MSc Projects — School of Computer Science
2023-24

Prototype Framework for Energy Profiling of Transformers

Mohamad Abdallah - MSc Data Science

September 2, 2024

Abstract

The widespread adoption of transformer-based language models, such as GPT, BERT, and LLAMA, is poised to revolutionize various industries and research fields, presenting opportunities and challenges. A prominent concern is the substantial energy consumption associated with these models, which poses a significant hurdle in global decarbonization efforts. Some studies suggest that transformers-powered AI models could account for up to 50% of all data center processing activities worldwide. Consequently, optimizing these systems for enhanced energy efficiency is crucial. However, comprehensive literature on the energy demands of such models still needs to be expanded. This study introduces a prototype framework to evaluate the energy consumption of various transformer models, including BERT and GPT-2, also called Large Language Models (LLMs). The development of this framework required a thorough examination of multiple stages, which are detailed in this report. Using the General Language Understanding Evaluation (GLUE) benchmark, we tested subsets with specific NLP tasks and found a nonlinear relationship between performance metrics (e.g., Accuracy and Matthews Correlation Coefficient) and energy consumption. While different query lengths showed no significant impact on energy use, substantial variations were observed, indicating the need for further research, particularly in testing query complexity.

Keywords : *Transformer-based models; GPT; BERT; Energy consumption; Energy efficiency; Framework prototype; GLUE benchmark; NLP; Performance metrics; LLM*

Abbreviations and Acronyms : *General Language Understanding Evaluation (GLUE); World Machine Translation (WMT); Legal Entity Database for General AI Research (LEDGAR); Corpus of Linguistic Acceptability (CoLA); Stanford Sentiment Treebank-2 (SST-2); Matthews Correlation Coefficient (MCC); Natural Language Processing (NLP); Support Vector Machine (SVM); Recurrent Neural Network (RNN); Long Short-Term Memory (LSTM); Generative Pretrained Transformer (GPT); Bidirectional Encoder Representations from Transformers (BERT); Language Models for the Advancement of Machine Learning and Artificial Intelligence (LLAMA); Translation Edit Rate (TER); NVIDIA Management Library (NVML); Central Processing Unit (CPU); Graphics Processing Unit (GPU); Random Access Memory (RAM); Kilowatt-hour (kWh).*

Acknowledgment

I would like to express my deepest gratitude to my supervisor, Dr. Rami Bahsoon, for his invaluable guidance, encouragement, and support throughout this project. I am also immensely grateful to Dr. Vincent Rahli for his insightful feedback and advice, which greatly contributed to the success of this work.

I would like to extend my heartfelt thanks to my family and friends for their unwavering support and understanding during this journey. A special mention goes to my wife, whose love, patience, and constant encouragement have been my greatest source of strength.

Thank you all for your belief in me and for being an integral part of this achievement.

List of Figures

List of Tables

Contents

1 Introduction

ChatGPT gets approximately 600 million visits every month [2], a massive demand for a product launched in November of 2022. The explosion of LLMs across industries compels researchers worldwide to innovate new ways of enhancing these models. The cost of energy associated with this has become a growing concern. The report begins with the motivation to study the energy profile of LLMs, followed by a short description of the issue, broken down in the energy dilemma Section 1.2. The report introduces a prototype framework for LLM energy profiling to help make better energy-efficient decisions. We move on to the scope and limitations of this project, followed by the methodology used. This section ends with the contributions toward LLM energy analysis, a description of the overall report structure, and a short legal compliance passage.

1.1 Background and Motivation

With one publication [1], the world turned its attention from the RNNs and LSTMs as the most popular choices for NLP tasks to transformer-based Large Language Models (LLMs) in 2022. Transformer-based models have become indispensable tools across various sectors, primarily due to the ingenious multiheaded attention mechanism discussed in Section 2.1.2 and outlined in the 2017 paper "Attention Is All You Need" by Google and the University of Toronto [1], which invented the Transformer architecture. Previous models handled words in sentences sequentially for NLP tasks and transformers fed sentences in parallel. This simple yet revolutionary concept quickly took the base transformer model to the top of the leaderboard, overtaking the runtimes and performances of previous models [1]. The growing demand for transformers underscores a growing symbiotic relationship between AI and humans, fundamentally reshaping how we interact with technology and operate within industries.

The architecture is both simple and complex in nature. It is reproducible and can be decomposed and reconstructed into various models, discussed in Section 3.4. Based on the first architecture, over 800,000 models were designed and can be accessed in the Hugging Face hub [3], all tasked to solve problems within NLP, computer vision, audio, and more. At the time of this report, all these developments had been accomplished in a couple of years. Research [4, 5] on the energy consumption of models found that

training a large-scale model can take up as much energy in megawatt-hours as an average American home annually. Although real-world application or 'inference' discussed in section 2.1.3 is less energy demanding, it requires substantial energy. This is becoming increasingly concerning to data centers, which have to optimize their systems for this new demand,future demand and for the environmental impact these can have. With this new world to explore, many new ways exist to optimize and make transformer models more energy-efficient while maintaining performance [6, 7, 8, 9]. Now more than ever, there is an urgency to study and profile the energy consumption of transformers. This report suggests a framework to research and analyze energy consumption and efficiencies. The insights provided could assist users and stakeholders in making better decisions and saving time, money, and energy. Finally, the research [10] done throughout this project furthers my knowledge of LLMs and transformers, has given me a solid foundation in transformers, and will further my studies within this field.

1.2 Problem Statement & Research Objective

The Energy Dilemma The increasing energy demands associated with LLMs present significant challenges, particularly for data centers. While offering substantial benefits, machine learning and deep learning models also pose environmental risks, potentially contributing up to 23% of global data center energy consumption and, in worst-case scenarios, up to 51% by 2030 [11, 12]. Understanding energy consumption patterns in transformer models is thus crucial for reducing carbon footprints, cutting costs, building robust infrastructure, and optimizing operations. This understanding can lead to a more sustainable future, where energy-efficient models are prioritized over marginal performance gains [13].

Energy Profiling Framework Inspired by the findings of Rigutini et al. (2024) [13] and discussed in Section 2.4, this report proposes a step-by-step process to replicate the findings and further the research by providing a prototype framework. The framework's goal is to compare the energy consumption of various transformer models and provide insights into relevant factors affecting energy consumption. The goals can be broken down into four milestones, each a step toward the prototype:

- Research & Development: Acquiring the knowledge necessary to create the framework and test various models for speed and memory usage for

post-processing.

- Stage 1: Create definitions and visualizations to profile the energy consumption of transformers with performance metrics
- Stage 2: Create a schema to analyze the impact of query length and complexity on the energy consumption of transformers
- Stage 3: Decompose transformer architecture to study the impact of each component on the energy consumption

1.3 Scope & Limitations

The project focuses on developing a unique NLP task-specific framework designed to profile transformer energy consumption. This replicable contribution could lay the groundwork for a fully automated framework. Utilizing the Hugging Face hub, any model and dataset can be paired to analyze their performances against the suggested criteria. My framework suggests a list of criteria to identify a well-performing model with a greater scope for its efficiency and accuracy.

Completing such a framework requires more time than this thesis project's provided time. There are over 880,000 models and almost 200,000 datasets with 300 metrics to apply; the volume of research necessary to create a framework for all tasks would require more labor, time, and resources; access to heavy-duty hardware, at least 200GB of GPU capability compared to the 6GB used for this report. Nevertheless, experimentations performed successfully, and the results match the ones outlined in [13, 14] with additional metrics and insights, making this report a novelty.

Accurately measuring the energy consumption was another challenge. To do so, full administrative access to the device is required, limiting usage of this hardware in Section 3.1. The function for energy measurement in Python reads the energy of the entire device (CPU, RAM, and GPU), including the code; this could reflect poorly on the results if any background processes were to fluctuate in demand. Several solutions were proposed for more accurate measurements. During code execution, the model and data were moved to the dedicated GPU to perform all tasks exclusively on it; it ensures no interference with other tasks on other devices. This solution also avoids having three devices monitoring the energy consumption, accumulating the inherent inaccuracies from the device, and merging the results. Most background

processes on the device, performed on the GPU, were paused temporarily during modeling, improving energy consumption accuracy. Let's now look into the methodical steps to achieve a prototype framework.

1.4 Methodology Overview: Prototype Framework & Research Stages

The methodology (Section 3) begins with the device and Python libraries used, followed by the framework design **??**. The figure outlines the flow from task to output. Producing this framework requires two phases: transformer research and framework production, which involves a three-stage analysis. The workload was 70% on research, including Python implementations, and 30% on product development (the final code). The research phase provided the tools and knowledge necessary to implement the project. This included loading subsets and models, data filters, map functions, model training, benchmarking, processing, tuning, and evaluation. The primary objective of the research phase was selecting models with similar accuracies and suitable datasets required for this project, discussed in 3.3.2.

With the selected models, the production phase starts. Stage 1 of the production phase begins with the aim to replicate the findings in the research done by Rigutini et al. [13] with additional metrics. Stage 1 analyzes these models using widely accepted metrics such as F1 score or accuracy and diagnoses the resulting metrics alongside the energy consumption and the additional metric. The prototype uses a binary classification dataset as a case study, and the suggested metric was the Mathews Correlation Coefficient (MCC), explained in section 2.2.1. Stage 2 examines the effect of each query while training the transformer by comparing the sequence length to its energy consumption. Despite having yet to produce Stage 3, the research was completed. The goal is to dissect the transformer architecture and study the energy consumption along the pipeline **??**, giving us insight into the architectural properties of various models. The analysis shows promising results for proceeding with this stage and further.

1.5 Contribution: LLM Energy Analysis

The scope of this project is LLM energy analysis. The research and design of a framework provided the following contributions to this field:

1. Proving that results can be replicated using basic instruments and provided additional context to performance
2. Showcased that batch size and query complexity affect energy consumption but not query length
3. Create the basis for a framework to analyze the energy consumption of various stages in the transformer pipeline

This report outlines the development of such a framework and presents conclusions emphasizing the need to consider various input aspects and model design in future analyses for the best-performing models while considering energy efficiency. The findings also suggest several avenues for future research, particularly concerning the energy consumption of different stages within the transformer architecture and other potential areas for optimization by comparing tuning techniques.

1.6 Regulatory & Ethical Considerations

This project adheres to all relevant legal standards, including full compliance with GDPR. No sensitive data is used, ensuring that all research and analysis are conducted within the bounds of current data protection regulations.

1.7 Report Structure

Now you've read a quick introduction on this and that. The next sections are as follows:

- **Introduction**: Introduces the motivation, problem statement, and research objectives related to the energy consumption of transformer-based models, along with the scope, limitations, and methodology overview.
- **Literature Review**: Provides a comprehensive review of the theoretical background of transformers, key metrics, related work, current trends, and identified gaps in the literature.
- **Methodology**: Details the research and framework development process, including the specific steps taken to profile and analyze the energy consumption of various transformer models.

- **Results and Discussion**: Presents the findings from the experiments conducted, analyzing the relationship between model performance and energy consumption, and discusses the implications of these results.
- **Conclusion and Future Work**: Summarizes the key contributions of the study, discusses the broader impact of the findings, and suggests directions for future research in energy-efficient AI model development.

2 Literature Review

This comprehensive literature review begins the theoretical background by examining the transformer architecture, the main components, and the primary pipeline. We follow this with the metrics used for this project. The dataset overview and the chosen subsets required for the project are provided in Section 2.2.3. Following this, we explore related work that laid the foundation for this report, focusing on key findings and methodologies. The background section discusses the history of NLP and the contributions made by the transformers. This section also covers the popularization and its impacts on energy demand, and some [13, 14, 6, 7, 8] have taken steps towards meeting these consequences through comparative studies or tuning techniques. Current trends cover techniques such as quantization [9] and distillation [7] that have emerged as solutions to enhance transformer performance and energy efficiency. Still, as this is still a very new industry (2022), the research has just begun, and finally, we have identified gaps in the literature.

2.1 Theoretical Background: The Transformer

The paper "Attention is All You Need" [1] introduced the transformer architecture 2.1. The architecture comprises an encoder and decoder stacks; both behave similarly with multi-headed attention mechanisms, feed-forward neural networks for the output, and normalization stages between each step. The main difference is that the decoder has an additional masked multi-head attention mechanism and is fed the output from the encoder. The masked multi-headed attention mechanism prevents the model of 'cheating' in tasks like translation where future words are unknown; further explanation would be out of the scope of this report. The attention mechanism enhances the

model's capability to capture dependencies and context in the input sequence, especially with long text. We consider the primary transformer [1] pipeline to have four stages. First is tokenization, where the sequence of words converts into a vector of input IDs and the positional encoding for each word in a sequence. Second and third are the encoder and decoder stages, and lastly, the output stage converts the encodings using a softmax function. Three main concepts are considered significant advancements in transformers: positional encodings, attention mechanism, and the self-attention mechanism. We explain each one in the following sections.

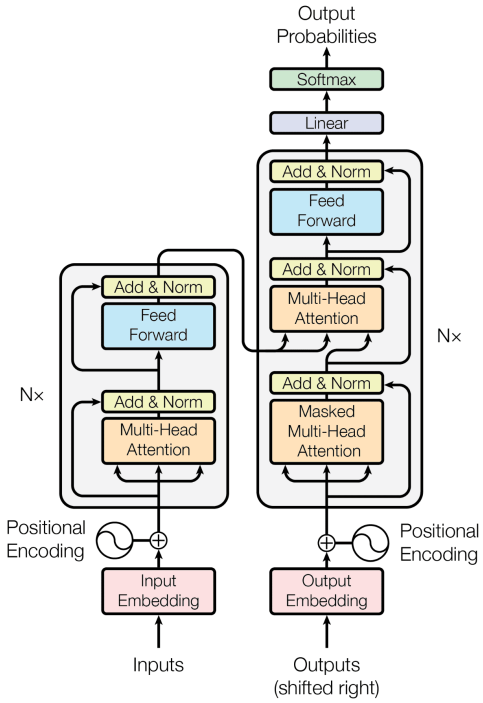


Figure 2.1: Transformer diagram as seen in Vaswani et al. [1]. The encoder represents the block to the left and decoder to the right.

2.1.1 Positional Encoding

Current transformers feed the input embeddings in parallel, while previous models like RNNs and LSTMs do it sequentially. Hence, unlike these previous models, the transformer lacks the words' inherent order. As a solution

introduced in [1], positional encodings are joined to the input embeddings as an additional number tagged to each word in a sequence. This method distinguishes between positions of words within the sequence and enhances the model's understanding of context no matter the distance. Equations 3.1 and 3.2 [1] create positional encodings for each word, ranging from [-1,1] as floating points. The main advantages include:

- Periodic Structure: Using cosine and sine functions means they are periodic and allows the model to find patterns in the sequence.
- Dimensionality: The equation's inclusion of dimensionality ensures that each tagged number can be directly assigned to the input embedding.
- Generalization to Sequence Lengths: A continuous function represents any floating point between -1 and 1.

The following equations 2.1 and 2.2 represent the positional encoding in even and odd dimensions, respectively.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{2.2}$$

2.1.2 Attention & Self Attention Mechanism

Figure 2.2: Representation of the attention score when each word is compared to the other and itself, the darker green indicates higher scores

The attention mechanism assigns the attention score, computed using Equation 3.3, to each word in the process. The score can be explained as a set of weighted vectors that assign each word to each other word and itself, giving a certain level of attention to each word and, hence, better context. The attention mechanism was initially introduced in 2014 by Bahdanau et al. [11] to focus on relevant input parts through basic alignment models. Since then, they have evolved into multi-layered complex self-attention mechanisms, which have become integral to transformer architecture. The self-attention mechanism allowed the model to capture the importance of each word in a sequence simultaneously as seen in 3.3. The mechanism is essential behind tasks requiring deep contextual understanding, such as translation, where context from the entire sentence is crucial.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.3}$$

2.2 Transformer Metrics, Lifecyle & Benchmark

2.2.1 ML & Transformer Metrics

Most papers discussed mention accuracy, F1 score, or Blue. The F1 score and accuracy belong to a more general class of metrics for machine learning, while Blue is based on language models. The MCC score is the most reliable for NLP binary classification tasks, with a better understanding of predictions than ground truth. However, first, a quick note on energy consumption units. Joules (J) is the standard unit for measuring energy in the International System of Units (SI). Kilowatt-hours (kWh) are commonly used worldwide for larger quantities. One kilowatt-hour is equivalent to 3.6 million joules (3.6 MJ).
The F1 score, deployable from Scikit-Learn (Python library), also has a micro (mF1) and macro (MF1) versions of it. The score is a metric that combines precision and recall into a single number. The F1 score can be computed using the equation (2.4) below:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.4}$$

Precision measures the accuracy of the positive predictions made by the

model. It tells us how many of the instances classified as positive are actually positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

Recall (or Sensitivity) measures the ability of the model to find all the actual positive instances. It tells us how many of the actual positives were correctly identified by the model.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

Micro averaging calculates metrics globally by counting the total true positives, false negatives, and false positives across all classes, and then computes precision, recall, and F1 score from these aggregated counts. Macro averaging calculates the metric for each class independently and then takes the average of these metrics. It treats all classes equally, regardless of their size. Before computing the Blue score (Bilingual Evaluation Understudy), equation 2.6, we have to first look into three concepts:

1. Precision (Equation 2.7): It measures how many n-grams in the generated text appear in the reference texts.

2. N-grams (Equation 2.8): BLEU compares sequences of n items (n-grams) between generated and reference texts to assess quality.

3. Brevity Penalty (Equation 2.9): To avoid favoring shorter texts, BLEU applies a penalty if the generated text is shorter than the reference texts.

$$P_n = \frac{\text{Number of n-grams in both generated and reference texts}}{\text{Number of n-gras in generated texts}} \tag{2.7}$$

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} \frac{1}{N} \log P_n\right) \tag{2.8}$$

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-\frac{r}{c}} & \text{if } c \leq r \end{cases} \tag{2.9}$$

The accuracy score from Scikit-Learn is a widely used metric to measure the performance of multi-label classification problems. As shown in Equation 3.4, the score ranges from [0;1], with 1 being a perfect score.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{2.10}$$

Mathews Correlation Coefficient (MCC), from Scikit-Learn, is a more robust metric for binary classification problems and is well designed to deal with unbalanced datasets. As shown in Equation 3.5, the score ranged from [-1;1], with 1 being the best, 0 acting similarly to a random function, and -1 worse than a random function. This score is introduced into our framework to provide a well-balanced understanding of the true performance of a certain dataset on a model.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.11}$$

2.2.2 Training, Evaluation & Inference of Transformers

Research has identified two primary categories of investigation concerning energy usage in large language models (LLMs): energy consumption during training and testing and energy consumption during inference. Training and testing LLMs for evaluation necessitates significant computational energy. With LLMs increasingly being deployed in the consumer market, the inference associated with the new data requires substantial resources.

Benchmark datasets are composed of subsets, each containing a training, validation, and testing set. The goal of training is to teach the model patterns within the data. It minimizes the loss function; the closer to 0, the better. At that stage, the data is labeled and trained over multiple epochs, a run of the model on the entire dataset. The trained model is evaluated using the validation set; at this stage, the desired performance metrics are computed using accuracy, BLEU, F1 Score, and perplexity, which are all general to NLP tasks. Finally, we have inference utilizing the testing set; at this stage, it represents a real-world application where the model predicts new data output.

2.2.3 GLUE Benchmark

The General Language Understanding Evaluation (GLUE) [15] benchmark is a collection of 9 subsets, each with a designated task within NLP that can be used to train, validate, and test the model. GLUE benchmark covers most tasks required in NLP, making it suitable for extensive research within this domain. For a cohesive comparison across the prototype, both datasets must be under the family of binary classification; this allows for thorough analysis in different stages of the project. The following subsets were carefully selected to match the requirements of the project:

- CoLa (Corpus of Linguistic Acceptability): Binary classification checks if a sentence is grammatically correct. With 10,700 rows (sentences), this dataset belongs to one of the smaller groups and is utilized in pre-research, which provided shorter training times over the entire set.
- SST-2 (Stanford Sentiment Treebank): Binary classification determines whether a sentence is positive or negative or sentiment analysis. With 70,000 rows (sentences), this dataset belonged to one of the larger available subsets in GLUE and was selected because it has some of the largest sentences within a row, making it perfect for our stage 2 query analysis.

2.3 Background Research

2.3.1 From RNN's to Transformers

The research done by Vaswani et al. in 2017 [1], inspired by the attention mechanism developed in 2014 by Bahdanau et al. [16], brought forth a revolution in the world of AI. Taking a step back, the most popular models for solving NLP tasks include the LSTMs developed in 1997 by Hochreiter et al. [17] and RNNs in 1986 by Rumbelhart et al. [18]. Both were at the forefront of NLP tasks. Nonetheless, they had their limitations. The primary drawbacks discussed include vanishing or exploding gradients [19], the mechanism responsible for updating predictions, and the lack of contextual understanding from being fed each word sequentially. Although the LTSM architecture was introduced to mitigate the exploding or vanishing gradient and long-term dependencies, it only partially solved the problem. The transformer [1] solved these issues and many more while achieving higher performance results.

The transformer architecture solved all these issues and, most importantly, parallelized the input of sequences, making them operate much faster and more scalable; section 3.1 runs through the mechanisms responsible for all this. Vaswani et al. [1] state that their BLEU performance, a widely used translation accuracy metric, for their base model outperformed any previous model using the WMT benchmark and required less than 1/4 of the training time of any other previous model. These speeds made it feasible to use and generate excellent results with quick times, bringing models like GPT and Bert to life. Having established itself as the best, it quickly became a valuable asset across all industries, such as finance, bioinformatics, and many others, making it indispensable. The increasing demand also brings higher energy consumption and costs, quickly becoming an inevitable consequence. As this is a relatively new subject, the research is still developing and needs time to catch up with its growing popularity, especially regarding energy usage.

2.3.2 Need for Energy-Efficient Transformers

Many studies now surround different options for energy and performance efficiency. However, making it run quicker or more efficiently affects the model's accuracy over a given task. In this context, [13, 14] compares models and various benchmarks, revealing minimal performance differences relative to code complexity. While numerous studies, such as [20], have explored energy optimization techniques for large language models (LLMs), they often overlook holistic performance considerations over the more minor details, such as the type of query or the stages of the architecture responsible for this energy consumption. Other approaches, such as model size reduction through pruning [6], distillation [7], vocabulary transfer [8], and quantization [9], have also been explored. However, the lack of comprehensive model comparisons significantly underscores the need to prioritize research efforts in energy optimization. Addressing this gap would steer industry focus and guide effective strategies for improving LLM efficiency.

2.4 Related Work

Starting with the most relatable work, this paper by Rigutini et al. [13], published in 2024, is a comparative study made to analyze automatic text classification approaches in the legal domain. The LexGLUE benchmark focuses on legal NLP tasks, such as multi-class tasks. They approached the

problem using three different families of text classification models: the classical Support Vector Machine (SVM) [21], the first generation of LLMs such as BERT and LegalBERT, and recent generative models freely available such as GPT-2 and LLAMA2. The metrics used are the standard F1 score, with both micro (mF1) and macro (MF1) averaging. The energy consumption was measured using the 'codecarbon' library, and the costs in euros (€) and carbon dioxide (CO2) emissions were estimated. The experiments ran on an Xeon processor, 503 GB of RAM, and 4 NVIDIA RTX A6000 GPUs, a total of 192GB of G-RAM. There were two phases for experimentation. The first was the R&D phase, where they iteratively evaluated the model performance and energy consumption over training, validation, and testing. The models are tested on their prediction over 100 documents in the production phase. The benchmark, like all others, is composed of several subsets. To highlight significant results for the ECtHR subset, the SVM model was strikingly close with the mF1 score at 0.75 compared to BERT 0.8 and outperforming current models like LLAMA2 at 0.69. The critical insight is that BERT and LLAMA2 consumed 73.93 and 167.21 times more energy (KWh) than the SVM model. This paper indicated a drastically unfair tradeoff between energy consumption and model performance. However, distilled versions of BERT, a smaller version, on the EUR-LEX subset achieved similar performances with SVM with an mF1 of 0.74 and managed to consume only 1.91 times the energy consumed by SVM.

Similarly, the paper authored by Gultikin et al. [14] in 2024 presents a comparative analysis of transformer models and their energy consumption. The LexGLUE benchmark was used, and models included BERT, LegalBERT, and DistilBERT compared to SVM-based NLP models. The F1 score and energy-related metrics, such as the power consumption (KWh), the cost, and estimated carbon dioxide output, were measured during the experiment. The results in this paper match those of Rigutini, who had very close performance in terms of F1 score and found that a transformer-based model could use 75 times more energy than an SVM-based model. This again indicated poor tradeoffs between energy consumption and performance, although the case is better with distilBERT consuming only 3 times more. The scope of this paper is limited to basic comparative study, although it is done over the entire benchmark, such as in [13]. With the right equipment, these tests would take little time to run.

2.5 Current Transformer Optimization Techniques

Current research [13, 14, 6, 7, 8, 9, 22] suggests four main techniques currently developed to optimize models' runtime and energy efficiency while marginally compromising performance. These four include quantization, distillation, vocabulary transfer, and pruning. Other studies [23] suggest basic techniques such as adjusting model size, number of layers, parallelized attention, and vocabulary size to reduce energy consumption. The paper focused on optimizing energy efficiency and latency during inference by utilizing quantization levels and batch size. Quantization is one of the popular techniques; however, distillation has proven to be very effective. The following passages discuss the distillation technique to create smaller/distilled versions of a model to enhance performance.

BERT, made by Google and one of the most popular models, has developed a distilled version [24]. The idea behind distillation is simple. Knowledge distillation, distillation for short, is a model compression technique where the smaller model replicates the behavior of the larger one; in this case, distilBERT is the student, and BERT is the teacher. BERT comprises 12 encoder layers from the transformer, and distilled BERT has only 6. Although half the layers were removed, 12 out of 16 attention heads remained with much 1/3 of the parameters. The distilled version of BERT was reduced by 60% from the original model while successfully retaining 97% of the task success rate. The smaller size makes it more computationally feasible for personal device use.

2.6 Gaps in the Literature

The world of transformers is relatively new, with sectors quickly adopting generative AI into their networks and research pipelines, and with new technology must come new research. A noticeable gap in the literature is found, with minimal studies focused on comparing the models and their efficiencies. The Hugging Face [3] hub has over 100,000 models, 180,000 available datasets, and 270 metrics for manipulation. A comparative study is a very challenging task; the leaderboard provided by Hugging Face is one of the best, featuring a variety of metrics and filters to choose from. Still, it lacks more accessible functions to find the best-performing model according to any task and does not include the energy efficiency of a model. As we have seen from different papers, choosing the wrong model could drastically impact en-

ergy costs, making it inefficient for large-scale deployment. Sectors employing transformers should consider the energy consumption during integration. The key is developing a framework for evaluating this energy consumption, which could assist industries in making environmentally sustainable and economically analytical decisions.

3 Methodology

This section outlines the methodology employed during the research and development stage, following the structured steps taken throughout the project. It includes a breakdown of the project stages and the tests conducted. The first step involves understanding the transformer architecture and the mechanisms that drive its performance. Next, the tools, packages, metrics, and other essential components used for coding and evaluation are discussed. Once we have an overview, we will delve into the framework and the stages undertaken to develop part of the prototype. The analysis to achieve this framework comprises three main stages, each detailing various aspects that might impact energy consumption. Before delving into the transformer and analysis, let us explore the GLUE benchmark with the selected subsets, models and device specifications for this prototype.

3.1 Python Libraries & Device

'Numpy' and 'pandas' are used to manipulate and clean data frames and numerical values. The 'dataset' and 'transformers' libraries from Hugging Face provided the models and datasets for this project. The accuracy and MCC metrics are taken from Scikit-learn, a general machine library. PyTorch, used for computer vision and NLP applications, runs the transformer library and is utilized alongside PyJoules to force the model and dataset to run exclusively on the GPU. PyJoules is a toolkit that measures the energy consumption of a host machine while executing a piece of code using a decorator function. The decorator function wraps around a definition and measures the energy consumption at each call, an example of which can be seen in appendix .1.

The device specifications for this project are:

- CPU: Intel Core i7 10th gen
- RAM: DDR4 32 GB - 3200 MHz

- GPU: NVIDIA RTX 3060 - 6GB

3.2 Proposed Framework

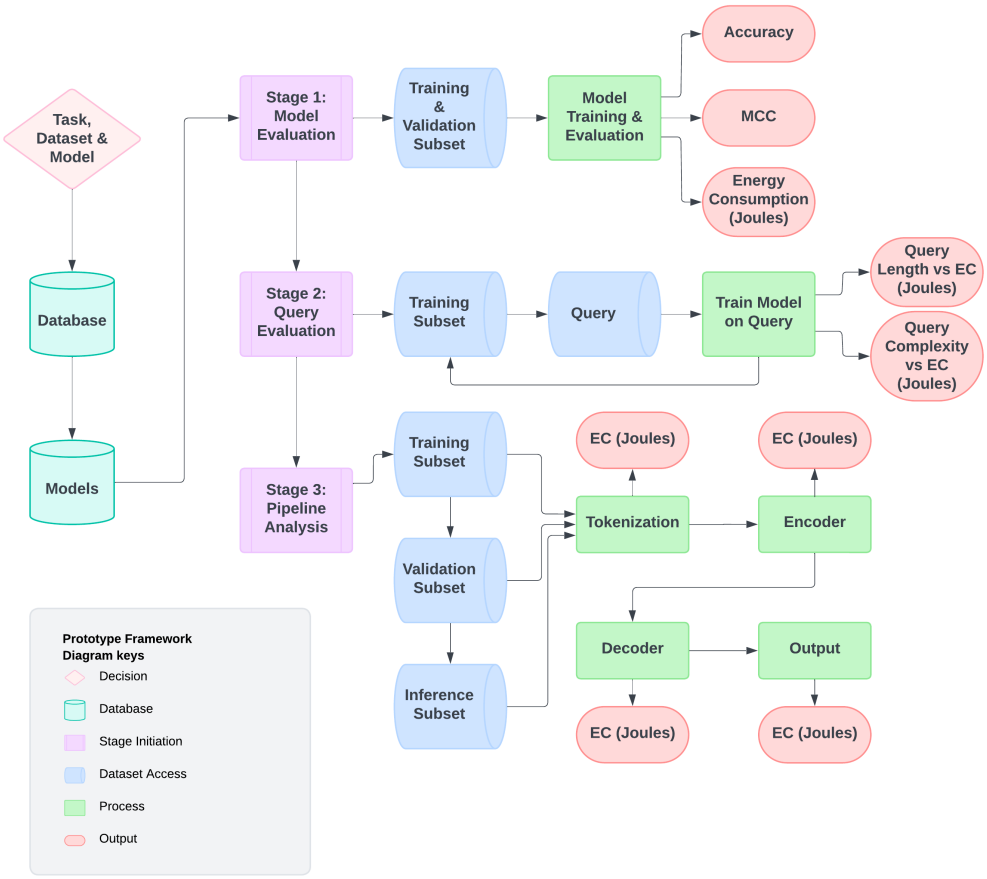


Figure 3.1: Diagram overview of the data pipeline through the different three stages

Figure 3.1 represents a general framework for analyzing a model's energy consumption and performance on NLP tasks, specifically binary classification tasks. To adopt this framework to other tasks in or outside of NLP, such as computer vision, the metrics such as MCC would have to change, but

accuracy, a general metric, can be used for all. The framework starts by selecting a task, which then provides the datasets and models appropriate to this task. The selected models and dataset then undergo a three-stage analysis.

1. Stage 1: The selected models for comparison are trained and evaluated over the entire dataset. For this report, the CoLa subset of Glue was used for this stage; the size of the dataset is small compared to others, allowing for quicker training times. As seen in section 5.2, the measurements captured compare the selected models with the metrics in this instance: the accuracy, MCC, and energy consumption in joules. The goal is to provide an overall perspective of which model would be more efficient by assessing the performance of the validation set versus the energy consumption it had to consume to achieve these results. The model with the best results is sent to stage 2.

2. Stage 2: We set up the selected model from Stage 1 to use the training subset. Every query or row in the dataset is fed into the model, and each query's energy output is compared to various aspects of the query itself. In the case of an NLP task, the query is a sentence; the query length and complexity are acquired from the query and compared to the energy output. Due to project time limitations, the query length alone was compared with energy consumption, as shown in section 5.3. This gives us an understanding of the factors behind an input's energy consumption.

3. Stage 3: The selected model is now decomposed into its core layers. In the above framework, the analysis would have been performed on the transformer from Vaswani et al. [1]. Models vary in architecture, and the framework would have to adapt to each one. For example, 'bert-base-uncased' is composed of the tokenization stage, 12 encoder layers, and the output stage, so stage three would reshape and represent this architecture. This step is crucial to understanding the architectural differences between models and which aspects of these architectures impact energy consumption, which could give rise to new models, which would be a merger of both.

The metrics used in Rugitini et al. [13], the comparative study inspiring this project, are limited to the F1 score, which is insightful but general

and needs deeper context to perform specific tasks. This project adds an additional layer of metrics that should be specific to the task specified, in this case, MCC for binary classification. This report analyzes three metrics layers, including the general performance metrics made to compare with other tasks and papers (Accuracy), the task-specific metric (MCC), and the energy consumption measurement (Joules) indicating efficiency, and it collectively concludes the best-performing model.

Regarding the energy consumption, the 'measure_energy' function from the PyJoules library [25]. To the model was restricted to running on the GPU alone with PyTorch; the function was decorated with the GPU domain, and consumption was measured in joules. This restriction prevented any additional errors from occurring and additional data manipulation while measuring from various devices. One of the significant drawbacks of this process is that the energy measurement represents the consumption of the entire GPU, including other processes; hence, during any code run, many other processes affecting the GPU were paused to ensure minimal usage of the GPU. This versatile tool offered many different implementations and can be stopped and tagged for energy consumption at different stages of a code, a significant reason why PyJoules was chosen for this research.

Hugging Face organizes the models, datasets, and metrics, and they are easily downloadable. This framework systematically compares models and their energy efficiency and could be applied to any task with proper automation.

3.3 Analysis Stages

This section explains the research and testing steps to achieve the prototype framework. Training is the most energy-consuming stage [4, 5]. As such, the majority of the analysis was conducted at this stage, which does not mean we should neglect the rest. The analysis sections start with the pre-stages of research involving the model selection through memory and speed inference and comparing the full and fraction of the data on accuracy. The next step, the production stage, details the three framework stages and the data flow from raw to outputs. Let's start with choosing the models and datasets for this case study.

3.3.1 Research Phase

Speed & Memory Inference for Model Selection
This first stage defined the models described later in this section. The results and discussion for this section can be found in Section 4.1.1. Based on the suggestion from Argerich et al. [23], this point of the research evaluates a variety of models for their run time and memory usage over different batch sizes and sequence lengths. Three batch sizes (size = 8, 16, 32), which depend on the device capacity, alongside four sequence lengths (size = 32, 64, 128, 256, 512), which depend on the model capacity; all selected models can take up to 512 in sequence length. The 'PyTorchBenchmarkArguments' class from PyTorch performed the speed and memory benchmarking. We first create an instance of this class, which encapsulates all the parameters necessary for the benchmarking process (models, batch sizes, and sequence lengths). We then initialize the 'PyTorchBenchmark' [26] class with these arguments, seen in appendix .2, and prepare the benchmark environment. Finally, we executed the run function on the benchmark instance and stored the results. The results are plotted and compared until the models have comparable performances.
To achieve a good mixture of models while maintaining similar performances, the selected models had to satisfy the following criteria: a popular base model, a tuned model for efficiency, as seen in Section 2.5, and models tuned for a specific dataset. Here is the final list of models used alongside their architectures (model_creater/model_name as seen in Hugging Face hub):

- 'google-bert/bert-base-uncased': This model consisted of 110M parameters with 12 transformer encoder layers and was one of the most popular models, with over 61M downloads in July 2024 alone, it is perfect as a benchmark for other models. It is uncased and pre-trained on a large corpus of English language data in a self-supervised fashion. It was mainly trained for two objectives but is used for others: masked language modeling (MLM) and next sentence prediction (NSP).

- 'distilbert/distilbert-base-uncased': Similar to 'bert-base-uncased' but smaller, this model consists of 6 encoder layers instead of 12 and still achieves remarkably close results. The model is 40% smaller yet performs 60% faster [24] and retained 97% of BERT's language understanding.

- 'distilbert/distilroberta-base': Similar to the above, this is the distilled version of the RoBERTa-base model but has 6 encoder layers and 85M parameters instead of 125M. It also performs similarly to 'distilbert-base-uncased' but is uncased.

- 'tanganke/gpt2_sst2': This is a tuned version of GPT-2 for the SST-2 subset, a great contender to the other models. The model architecture has 125M parameters and 12 layers, making it larger than the rest.

- 'dimboump/glue_sst_classifier': This is a fine-tuned version of 'bert-base-uncased' specifically made to tackle the SST-2 subset from the GLUE benchmark.

Full Dataset vs 5% for Accuracy Comparison

Training the BERT model on the CoLa subset (10.7k rows) from the GLUE benchmark took approximately 8 hours. Performing numerous experiments with complete datasets would be time-consuming and could lead to device wear.

Using the same setup seen in 3.3, the base model, in this case 'bert-base-uncase', is trained and evaluated in the CoLa subset (Section 2.2.3). The experiment is run twice, first with the entire data set and secondly with a portion. Cola was chosen to examine this stage of the process as it is one of the smallest and also a binary classification problem, SST-2 is much larger in size and sequence lengths.

This study, which focused on energy consumption rather than model performance enhancement, used 5% of the data to expedite experiments. This approach significantly reduced the training time to almost half an hour to an hour, with only a marginal accuracy reduction (about a 5.03% drop). The accuracy and Matthews Correlation Coefficient (MCC) of a base model were collected on both the small and large datasets, and minimal differences in performance were found 4.1. The results allowed the production stage to begin confidently proceeding with 1% to 5% of GLUE's SST-2 subset (70k rows) to measure energy consumption, with a small sacrifice in performance.

3.3.2 Production Stage

In the production stage we aim to develop a methodology that is reusable and extendable to future studies. the first stage of the framework aims to replicate the findings in other comparative studies [13, 14]. To further

the research already done, stage two looks into the affect of queries on the energy consumption. Stage 3 was never put into production officially in this report and code, however insights are still valuable. At each stage, begin by loading the datasets. Use the 'load_data' function from the 'datasets' library in Hugging Face, and include the 'split' parameter in the function to take a subset of the data. For example, to load 5% of the training data, use "split='train[:5%]'". Apply the same approach to load subsets for validation and test sets as needed.

Stage 1: Energy-Consumption over Training & Evaluation

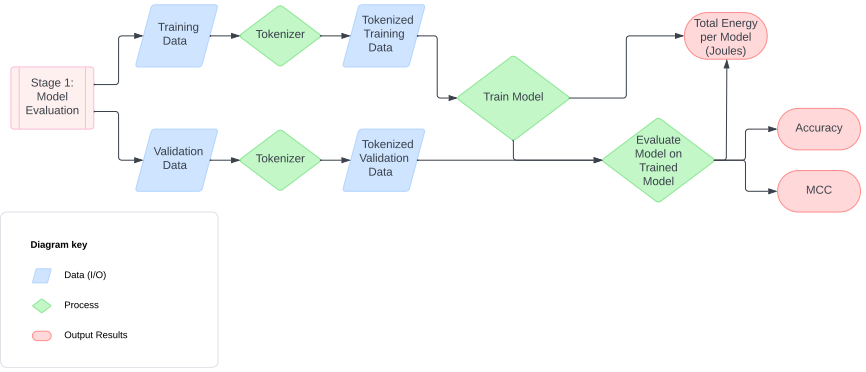


Figure 3.2: Stage 1 of the prototype framework: Model vs Energy vs Performance

Setting up this stage requires three main stages: tokenization, training, and evaluation. Using the split parameter above, we take 5% of the dataset for the training and validation set. Two definitions were created for this task: 'evluate_model' and 'compute_metrics'. During the evaluation stage,

the evaluate_model functions call the compute_metrics to compute the accuracy and MCC scores. 'evaluate_model' takes in 3 parameters: the model name, the raw training, and evaluation subsets. It first utilizes the classes 'AutoTokenizer' and 'AutoModelForSequenceClassification' from the 'transformers' library in Python to automatically download a pre-trained version of the model name. Once the tokenizer and model instances have been created using these classes, we move the model using the 'to(device)' function from PyTorch to force the model to the device designated as the GPU using CUDA, created by NVIDIA and development for GPU processing.

The data still needs to be tokenized before processing. The functions 'map()' and 'set_format' were used to map the tokenization function to each row in the data and convert it to tokens using the created tokenizer instance, and the tokenized data is PyTorch formatted to be applied on the GPU. We can now finally set up the training arguments for the trainer.

In this phase, we specify the parameters for the training arguments, such as the number of epochs, the batch size, the number of evaluation samples, and the learning rate. The trainer instance from the 'Trainer' class is created using the previous training arguments, the model instance, the tokenized data, and the 'compute_metrics' function for evaluation.

We decorate the 'evaluate_model' definition with the '@measure_energy' function, which records the energy consumption from the specified domain found while testing for the GPU, and the handler, which specifies the method the results will be saved (CSV in this case). The training and evaluation process may now begin.

Each selected model, established in Section 3.3.1, is trained and evaluated on the subset using the pipeline in 3.3. The results for accuracy, MCC, model loss, evaluation runtime, and energy consumption of training and evaluation are recorded. Having organized and cleaned the retrieved data, we visualize the results using the Matplotlib library and create bar plots for each one of the metrics 4.2. Note that the model loss represented is the last one from training, the last epoch.

By the end of this process, the model is fully prepared, trained, and evaluated with precision, setting the stage for further analysis.

Stage 2: Energy-Consumption over Training per Query

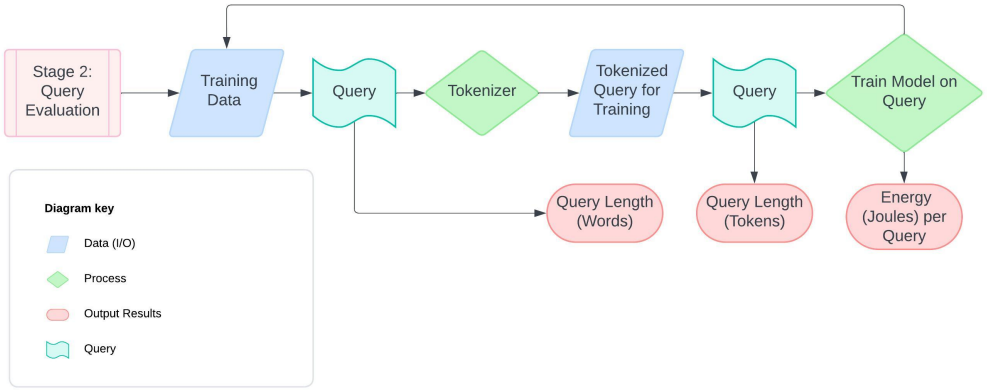


Figure 3.3: Stage 2 of framework: Query vs Energy

Stage 2 of the analysis mirrors Stage 1 but with a specific focus on measuring the energy consumption of training individual queries across different models. In this stage, only 5% of the training subset is used to capture energy consumption data, as performance metrics are not the priority here. The 'evaluate_model_query' function plays a key role in this process. It trains the model on a single query and measures the energy usage during training. Unlike Stage 1, this function is streamlined, excluding model instantiation and evaluation metrics, focusing solely on training to envelop it with the 'measure_energy' function. This function is paired with a CSV handler to store the collected energy data, specifically within the GPU domain.

The main body of the code then uses a nested loop structure to apply this function. The outer loop iterates over a list of selected models, initializing both the tokenizer and model, and moving the model to the GPU. The inner loop processes each tokenized query, ensuring the data is in the correct format for PyTorch. This setup enables the systematic measurement of energy consumption for each query across different models, providing valuable insights into the energy demands of various transformer-based models.

Stage 3: Energy-Consumption over Training per Stage

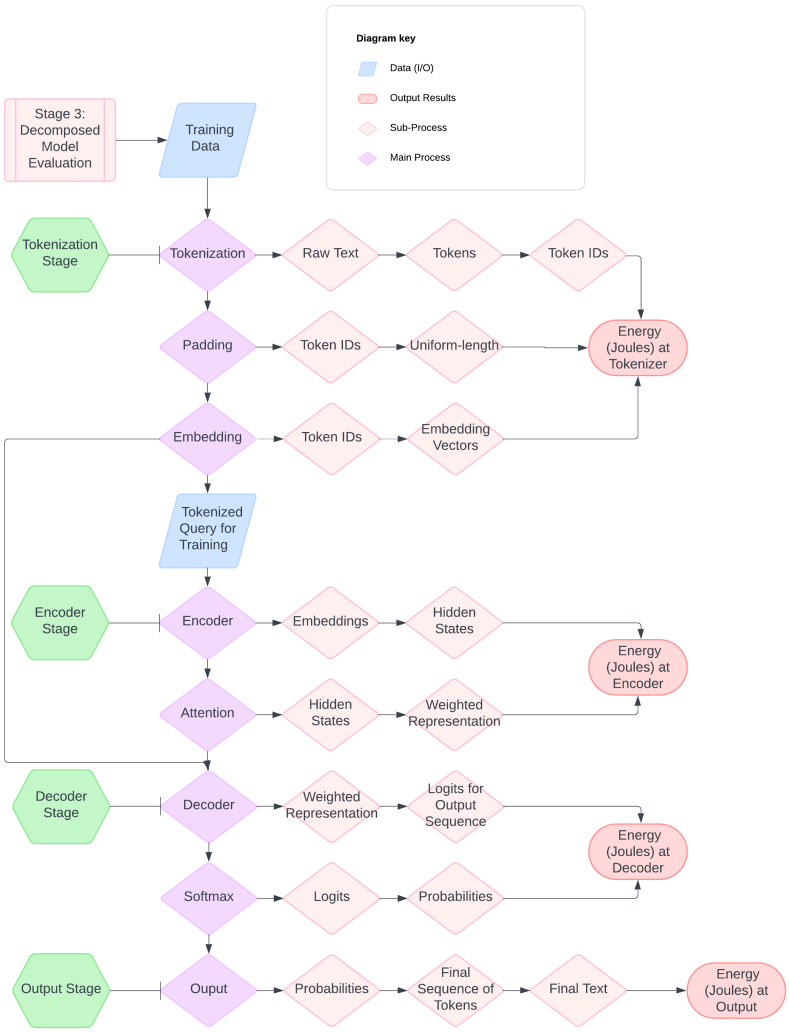


Figure 3.4: Stage 3: Energy vs Transformer components

In Python, the first step in a sequence-to-sequence model is the tokenizer layer, where we convert raw text into tokens using libraries like Hugging Face's transformers. These tokens are then mapped to numerical IDs based on a pre-defined vocabulary, forming the foundation for the model's input. To handle sequences of varying lengths, padding is applied to shorter sequences

so that all inputs in a batch have the same length, which is crucial for efficient processing. The energy demands at this stage primarily stem from the computational load of tokenizing large datasets and managing padding effectively.

Next, the encoder layer takes these padded sequences and processes them to capture their contextual information. Typically, this involves using a transformer encoder, though other architectures like RNNs can be used as well. The encoder transforms the token IDs (or their embeddings) into hidden states, which represent the meaning and context of each token within the sequence. If an attention mechanism is in place, it helps the model focus on the most relevant parts of the input sequence. The energy usage at this stage increases with the complexity of the model and the length of the input, especially when incorporating attention mechanisms.

The decoder layer is where the model generates the output sequence, one token at a time. It uses the hidden states from the encoder and the attention outputs to produce logits, which are essentially raw predictions for each token in the vocabulary. The decoder then applies a softmax function to these logits, turning them into probabilities. The energy required here scales with the size of the vocabulary and the sequence length, as generating each token requires significant computation.

Finally, the output layer selects the most likely tokens based on the softmax probabilities, creating the final output sequence. This sequence is then detokenized back into readable text. While this stage is less energy-intensive compared to earlier steps, it's crucial for ensuring that the final output is both accurate and coherent, especially in real-time applications where performance and correctness are critical.

3.4 Project Management

The project management plan outlines the timeline, key activities, milestones, resources, and deliverables to ensure successful completion.

Timeline & Gantt Chart

This gantt chart 3.5 is automatically syncornised to my email and callendar keeping me alert of all project deadlines, using Notions' embedded mermaid code.

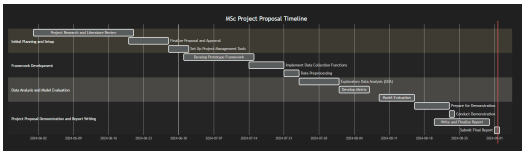


Figure 3.5: Enter Caption

Here is an overview of how the summer period was organized:

- June: Initial Setup and Literature Review
    1. Set up project management tools (Notion page, Gantt chart).
    2. Conduct an in-depth literature review to identify gaps and establish a theoretical basis.
    3. Finalize research questions and objectives.
- July - August: Framework Development & Model Evaluation and Analysis
    1. Develop a prototype framework for energy profiling of LLMs.
    2. Apply the prototype framework to 5 models using the GLUE benchmark dataset and analyze stage 1 and 2 of this project.
    3. Implement data collection functions to measure energy consumption and performance metrics.
    4. Begin data preprocessing and exploratory data analysis.
    5. Analyze the energy consumption at different stages of the LLM pipeline.
    6. Develop and validate new metrics to assess energy efficiency and model reliability.
- August - September: Demonstration & Report
    1. Prepare documentation for the framework, including user guidelines and technical details.
    2. Develop a public dashboard to visualize energy consumption and performance metrics.
    3. Conduct a demonstration to showcase the framework's functionality and findings.

4. Finalize the report for the project.

Key Activities:

1. Setup and Planning:
   - Organize project structure and timelines.
   - Conduct literature review.
2. Framework Development:
   - Create a prototype framework.
   - Implement data collection and preprocessing functions.
3. Analysis and Evaluation:
   - Apply framework to selected model and dataset.
   - Analyze energy consumption and performance metrics.
4. Documentation and Reporting:
   - Document the framework and findings.
   - Prepare for project demonstration and final report submission.

4 Results & Discussion

Similarly to the methodology this section will reflect the research and production steps taken starting off with the results of the research phase where we loop into the plots for various batches and sequence length vs time and memory usage.

4.1 Research Phase Results

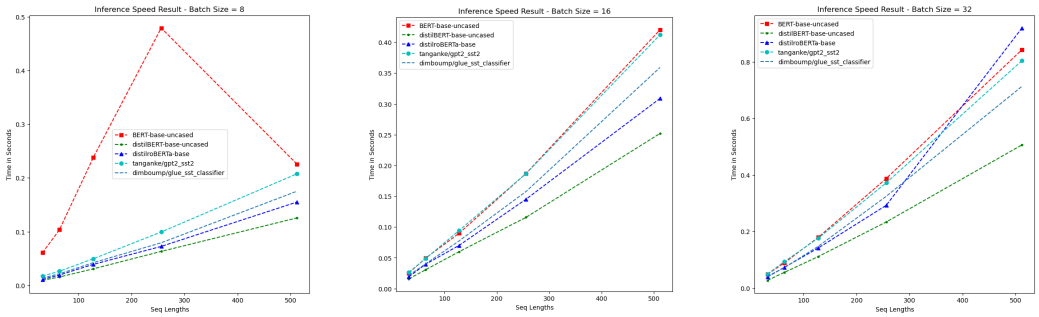
The goal of this phase was to select models and datasets appropriate for the analysis stages. The results seen below belong to the selected models only.

4.1.1 Speed & Memory Inference

As mentioned, we first perform speed and memory inference, beginning with analyzing speed results. The first row of results in figure Figure 1 below displays the speed inference and second memory inference outcomes. Each figure corresponds to a specific batch size ranging from 8 to 32. The results indicate consistent speeds across all batch sizes and sequence lengths for the four selected models, with a few notable observations for the base model 'BERT-base-uncased'. The first notable difference from all figures is that BERT takes significantly more time with small batch sizes; it is estimated that this is due to its inherent large size. Moreover, although 'BERT-base-uncased' has similar performance in speed for a batch size of 16 and above, the memory is consistently lower, meaning it would take the model more time to process certain information, also probably due to its large size. The weird behavior seen in Figure 4.1 when batch size is 8 and sequence length is around 300, there is a spike in inference runtime; this requires further research.

Metric	Complete Data	5% of Data
Accuracy	0.8303	0.7885
MCC	0.5833	0.4892
Eval Loss	0.5968	0.9554

Table 4.1: Performance metrics for the complete dataset and 5% dataset.



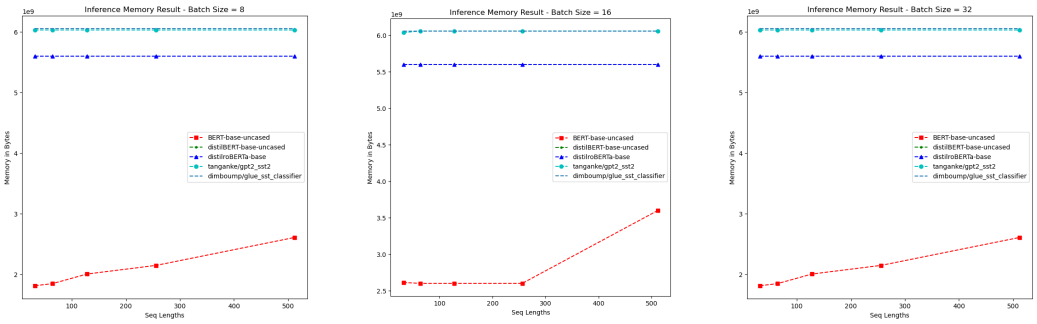


Figure 4.1: Plots showcase the effects of sequence length and batch size on speed and memory inference for 5 models

4.1.2 Full Dataset vs 5% for Accuracy Comparison

Using the 'BERT-base-uncased' model, we trained and evaluated the full CoLA dataset from GLUE, as well as on a 5% subset. The goal was to compare the resulting loss in accuracy and other metrics in the following testing stages. 4.1 presents the results of that comparison. We observed a 5.1% drop in accuracy and a 15.9% drop in MCC. This clearly shows how much more accurate the fully trained model is, as its losses is much closer to 0. Despite these performance drops, it is important to note that we reduced the dataset size by 95%, which significantly sped up data processing times. The primary goal of this project is to compare the models with each other, not necessarily to create a better model.

4.2 Production Phase Results

4.2.1 Stage 1

In regular testing circumstances, 'BERT-base-uncased' would have been more efficient, as it is a larger model designed to handle large batch sizes. However, with smaller batch sizes like these, it performed poorly compared to its counterparts. Despite this, the results still support the overall conclusion.

Table 4.1 and 4.2 compare the performance and energy efficiency of five models, highlighting the trade-offs between accuracy and GPU energy consumption. BERT-base-uncased achieves solid accuracy (0.91) and an MCC of 0.8216, but at a steep energy cost of 186.23 kWh. In contrast, the distilled

versions—distilBERT-base-uncased and distilroBERTa-base—maintain a higher accuracy (0.95) and MCC (0.91) while drastically reducing energy consumption to just 9.22 kWh and 8.97 kWh, respectively. These distilled models are approximately 95% more energy-efficient than the original BERT-base-uncased model, proving that significant energy savings can be achieved without a corresponding drop in performance as compared to results found in Rigutini et al. [13] (Section 2.4).

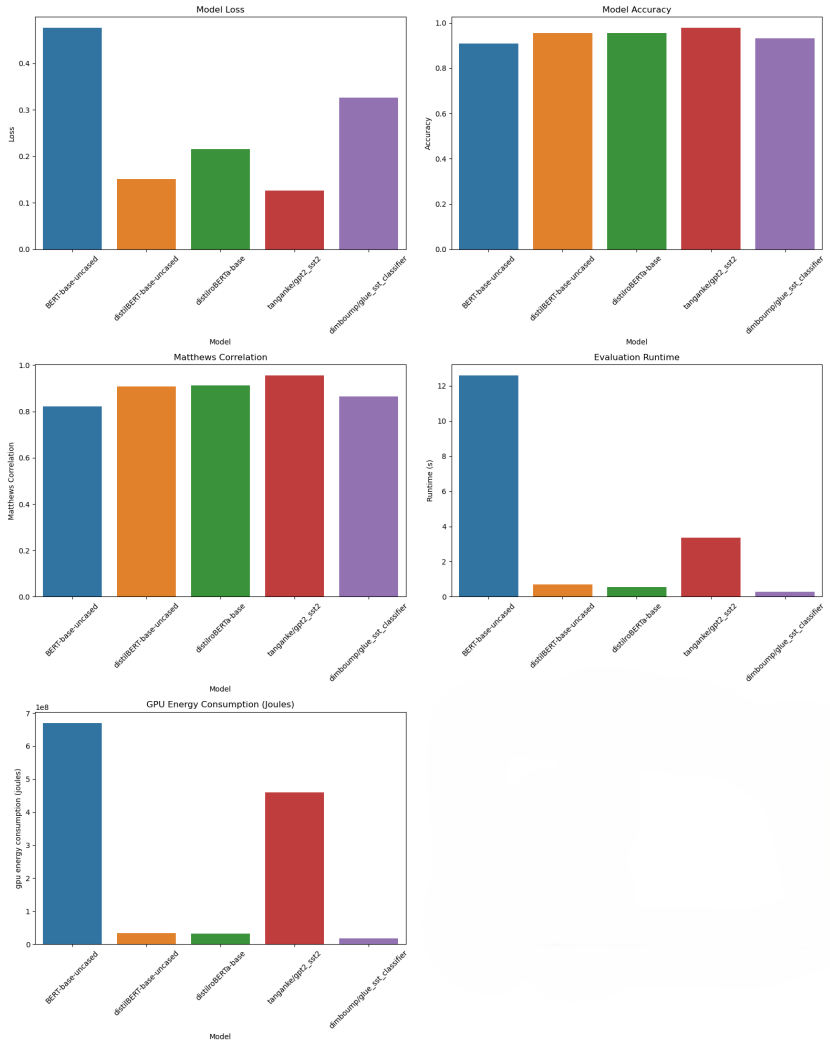


Figure 4.2: Stage 1 Model Performance vs Energy Consumption

tanganke/gpt2_sst2 stands out with the highest accuracy (0.98) and MCC (0.9555) among the models tested, but this comes at an energy cost of 127.88 kWh, which is still considerably lower than BERT-base-uncased but much higher than the distilled models. While tanganke/gpt2_sst2 offers exceptional performance, the energy trade-off may not be justifiable when compared to more efficient models like distilBERT-base-uncased and distilroBERTa-base, which use over 92% less energy. On the other hand, dimboump/glue_sst_classifier offers a balance with moderate accuracy (0.93) and MCC (0.8645) while consuming the least energy at 4.77 kWh, making it the most energy-efficient option for tasks where absolute performance is less critical. In summary, if energy efficiency is a key concern, distilBERT-base-uncased and distilroBERTa-base offer the best balance between performance and energy consumption, whereas models like tanganke/gpt2_sst2 are more suitable when the highest accuracy is necessary, despite the higher energy cost.

Table 4.2: Performance metrics and GPU energy consumption for different models.

Model	Loss	Accuracy	MCC	GPU Energy Consumption (kWh)
BERT-base-uncased	0.4761	0.91	0.8216	186.2282
distilBERT-base-uncased	0.1512	0.95	0.9091	9.2228
distilroBERTa-base	0.2154	0.95	0.9129	8.9692
tanganke/gpt2_sst2	0.1257	0.98	0.9555	127.8752
dimboump/glue_sst_classifier	0.3265	0.93	0.8645	4.7659

4.2.2 Stage 2

Before comparing each query's energy consumption, we first look at 4.3 and 4.4, which show the distributions for query lengths and token lengths, respectively. This step was crucial to ensure that the tokens produced similar distributions, considering that tokens can represent entire words, parts of words, or even individual letters and numbers. Models handle varying token lengths differently. Ensuring similar distributions helps assess the models' consistency in their performance and energy consumption, regardless of the input tokenization. The results for stage 2 can be seen in 4.5, 4.6, and 4.7. 4.5 is a scatter plot of the sequence length vs energy consumption with the hue set to the model. The figure nicely showcases that the energy consumption within each sequence length can fluctuate drastically, which was a surprising results. For sentences with sequence lengths of 0 to 50, the energy consump-

tion required to train a single sentence ranged from 25,000 to 65,000 joules, which is about 1.39 and 3.61 grams of CO2 emissions per query, considering the UK, where the standard estimate is 0.2 KG per kWh. For perspective, we can either drive at 60 mph for 30 min, covering 30 miles, or we can train on 3000 data points. Using this assumption, we could have either trained GPT-3 or driven 118 years straight at 60 mph. Further examining 4.5, we notice that the sequence length does not affect energy consumption. For further investigation, 4.6 and 4.7 were made. The mean line in 4.6 shows very minor changes over sequence length, but due to the lack of sequences with lengths above 100, we could assume that the lack of data points could be provoking a slight slope. 4.6 further proves this point. The sequence length was binned, and the average energy consumption of the sequences within each bin was plotted; the figure clearly shows no real change in energy consumption for sequence length. Although this looks conclusive, more research should be done to verify these results, such as changing the batch sizes or adjusting testing with larger datasets with more extensive sequences. Since a sentence with a specific sequence length does not affect energy consumption, the range seen in these figures suggests that other research must be done. One suggestion would be to look at query complexity.

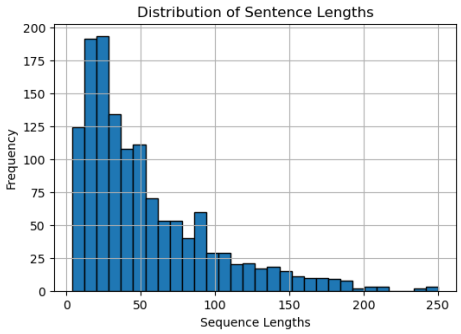


Figure 4.3: Distribution of sequence lengths

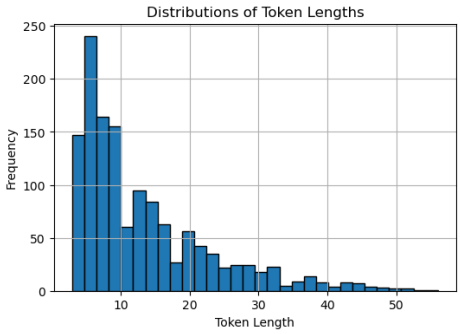


Figure 4.4: Distribution of token lengths

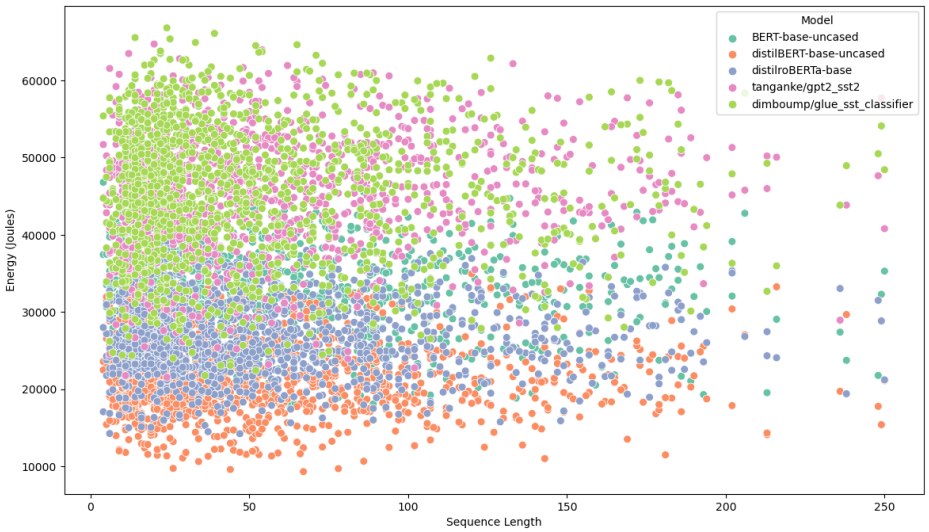


Figure 4.5: Scatter plot of energy consumption in joules vs sequence length with the hue set to each model

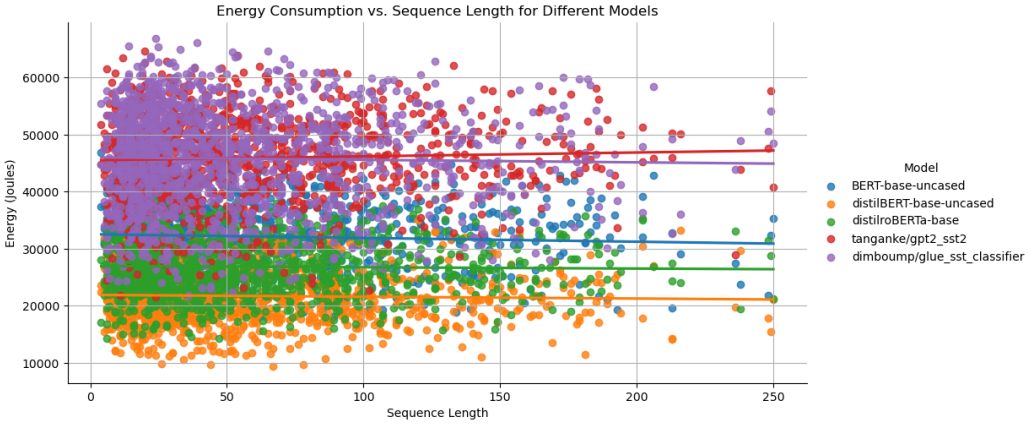


Figure 4.6: Scatter plot of energy consumption in joules vs sequence length with the hue set to each model and the mean lines for each models

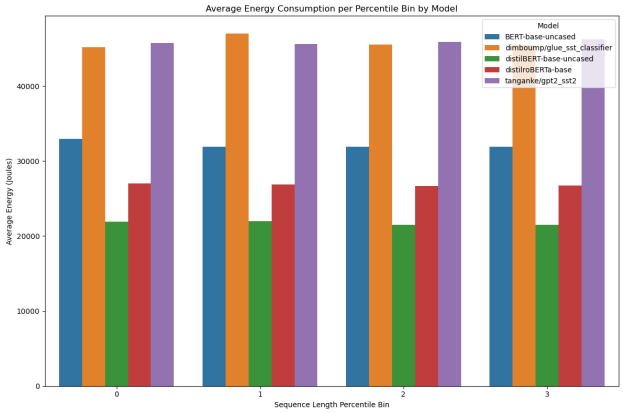


Figure 4.7: Bar plot of average energy consumption per binned percentile sequence lengths

5 Conclusion & Future Work

The results indicate the need for ongoing research in this domain, revealing that energy consumption does not scale linearly with performance and runtime, as different machines would exhibit varying energy profiles.

5.1 Contribution

This report provides a comprehensive analysis of energy consumption across various transformer-based NLP models, contributing to a deeper understanding of the trade-offs between model performance and energy efficiency. The key contributions of this work include:

1. **Replication of Results Using Basic Instruments:** The successful results in Section 4.2.1 (Stage 1) demonstrated that the energy consumption of various transformer models can be replicated [13] using readily available basic instruments (device in Section 3.1). For example, comparing the best model for accuracy, 'gpt_sst2', and the best energy-efficient model, 'distilBERT-base-uncased', we find that the MCC only dropped by 4.2% and compared to a 92% drop in energy consumption; if accuracy is not your primary concern this tradeoff is not balanced. The variant of GPT-2 consumed 127 kWh of energy compared to distilBERT, which consumed only 9 kWh. For UK households, the average

cost for one kWh is about 35 pence; the cost attributed to this consumption would be about 44.45 GBP vs. 3.15 GBP for small-scale applications such as mine.

2. **Influence of Query Length on Energy Consumption:** Section 4.2.2 reveals that while there are evident factors, as seen in 4.5, affecting the energy consumption of a query, such as batch size and query complexity, which significantly impact energy consumption from 2000 to 7000 Joules on the same query. The results show that query length does not affect energy consumption. This counterintuitive finding is likely due to the role of the DataCollator function (acts on during tokenization stage), which pads sentences based on the model's input token length. For instance, if a model is designed to handle up to 512 tokens, shorter queries are padded to this length, thereby normalizing energy consumption across different query lengths. This insight explains why query length alone does not influence energy usage and highlights the importance of considering other mechanisms, such as query complexity and batch sizes. These are solid speculations; however, there is still a need for further extensive research as this industry is relatively new.

3. **Framework for Analyzing Energy Consumption in Transformer Pipelines:** This report laid the groundwork for a comprehensive framework to analyze energy consumption across different stages of the transformer pipeline 3.4, including tokenization, encoding, decoding and output. This framework can serve as a foundational tool for future research, offering a structured approach to identifying energy-intensive processes and optimizing them for more efficient model deployment.

5.2 Challenges & Limitations

The project faced two primary challenges: hardware limitations and time constraints.

Hardware Limitations:

- **Limited Access to High-Performance GPUs:** The experiments required significant computational resources, particularly high-performance GPUs, 200GB and above [13], to process large models and extensive datasets. Limited administrative access to such hardware constrained

the ability to run more extensive experiments, particularly those involving larger batch sizes or more complex models.

- **Energy Monitoring Constraints:** Accurate measurement of energy consumption required additional research and coding. Solutions such as building a virtual environment are suggested. This limitation affected the precision of energy consumption data, particularly for fine-grained measurements during short or low-power operations such as energy consumption per query.

Time Constraints:

- **Extended Experimentation Periods:** Running multiple iterations of experiments to obtain statistically significant results, especially across different batch sizes and model configurations, required more time than initially anticipated. This extended the project timeline and limited the scope of experiments.

- **Comprehensive Analysis Requirement:** The project's goal to develop a framework for analyzing energy consumption across various stages of the transformer pipeline demanded an extensive analysis of numerous variables.

5.3 Future Work

There are numerous opportunities for future work, and based on the insights gathered from this project, the following steps should be prioritized:

1. **Exploring Energy Consumption Over Query Complexity:** A deeper investigation is needed to understand how query complexity impacts energy consumption, especially concerning batch size, model size, and tokenization strategies. This research will refine our understanding of how transformer models utilize energy.

2. **Completing Stage 3:** Replicating Findings on Current Models: The next essential step is to replicate the findings on a widely-used model, such as one with an encoder-based architecture, to compare it against decoder-based architectures. This will help identify any differences in energy consumption across various model types.

3. **Completing Stage 4:** Assessing Energy Efficiency of Tuned Models: The focus will then shift to evaluating the energy efficiency of various model tuning techniques, including quantization, pruning, distillation, and vocabulary transfer. The goal is to find the best methods that balance performance with energy savings.

4. **Addtional Work:**

    - **Isolating Processes in a Virtual Environment:** To achieve more precise energy consumption measurements, isolating the experiments in a virtual environment with full administrative access will be crucial. This approach will help remove external factors that could skew the results.
    - **Automating the Stage 1 Process:** There's potential to automate the Stage 1 process, allowing users to choose specific tasks, datasets, models, and metrics, and have the system generate the outputs automatically. However, this would require significant computational resources, as models and datasets are continually updated.

These steps should provide a clear roadmap for future work, building on the foundations laid by this project.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *arXiv*, 1706.03762v7, June 2017. Available from: https://doi.org/10.48550/arXiv.1706.03762.

[2] F. Duarte. Number of chatgpt users (aug 2024), July 2024. Available from: https://explodingtopics.com/blog/chatgpt-users.

[3] Hugging Face. Hugging face – the ai community building the future. Accessed 2024 Jun 20, 2024. Available from: https://huggingface.co/.

[4] E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in nlp. *Proceedings of the 2019 Association for Computational Linguistics (ACL)*, pages 3645–3650, 2019. Available from: https://aclanthology.org/P19-1355/.

[5] D. Patterson, J. Gonzalez, M. Le, et al. Carbon emissions and large neural network training. *Communications of the ACM*, 64(6):24–26, 2021.

[6] M. Sun, Z. Liu, A. Bair, and J.Z. Kolter. A simple and effective pruning approach for large language models. *arXiv*, 2306.11695, June 2023. Available from: https://doi.org/10.48550/arXiv.2306.11695.

[7] E. Latif, L. Fang, P. Ma, and X. Zhai. Knowledge distillation of llm for automatic scoring of science education assessments. *arXiv*, 2312.15842, December 2023. Available from: https://doi.org/10.48550/arXiv.2312.15842.

[8] A. Yamaguchi, A. Villavicencio, and N. Aletras. An empirical study on cross-lingual vocabulary adaptation for efficient language model inference. *arXiv*, 2402.10712, February 2024. Available from: https://doi.org/10.48550/arXiv.2402.10712.

[9] J. Lin, J. Tang, H. Tang, S. Yang, WM. Chen, WC. Wang, et al. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv*, 2306.00978, June 2023. Available from: https://doi.org/10.48550/arXiv.2306.00978.

[10] S. Yildirim. *Mastering a Transformer*. Packt Publishing, Canada, 2024.

[11] K. Blunt and J. Hiller. Big tech's latest obsession is finding enough energy. *Wall Street Journal*, 2023. Available from: https://www.wsj.com/business/energy-oil/big-techslatest-obsession-is-finding-enough-energy-f00055b2.

[12] A. Andrae and T. Edler. On global electricity usage of communication technology: Trends to 2030. *Challenges*, 6:6–17, 2015.

[13] L. Rigutini, A. Globo, M. Stefanelli, A. Zugarini, S. Gultekin, and M. Ernandes. Performance, energy consumption and costs: A comparative analysis of automatic text classification approaches in the legal domain. *Int J Nat Lang Comput*, 13(1):19–35, 2024.

[14] S. Gultekin, A. Globo, A. Zugarini, M. Ernandes, and L. Rigutini. An energy-based comparative analysis of common approaches to text classification in the legal domain. In DC Wyld, editor, *The 4th International Conference on NLP Text Mining (NLTM 2024)*, volume 14, pages 2–3, Copenhagen, Denmark, 2024.

[15] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and SR. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv*, 1804.07461, February 2019. Available from: https://doi.org/10.48550/arXiv.1804.07461.

[16] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 1409.0473, 2016. Available from: https://doi.org/10.48550/arXiv.1409.0473.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[18] DE. Rumelhart, GE. Hinton, and RJ. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[19] N. Zucchet and A. Orvieto. Recurrent neural networks: vanishing and exploding gradients are not the end of the story. *arXiv*, 2405.21064, May 2024. Available from: https://doi.org/10.48550/arXiv.2405.21064.

[20] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrellas. Towards greener llms: Bringing energy-efficiency to the forefront of llm inference. *arXiv*, 2403.20306, March 2024. Available from: https://doi.org/10.48550/arXiv.2403.20306.

[21] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[22] N. Kumar, U. Chatterjee, and S. Schockaert. Ranking entities along conceptual space dimensions with llms: An analysis of fine-tuning strategies. *arXiv*, 2402.15337, June 2024. Available from: https://doi.org/10.48550/arXiv.2402.15337.

[23] MF. Argerich and M. Patiño-Martínez. Measuring and improving the energy efficiency of large language models inference. *IEEE Access*, 12:80194–80207, 2024.

[24] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv*, 1910.01108, October 2019. Available from: https://doi.org/10.48550/arXiv.1910.01108.

[25] PyJoules. Pyjoules documentation. Accessed 2024 Aug 26, 2024. Available from: https://readthedocs.org/projects/pyjoules/.

[26] PyTorch. Benchmarking. Accessed 2024 Aug 26, 2023. Available from: https://pytorch.org/tutorials/recipes/recipes/benchmark.html.

Appendix

Git Repository

```python
# Devorator function
@measure_energy(handler=csv_handler, domains=[NvidiaGPUDomain(0)])
def evaluate_model_on_cola(model_name, data_train, data_validation, data_test):

    # Load tokenizer and and model (Pytorch Version)
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSequenceClassification.from_pretrained(model_name)

    # Move the model to the GPU
    model.to(device)

    def tokenize_function(example):
        return tokenizer(example['sentence'], padding="max_length", truncation=True)

    # Tokenize the data
    tokenized_data_train = data_train.map(tokenize_function, batched=True)
    tokenized_data_validation = data_validation.map(tokenize_function, batched=True)
    tokenized_data_test = data_test.map(tokenize_function, batched=True)

    # Set the format for PyTorch and move tensors to the GPU
    tokenized_data_train.set_format("torch", columns=["input_ids", "attention_mask", "label"])
    tokenized_data_validation.set_format("torch", columns=["input_ids", "attention_mask", "label"])
    tokenized_data_test.set_format("torch", columns=["input_ids", "attention_mask", "label"])

    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

    # Set the training arguments
    training_args = TrainingArguments(
        output_dir="./result_1",
        evaluation_strategy="epoch",
        learning_rate=2e-5,
        # Reduce from 16 to 8 to help with gpu memory error
        per_device_train_batch_size=8,
        per_device_eval_batch_size=8,
        num_train_epochs=3,
```

Figure .1: Code Snippet for Energy Function decorator and definition

```python
# Set the training arguments
training_args = TrainingArguments(
    output_dir="./result_1",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    # Reduce from 16 to 8 to help with gpu memory error
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

# Create the trainer for the model
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_data_train,
    eval_dataset=tokenized_data_validation,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

Figure .2: Model training arguments