. "TensorRT github repository." (2024), [Online]. Available: https://github.com/NVIDIA/TensorRT (visited on 07/02/2024).

[138] Microsoft. "DeepSpeed github repository." (2024), [Online]. Available: https://github.com/microsoft/DeepSpeed (visited on 07/02/2024).

[139] Georgi Gerganov and community. "Llama.cpp github repository." (2024), [Online]. Available: https://github.com/ggerganov/llama.cpp (visited on 07/02/2024).

[140] vLLM Team. "vLLM github repository." (2024), [Online]. Available: https://github.com/vllm-project/vllm (visited on 06/28/2024).

[141] Aishwarya Goel and Rajdeep Borgohain. "Exploring LLMs speed benchmarks: Independent analysis," inferless. (Mar. 19, 2024), [Online]. Available: https://www.inferless.com/learn/exploring-llms-speed-benchmarks-independent-analysis (visited on 06/28/2024).

[142] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, *et al.* "vLLM: Easy, fast, and cheap LLM serving with PagedAttention," vLLM Blog. (Jun. 20, 2023), [Online]. Available: https://blog.vllm.ai/2023/06/20/vllm.html (visited on 06/28/2024).

[143] vLLM Team. "Notes on vLLM vs. DeepSpeed.," vLLM Blog. (Nov. 14, 2023), [Online]. Available: https://blog.vllm.ai/2023/11/14/notes-vllm-vs-deepspeed.html (visited on 06/28/2024).

[144] D. Narayanan, M. Shoeybi, J. Casper, *et al.*, *Efficient large-scale language model training on GPU clusters using megatron-LM*, Version Number: 5, 2021. DOI: 10.48550/ARXIV.2104.04473. [Online]. Available: https://arxiv.org/abs/2104.04473 (visited on 07/05/2024).

[145] HuggingFace. "Model parallelism," Transformers Documentation. (2024), [Online]. Available: https://huggingface.co/docs/transformers/v4.42.0/en/perf_train_gpu_many#tensor-parallelism (visited on 07/05/2024).

[146] vLLM Team. "Automatic prefix caching," vLLM Documentation. (2024), [Online]. Available: https://docs.vllm.ai/en/latest/automatic_prefix_caching/apc.html (visited on 07/05/2024).

[147] L. Tianhua, Z. Hongfeng, C. Guiran, and Z. Chuansheng, "The design and implementation of zero-copy for linux," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, Kaohsuing, Taiwan: IEEE, Nov. 2008, pp. 121–126, ISBN: 978-0-7695-3382-7. DOI: 10.1109/ISDA.2008.102. [Online]. Available: http://ieeexplore.ieee.org/document/4696190/ (visited on 07/05/2024).

[148] NVidia. "GPUDirect," NVidia Developer. (2024), [Online]. Available: https://developer.nvidia.com/gpudirect (visited on 07/05/2024).

[149] NVidia, *Datasheet - NVIDIA h100 tensor core GPU*, 2023. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/nvidia-tensor-core-gpu-datasheet?ncid=no-ncid (visited on 07/05/2024).

[150] Vinh Nguyen, Michael Carilli, Sukru Burc Eryilmaz, *et al.* "CUDA-graphs," PyTorch Blog. (Oct. 26, 2021), [Online]. Available: https://pytorch.org/blog/accelerating-pytorch-with-cuda-graphs/ (visited on 07/05/2024).

[151] Alan Gray. "CUDA-graphs," NVidia Developer. (May 9, 2019), [Online]. Available: https://developer.nvidia.com/blog/cuda-graphs/ (visited on 07/05/2024).

[152] S. Systems, *SN40l RDU AI chip: The GPU alternative*, 2024. [Online]. Available: https://sambanova.ai/technology/sn40l-rdu-ai-chip.

[153] C. Systems, *Cerebras wafer-scale engine (WSE-3): AI processor overview*, 2024. [Online]. Available: https://cerebras.ai/product-chip/.

[154] A. Analysis, *Llama 3.1 instruct 70b: API provider performance benchmarking & price analysis*, 2024. [Online]. Available: https://artificialanalysis.ai/models/llama-3-1-instruct-70b/providers.

[155] I. Groq, *GroqThoughts power paper 2024*, 2024. [Online]. Available: https://groq.com/wp-content/uploads/2024/07/GroqThoughts_PowerPaper_2024.pdf.

[156] Tianle Li, Anastasios Angelopoulos, and Wei-Lin Chiang. "Does style matter? disentangling style and substance in chatbot arena," LMSYS Chatbot Arena. (Aug. 29, 2024), [Online]. Available: https://lmsys.org/blog/2024-08-28-style-control/ (visited on 09/10/2024).

[157] P. Li, J. Yang, M. A. Islam, and S. Ren, *Making AI less "thirsty": Uncovering and addressing the secret water footprint of AI models*, Oct. 29, 2023. arXiv: 2304.03271[cs]. [Online]. Available: http://arxiv.org/abs/2304.03271 (visited on 06/27/2024).

[158] D. Patterson, J. Gonzalez, U. Holzle, *et al.*, "The carbon footprint of machine learning training will plateau, then shrink," *Computer*, vol. 55, no. 7, pp. 18–28, Jul. 2022, ISSN: 0018-9162, 1558-0814. DOI: 10.1109/MC.2022.3148714. [Online]. Available: https://ieeexplore.ieee.org/document/9810097/ (visited on 06/27/2024).

[159] S. Ghaffary, "OpenAI pitched white house on unprecedented data center buildout," *Bloomberg News*, 2024. [Online]. Available: https://www.bloomberg.com/news/articles/2024-09-24/openai-pitched-white-house-on-unprecedented-data-center-buildout?embedded-checkout=true.

A SYSTEM PROMPT AND PROMPT TEMPLATE FOR THE ARENA-HARD-AUTO BENCHMARK

<|System Prompt|>

Please act as an impartial judge and evaluate the quality of the responses provided by two AI assistants to the user prompt. You will be given assistant A's answer and assistant B's answer. Your job is to evaluate which assistant's answer is better.

Begin your evaluation by generating your own answer to the prompt. You must provide your answers before judging any answers. When evaluating the assistants' answers, compare both assistants' answers with your answer. You must identify and correct any mistakes or inaccurate information.

Then consider if the assistant's answers are helpful, relevant, and concise. Helpful means the answer correctly responds to the prompt or follows the instructions. Note when user prompt has any ambiguity or more than one interpretation, it is more helpful and appropriate to ask for clarifications or more information from the user than providing an answer based on assumptions. Relevant means all parts of the response closely connect or are appropriate to what is being asked. Concise means the response is clear and not verbose or excessive.

Then consider the creativity and novelty of the assistant's answers when needed. Finally, identify any missing important information in the assistants' answers that would be beneficial to include when responding to the user prompt.

After providing your explanation, you must output only one of the following choices as your final verdict with a label:

Assistant A is significantly better: [[A≫B]]

Assistant A is slightly better: [[A>B]]

Tie, relatively the same: [[A=B]]

Assistant B is slightly better: [[B>A]]

Assistant B is significantly better: [[B≫A]]

Example output: "My final verdict is tie: [[A=B]]".

<|User Prompt|>

{BENCHMARK-QUESTION}

<|The Start of Assistant A's Answer|>

{ANSWER-1}

<|The Start of Assistant B's Answer|>

{ANSWER-2}

B PROMPT FOR THE GUIDANCE CREATION OF THE ARENA-HARD-AUTO BENCHMARK

<|System Prompt|>

You are a sophisticated AI-Assistant there to help users solve tasks efficiently and accurately.
Based on the following question and ideal answer, provide concise and structured guidance to help solve the task.
Do not reveal the solution directly but instead offer the necessary context, subtle hints, and a brief step-by-step plan that encourages thoughtful consideration without providing the answer outright.

<|Question|>

{QUESTION}

<|Ideal Answer|>

{IDEAL_ANSWER}

<|User Prompt|>

Please generate the following:

1. **Contextual Information**: Provide relevant background information necessary to understand the task without giving away the solution.

2. **Hints and Tips**: Offer subtle hints or tips that can guide the user toward the solution, without stating it explicitly.

3. **Step-by-Step Plan**: Provide a concise sequence of steps that the user can follow to approach and solve the task. Each step should lead logically to the next, but be brief and focused on guiding rather than solving.

4. **Encouragement for Reasoning**: Suggest that the user should consider each step carefully and think critically about each and every step.

5. **Additional Considerations**: Include any extra information that might be useful, such as common pitfalls to avoid or alternative methods to consider, but keep it brief.