KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Technology

First cycle, 15 credits

Deriving an Natural Language Processing inference Cost Model with Greenhouse Gas Accounting

Towards a sustainable usage of Machine Learning

TOM AXBERG

Stockholm, Sweden, 2022

Deriving an Natural Language Processing inference Cost Model with Greenhouse Gas Accounting

Towards a sustainable usage of Machine Learning

TOM AXBERG

Degree Programme in Information and Communication Technology
Date: June 22, 2022

Supervisors: Oliver Gindele, Anders Västberg
Examiner: Gerald Quentin Maguire Jr
School of Electrical Engineering and Computer Science
Host company: Datatonic
Swedish title: Härledning av en Kostnadsmodell med växthusgasredovisning angående slutledning inom Naturlig Språkbehandling
Swedish subtitle: Mot en hållbar användning av Maskininlärning

Abstract

The interest in using State-Of-The-Art (SOTA) Pre-Trained Language Models (PLMs) in product development is growing. The fact that developers can use PLMs has changed the way to build reliable models, and it is the go-to method for many companies and organizations. Selecting the Natural Language Processing (NLP) model with the highest accuracy is the usual way of deciding which PLM to use. However, with growing concerns about negative climate changes, we need new ways of making decisions that consider the impact on our future needs. The best solution with the highest accuracy might *not* be the best choice when other parameters matter, such as sustainable development.

This thesis investigates how to calculate an approximate total cost considering Operating Expenditure (OPEX) and CO_2 emissions for a deployed NLP solution over a given period, specifically the inference phase. We try to predict the total cost with Floating Point Operations (FLOPs) and test NLP models on a classification task. We further present the tools to make energy measurements and examine the metric FLOPs to predict costs.

Using a bottom-up approach, we investigate the components that affect the cost and measure the energy consumption for different deployed models. By constructing this cost model and testing it against real-life examples, essential information about a given NLP implementation and the relationship between monetary and environmental costs will be derived.

The literature studies reveal that the derival of a cost model is a complex area, and the results confirm that it is not a straightforward procedure to approximate energy costs. Even if a cost model was not feasible to derive with the resources given, this thesis covers the area and shows why it is complex by examine FLOPs.

Keywords

Machine Learning, Inference, Inferencing, Natural Language Processing, Pre-trained Language Model, Greenhouse Gas, Software Energy Measurement, Floating Point Operations, Green-AI, Green-IT, OPEX

Sammanfattning

Intresset att använda SOTA PLMs i produktutveckling växer. Det faktum att utvecklare kan använda PLMs har förändrat sättet att träna tillförlitliga modeller på och det är den bästa metoden för många företag och organisationer att använda SOTA Naturlig Språkbehandling (NLP). Att välja NLP-modellen med högsta noggrannhet är det vanliga sättet att bestämma vilken PLM som ska användas. Men med växande oro för miljöförändringar behöver vi nya sätt att fatta beslut som kommer att påverka våra framtida behov.

Denna avhandling undersöker hur man beräknar en ungefärlig totalkostnad med hänsyn till OPEX och CO_2 utsläpp för en utplacerad NLP-lösning under en given period, dvs slutledningsfasen. Vi försöker förutspå den totala kostnaden med flyttalsoperationer och testar mot en klassificerings uppgift. Vi undersöker verktygen för att göra mätningar samt variabeln Flyttalsoperationer för att förutspå energiförbrukning.

Nyckelord

Maskininlärning, Slutledning, Naturlig Språkbehandling, Förutbildad språkmodell, Växthusgas, Energimätning av Mjukvara, Green-AI, Green-IT. OPEX

Acknowledgments

I would like to thank my examiner Gerald Q. Maguire Jr. for helping me challenge myself to use my own knowledge. I could not be happier with the learning outcomes.
I also want to thank my supervisor Oliver Gindele for the support in this complex subject.
Lastly I want to thank the Bjango team for supporting me with the software iStat Menus 6 to measure energy consumption.

Stockholm, June 2022
Tom Axberg

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CSP	Cloud Service Provider
CV	Computer Vision
DNN	Deep Neural Network
DRAM	Dynamic Random Access Memory
FLOP	Floating Point Operation
FLOPS	Floating Point Operations per Second
FP16	Half-precision Floating-point
FP32	Single-precision Floating-point
FP64	Double-precision Floating-point
FP8	8-bit Precision Floating-point
GCP	Google Cloud Platform
GHG	Greenhouse Gas
GPU	Graphics Processing Unit
ICT	Information and Communication Technologies
ML	Machine Learning
NIC	Network Interface Controller
NLP	Natural Language Processing
OPEX	Operating Expenditure
OS	operating system
PLM	Pre-Trained Language Model
PTM	Pre-Trained Model

PUE	Power Usage Effectiveness
RAPL	Running Average Power Limiting
SDG	Sustainable Development Goal
SIMD	Single Instruction/Multiple Data
SOTA	State-Of-The-Art
TDP	Thermal Design Power
TEA	Techno-Economic Assessment
TPU	Tensor Processing Unit
UN	United Nations
VM	Virtual Machine
WLAN	Wireless Local Area Network

Chapter 1

Introduction

Making decisions is crucial to making progress and is usually an ambiguous action until the decision maker's impact has taken effect. The better the information is when making the decision, the higher the probability is of getting the desired result. Technological products do not always share the same characteristics, making every specific product decision heavily dependent on the particular technology. In this thesis, the focus is on Natural Language Processing (NLP) solutions from the perspective of fine-tuning, deployment, and inference. The information presented in this thesis should help decisions makers make the best decision when deploying NLP models.

The focus on sustainability in the Information and Communication Technologies (ICT) sector has never been greater than today as it has the opportunity to lower Greenhouse Gas (GHG) emissions by 20% by 2030 [1]. The sustainability pressure on the sector comes from various directions, *e.g.,* government policy, customer requirements, and the individuals' responsibility. As a result, engineers have to construct products that deliver a more sustainable world; thus, the engineer needs a new set of tools.

It all comes down to what the cost is, and if the cost is lower than the returns, then we have an opportunity to make good choices. Therefore, several questions arise: What metric gives a good understanding of the cost? Is the model that calculates this cost reliable, *i.e.,* reflects the actual cost? How accurate can we expect the model to be? We will address these questions in this thesis and try to derive a good approximation of the total cost of a classification task. We will see that the findings give the engineer the information needed to make better decisions.

The costs of an Artificial Intelligence (AI)-system life cycle usually consist of two parts: (i) the training of the model and (ii) the inference period. As

there has been much work done on the cost of training [2, 3, 4, 5, 6], this thesis will focus on the inference costs. To say anything about these costs, we will take a bottom-up approach, taking apart the cost regarding NLP applications to examine the variables. To start with, the costs of inference include (i) the monetary aspect, which we will express in U. S. dollars ($); this reflects the costs of using the hardware needed to execute the inference, and (ii) is the sustainable aspect that reflects the cost on the environment.

"Green-IT" introduced by Kern *et al.,* [7], will be used to give the term "sustainable" a calculable context. Green-IT includes the software level of the sustainable aspects of the NLP *application*. Kern *et al.,* pointed out that small changes in the way we choose to implement our software can have a considerable impact, as the product(s) using the software will be executed many times. "Green-IT" goes hand in hand with the term "Green-AI", which is the approach of yielding State-Of-The-Art (SOTA) results *without* consuming more computational power than already existing solutions [8].

1.1 Background

In 2017 the paper "Attention is All You Need" by Ashish Vaswani *et al.,* revolutionized the NLP area [9]. Their attention-based model was a simple network architecture that performed better with fewer Floating Point Operations (FLOPs) than previous architectures based on recurrent or convolutional layers.

We have seen an impressive development in the area of NLP in the last five years. From 2018, when Google introduced the Bidirectional Encoder Representations from Transformers (BERT) model, more transformer-based models are being developed [10, 11]. This development has allowed the industry to use language-specific Pre-Trained Models (PTMs), in this paper called Pre-Trained Language Models (PLMs), in their products, giving them the power of highly accurate models *without* high training costs[12, 11]. However, the cost of training these PLMs is huge and training one transformer model is estimated to generate about the lifetime CO_2 emissions of five average cars, including fuel. This training could cost around a million dollars ($) if the number of parameters reaches the billions [3, 4].

These costs described in the previous paragraph are big numbers, and the number of parameters used when training new models is continually increasing and are now in the hundreds of billions [13]. While we need to develop new models to tackle the need for more accurate models, the existing models are accurate enough for many applications. The transformer

architecture has generated a lot of good models, and for most tasks, there is no need to build and train a model from scratch. Using existing transformer models goes along with the Sustainable Development Goals (SDGs) goal 12 of the United Nations (UN) as the re-usage of PLMs has zero GHG emissions needed for training the PLMs.

The next phase is to deploy the model in an application. As of now, not many papers have taken up the discussion of how to calculate the costs of inference. One article does discuss the cost of inference when analyzing the number of FLOPs required to process one request/input [14]. We will look closer into these measurements Section 2.4.1.

To give the reader an overview of the AI-system life-cycle, a discussion with Datatonic was held to derive a series of good figures [15] to present this life-cycle.

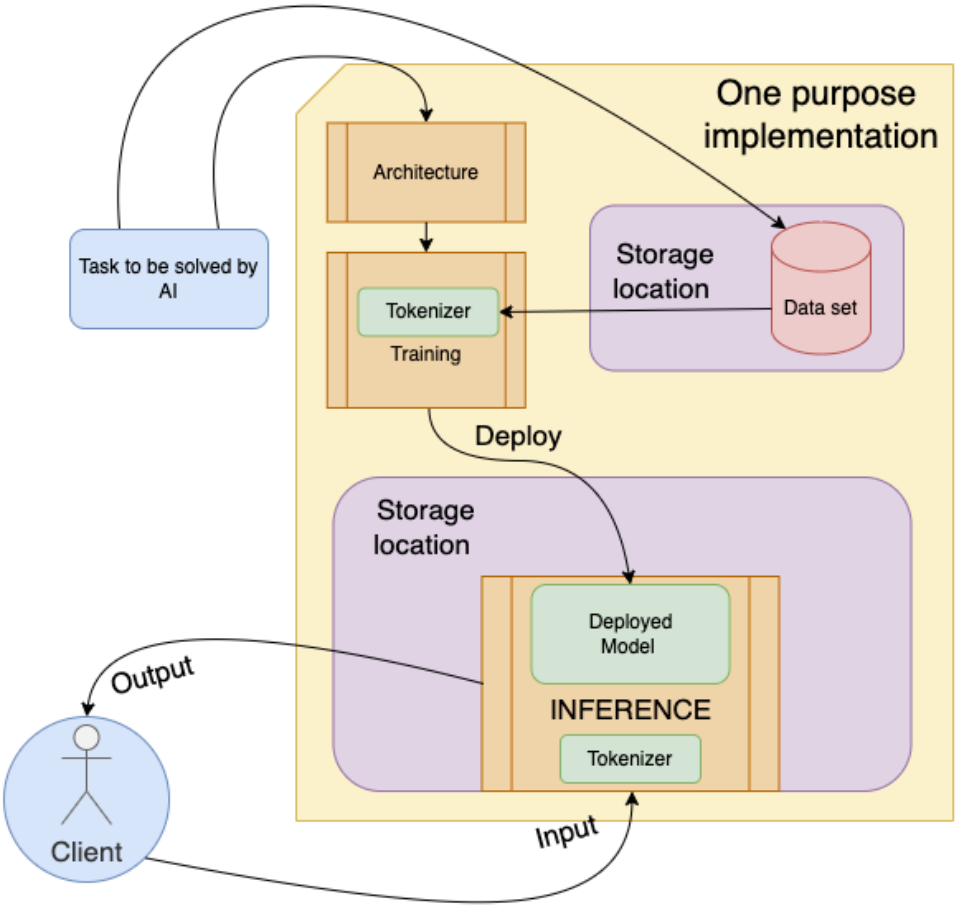

Figure 1.1: AI system life-cycle [15]

Figure 1.1 shows the main parts that constitute the life-cycle and their components when a network is trained from scratch for a specific task. This figure will help us untangle the inference phase when we explore how PLMs changed the way of developing AI-systems.

1.2 Problem

Before any investments, we want to make a cost analysis to understand what returns to expect at what cost. Developing an AI-system that meets the requirements regarding GHG and monetary cost is today a challenging assessment, especially if we look at the inference phase. Today, there is no standardized way of assessing these requirements, and those papers that estimate costs have shown little interest in this issue within the AI community. Therefore, this paper will focus on the inference phase.

A cost model will help developers deliver an AI system that meets both GHG and monetary requirements. To ensure present and future NLP topics are compatible with the derived cost model, we want to choose metrics that are likely to be consistent with the requirements. The technical aspects of the metrics will not be discussed in detail, but will be presented enough to say something general. Further findings will be discussed in the end of the thesis.

The question then arises, what are consistent metrics? How can we ensure that the cost model is compatible with the whole NLP area? Answer these questions is the key to deriving a reliable cost model. Based on previous articles and by conducting empirical studies, answers to these questions will be researched.

1.2.1 Original Problem and Definition

The original question concerned the company Datatonic's interest in developing solutions that solved several problems in NLP, specifically summarization and classification. Datatonic had researched the area, the NLP models to use for summarization and classification were defined, and deployment was the next step. As the goal was to give their clients high-end solutions, we expect that the company's goal was that the accuracy of the output of these NLP should be as high as possible.

Several questions arose when defining the problem of this thesis further. A key concern was sustainability. The thesis problem is defined to consider the sustainability and monetary cost of implementation and how these costs changed with technical choices made during the implementation. The problem

expanded to understanding the environmental impact of developing NLP solutions, which might have a GHG as well as monetary cost attached to it.

1.2.2 Research Question and Engineering Issues

Green-IT is a term that is spreading, and this concept helps software developers practice sustainable development [7]. There are many aspects of sustainable development when developing software solutions. As an engineer, you cannot just write a program that works – as even small changes in the code can have a significant impact due to the scale-ability of software. Therefore, the engineer can make a substantial impact by suitable designing and editing the code with sustainable development in mind [7].

Schwartz *et al.,* coined the terms *Green-AI* and *Red-AI* in 2019, where they went into depth about what differs between them [8]. When this thesis was written, others had already cited their work 377 times, which indicates that the terms gave value to the AI community. Roughly put, the terms differ in their aim when achieving SOTA results. Green-AI aims not to increase computational costs and ideally reduce them. In contrast, Red-AI does not consider anything other than getting the highest accuracy possible – *no matter the price* [8].

To ease the sustainability concerns previously mentioned and examine the monetary effect of lowering the computational cost, we state the research question as follows:

> d*How can a reliable cost model be developed to give a good overview of the monetary and GHG cost of a given AI (specifically, a NLP model that does inferencing) system's implementation and deployment to enable Green-AI development?*

1.3 Purpose

Since the Transformer model was introduced in 2017, we have seen an impressive development in the NLP area. The SOTA has advanced more and more, and today most transformer PLMs are better than humans on the current NLP tasks. The price we have paid to get this rapid growth is to ignore the sustainability aspects the world faces. To be able to meet the UN's SDGs we need a shift in how stakeholders develop and use AI systems, especially in the area of NLP.

Other Machine Learning (ML) areas appear not to have similar escalating costs attached, as that of NLP, which Sharir *et al.,* discussed [4]. There is a common way of addressing these costs, dividing the application into the training and inference phases. Nvidia has stated that this latter phase contributes 80-90% of the cost [16]. These numbers make sense as the period of using the model is substantially longer than the time spent training it.

Datatonic, a leader in cloud data engineering, advanced analytics, and machine learning solutions, is becoming more interested in sustainable development as a new initiative (Greentonic) in the company has begun. Datatonic's products need to be efficient, accurate, and sustainable to deliver the best solutions for their clients. To meet these requirements, assessments of these aspects need to be as precise as possible for each implementation as each product is unique.

New tools to assess the costs of ML-based solutions are needed as the sustainable development problems of the ML sector are evident [3]. A cost model is a good tool for predicting the costs for a set of metrics, and this thesis aims to derive a suitable cost model by making the inference phase as cost-efficient as possible.

If the goals in the next section (Section 1.4) are met, the thesis will contribute with benefits to the AI-community, and Datatonic's engineers and product owners. In Datatonic's pursuit to lower GHG emissions and monetary costs for their clients and the world, this tool is going to make a difference – as no such tool exists today.

1.4 Goal

The goal of this project is to create a cost model for NLP deployment. The goal has been divided into the following three sub-goals:

Subgoal 1 Derive a cost model that mainly focuses on the energy cost. As sustainability is a key topic of this paper, we will focus on having a general energy cost model. This model will predict the energy used by the application during the deployment period with chosen metrics. The aspects focused on in this stage will mainly be [11]: model choice, tokenizer choice, and inference period. We will perform a series of tests to see if the chosen metrics are suitable.

Subgoal 2 When a general model has been found (*i.e.,* Subgoal 1 has been achieved), the next step is to look at how deployment affects it,

i.e., the storage of components needed to construct the application. How are we storing the data set? Do we have the application on-premise or on a cloud service provider? We will also test the floating-point precision effect explained in Section 2.4.2 and the impact of the deployment code *i.e.,* the web framework used.

Subgoal 3 The last step is to include the monetary aspects and to construct a Techno-Economic Assessment (TEA). The cost model will now have a model to translate energy cost to dollars ($), and we will now analyze the aspects described above with a price in dollars. We will now have a cost model that gives an approximate total $ cost of the deployed application. With this model, we will discuss the advantages and disadvantages of having the application deployed in the cloud from both energy and monetary aspects in Chapter 6.

1.5 Research Methodology

To derive a reliable cost model, quantitative and analytical research is used with a bottom-up approach for measuring costs in the cost model. With the bottom-up approach, we break down the deployed application to find inference cost source(s).

As a result, a good understanding of how a cost model applies to NLP will be gained. We will also find if the measurement techniques lacked the precision needed to be valuable to the project's stakeholders.

A cost model could be derived strictly using the current methods and techniques with qualitative research alone. Computer Vision (CV) has already made progress in this field and developed efficient approaches to lower model costs. However, ML model architectures are different in different areas, and how these models execute in hardware is not the same. The number of instructions for two compiled models with different architectures can vary, but their energy usage may differ. In the area of NLP we have not found the same research as in CV; therefore, we choose to use a quantitative methodology in addition to a qualitative methodology.

1.6 Delimitations

Deriving a cost model is limited to the areas accessible for measurements. If we had all data concerning what the cost model *should consider*, we could do

a regression model and map the new inputs to that model. However, we lack some of the relevant data, as will be further described in Sections 1.6.1 and 1.6.2.

1.6.1 Clear-Cut Limitations

Regarding NLP, calculating monetary and energy costs have three clear limitations, all connected to hardware: (i) the ability to measure hardware performance in a cloud environment is limited, *i.e.,* measurements of the hardware are not possible in the same way as our on-premises setup; (ii) the ability to measure energy precisely on hardware is limited to hardware support, and we will only study Intel's processors: and (iii) hardware optimization optimizes the execution of the instructions, making it hard to use straightforward metrics such as the number of instructions; therefore, we will not study how instructions are executed and instead consider metrics that give us a correlation between instructions executed, starting with FLOPs, and energy consumption.

1.6.2 Memory Limitation

The components that are significant to energy consumption when performing inference are: Central Processing Unit (CPU), Graphics Processing Unit (GPU), and memory. The efficiency for the first two processing units is expected to have a linear correlation between energy consumption and efficiency for a chosen inference deployment platform. Memory on the other hand is a bit trickier, as the CPU and GPU do not use the same memory and we will not be able to measure some of the aspects of the GPU's memory as this memory is integrated on the GPU board. To avoid making the cost model too technical to use, the CPU memory's energy consumption will be measured together with the CPU energy estimation.

1.6.3 Machine Idle Time

When a machine, usually a server, does inferencing, it will be idle sometimes. This time is hard to predict as it will be directly connected to the specific hardware that constitutes the machine. As most machines are unique regarding their components, the idle time energy consumption is as well. Therefore, idle time energy consumption will not be regarded.

1.6.4 Deployment Location

Lastly, the deployment location plays a substantial role when deploying an AI-System as the energy used in different areas is more or less derived from renewable sources. This thesis will not consider this fact and leave it to the developer to choose the deployment location.

1.7 Thesis Outline

Chapter 2 presents relevant background information about PLMs and the progress of cost modeling in other areas, such as CV. Illustrations will be given to explain the AI-system of concern and how PLMs fits into such systems. Further, we focus on the part of the AI-system that affects the inference phase and how to measure the energy consumption of its components. Related areas close that chapter together with a summarization of the chapter.

Chapter 3 presents the methodology and method used to solve the problem. We present tools and frameworks to evaluate the results from solving the problem and system documentation.

The data collection is presented in Chapter 4 as well as the code used to collect data.

Chapter 5 goes through the results and analyze them with suitable Figures. A brief discussion is presented with each analysis to state key findings, which we will further discuss in Chapter 7.

Chapter 6 presents the areas which have been an obstacle in accomplishing all sub goals presented in Section 1.4. We look at why the areas have been problematic and how we would benefit if they were more explored.

Lastly, in Chapter 7 we present the key results and answer Subgoal 1. We connect to Chapter 6 and bring up areas regarding future work to cover Subgoal 2 and Subgoal 3.

Chapter 2

Model Deployment

The phrase "A picture is worth a thousand words" was probably used well before the birth of the first computer. Yet it is a powerful phrase that still shows its deep meaning in modern times.

Studies have been made on making CV models green and efficient [17, 18]. One might wonder why the research on CV has not affected the area of NLP in the same way, and this has been discussed by Sharir *et al.,* [4].

The Convolutional Neural Network (CNN) model's architecture is well suited to processing images, as it uses all pixels to accumulate information and all pixels affects the result [19]. In contrast, as Sharir *et al.,* states NLP it is a bit harder as the text to be processed is not processed in the same way as a picture [4]. For example, images are local and smooth as the neighbouring pixels values do not change much; while words are drastically different from one another and do not share the same characteristics as pixels do. Therefore, we cannot simply put a thousand words into a frame and process it as an image.

To be able to get contextual information from the surrounding words, similar to data from surrounding pixels in an image, the transformer model was developed [9]. The transformer model has been the SOTA architecture in the area of NLP for several years but is different than CNNs and more costly [4].

If we are to meet the UN's SDGs we all need to take new approaches regarding how we make decisions and act upon those decisions. Decision makers and stakeholders need new tools to make better choices in various areas and ML is one method that is becoming widely used to provide better tools in many application areas. Training one large ML model, specifically an NLP model, has a substantial impact on the environment as the GHG emissions reaches the CO_{2eq} of flying one passenger from New York to San Francisco [3].

Sharir *et al.,* made a substantial contribution to understanding the cost

associated with training large NLP models [4]. They stated that training a model with billions of parameters can reach costs of millions of dollars ($). As their paper only considers the cost of training the model, they state that there is a whole other discussion needed about the cost of these NLP models at inference time.

The inference discussion was later brought up by Desislavov, Martínez-Plumed, and Hernández-Orallo where they cite Thomas, 2020, to estimate the inference cost as 90% of the AI-system costs [14, 20]. This occurs as training is done once but inference is done repeatedly. Desislavov, Martínez-Plumed, and Hernández-Orallo have also observed that there are few inference focused papers regarding the associated energy consumption and the costs. In contrast, this thesis will directly address this phase. Moreover, as this thesis is interested in examining the GHG emissions and cost of deploying PLMs, this thesis will explicitly take up this discussion.

2.1 The Transformer Model and PLMs

PLMs have been used since 2006 when Hinton and Salakhutdinov had their breakthrough with greedy layer-wise unsupervised pre-training[21]. They showed that a model could be trained without supervision and later fine-tuned to a specific task with a supervised training step. Moreover, this fine-tuning step is far less computationally expensive than the unsupervised training phase [21].

Languages are complex and to capture the linguistic rules and common sense knowledge that hides in text data requires Deep Neural Networks (DNNs) models with a large number of parameters. To train these large models is costly, but using a PLM drastically reduces the training cost. A comprehensive review has been done on the area of PLMs, and we will use models presented by Que *et al.,* [12].

The use of PLMs is illustrated in Figure 2.1. This is one of a series of figure derived through discussion with Datatonic [15]. This figure shows how the PLM is used in the AI-system and how this differs from Figure 1.1. With the inference phase separated from the training phase, we can discuss the inference components (see Section 2.2).

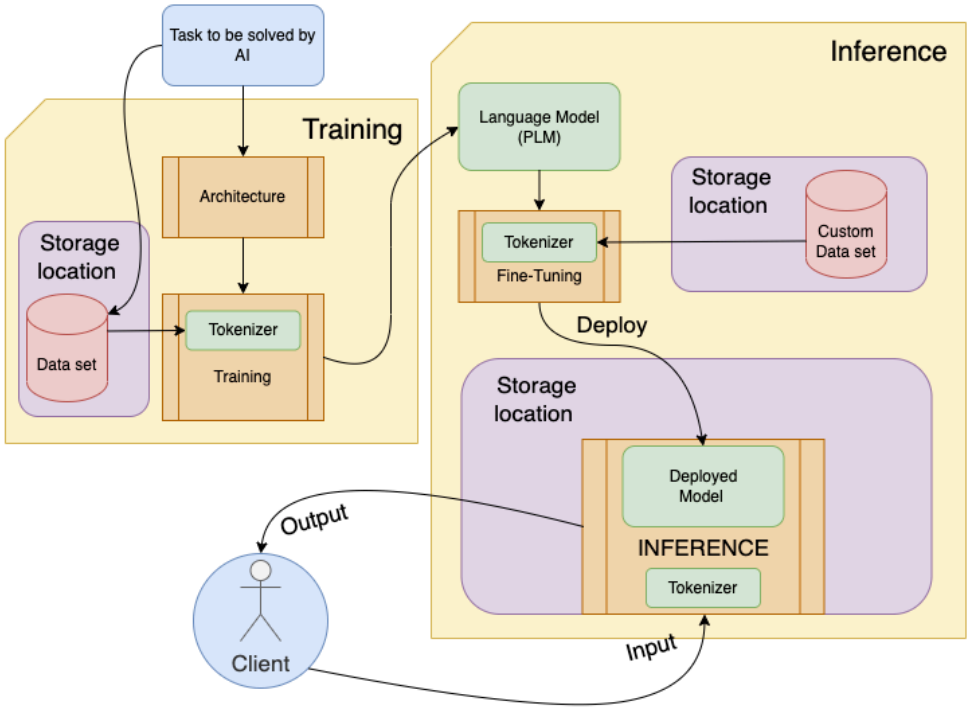

Figure 2.1: AI system life cycle with use of PLM [15]

As stated earlier, we will focus on the transformer model as it had substantial impact in the field of NLP, and many transformer models are available to the public [9]. When a CNN processes an image it processes areas of pixels in such a way that every pixel affects the final result. With the transformer model the results is similar, as words are processed in parallel which gives them context to each other. This is similar to when you have a discussion with someone and both people are talking about something that is implied. Humans know what is implied but it does not make sense for a machine without attention to context.

Huggingface has gathered most SOTA PLMs available to the public to make them easy to access [22]. The models that will be explored in this thesis [10, 23, 24, 25] are all accessible there.

2.2 Inference Components

Separating the concerns regarding training and inference is a natural way of partitioning AI-system's costs. Figure 2.1 shows the components necessary

for deploying the AI-system while Figure 2.2 shows the inference portion of AI-system, *i.e.,* the components that constitutes the inference system.

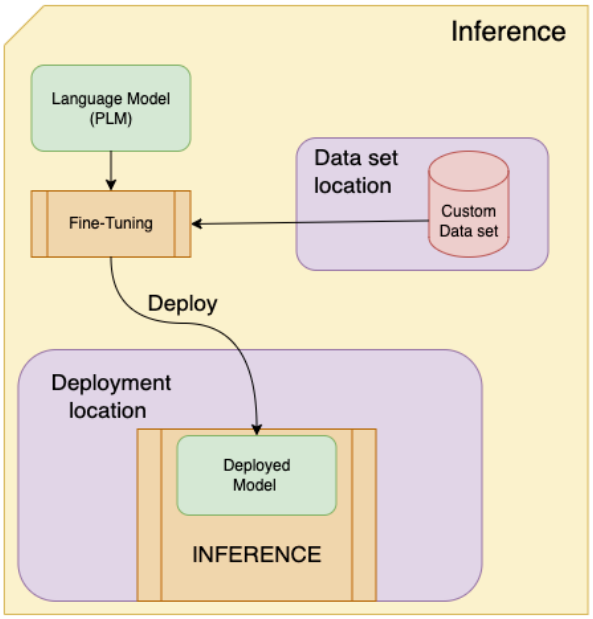

Figure 2.2: Inference system [15]

As we are interested in calculating the cost and the associated energy consumption, this cost and energy consumption will be divided into three logical areas: (i) the preparation process where a tokenizer and optimizer are selected, and the selected PLM fine-tuned; (ii) the custom data set, stored at a location, that will make the PLM more suited to solve the task at hand, and (iii) the deployed model that will handle the inference process. All these three areas affect the cost and energy consumption; therefore, all will be treated separately in the following sections.

2.2.1 Preparation and Fine-tuning

In this step the focus is selecting the PLM and tokenizer for an AI-system. There are many PLMs available and some will be examined in this thesis. Each PLM transformer model comes with the tokenizer that the model was trained with. This tokenizer ensures that the tokens generated are compatible with the model. If a tokenizer, other than the one used in pre-training, is used then one should expect inconsistency. Huggingface have made consistent fast tokenizers available in their Application Programming Interface (API) that

are written in the functional programming language Rust. As the tokenizer is used to encode/decode incoming requests, seeing its impact on the total cost is interesting.

Huggingface can use different optimisation tools to fine-tune the model. There are many optimization options when performing fine-tuning, but we will not consider different optimizers in the data collection.

2.2.2 Data Set and Maintenance

The custom dataset shown in Figure 2.2 is usually updated as new data become available. Retraining the model is necessary to deliver up-to-date results, but how often retraining is done varies from project to project. In discussion with Datatonic, A. Nieto and I derived four metrics that measure this component [15]:

1. the dataset size as storage costs over time,
2. the storage location, which influences the storage cost,
3. there is a cost attached to the retraining computation
4. this retraining cost, scales with the retraining interval.

The retraining computation will be measured in FLOPs, which is the same unit in the fine-tuning.

2.2.3 Inference

Once the model is fine-tuned it needs to be deployed somewhere to handle inference requests. This deployment can be done on-premise or in the cloud.

As the code that handles the inference is kept to the necessary minimum, the code size is generally negligible, assuming that the engineer develops code from the perspective of ”Green-IT” and ”Green-AI”. The tokenizer chosen earlier will be the same tokenizer used during the inference phase.

The metrics we will use for this part part of the cost model are the number of FLOPs for the selected PLM when executing a request, and the efficiency in Floating Point Operations per Second (FLOPS)/Watt for the selected GPU. The last and the thing that will most affect the cost is the inferencing period (*i.e.,* the period during which the PLM is performing inferencing). It is the long period of performing inferencing to execute requests that leads to Thomas' estimate of the inference cost as 90% of the AI-system costs [14, 20].

2.3 PUE

The Power Usage Effectiveness (PUE) is the industry preferred metric to understand the energy efficiency of an IT facility [26]. Equation 2.1 describes the efficiency.

$$\textbf{PUE} = \frac{\textbf{Total Facility Energy}}{\textbf{IT Equipment Energy}} \tag{2.1}$$

From Equation 2.1 we see that we get a multiplayer that we use to calculate the real energy consumption of the server facility used. When the cost model has predicted the energy consumption, we simply multiply the result by the **PUE** to attain the energy consumption prediction.

2.4 Cost and Associated Energy Consumption

Although little work has been done on analysing inference costs, there has been research on the reporting of costs and its metrics regarding ML [3, 8, 14, 27]. Henderson *et al.,* [27] worked on standardising the reporting of ML carbon footprint. This work has had an substantial impact*. Their work considers the full AI-system and their proposed standardization will be used in this thesis.

2.4.1 How to Measure Energy Consumption

The intuitive measurement unit, which is used in other studies to measure the work performed by an AI-system is FLOPs [8, 14]. The more FLOPs a program executes, the more work it performs, one might think. However, this conclusion is not always accurate as other parameters affect the total energy consumption. Henderson *et al.,* conclude that the model's architecture plays a significant role in energy consumption and that using FLOPs as a metric of energy consumption between different model architectures is **untrustworthy** [27].

Desislavov, Martínez-Plumed, and Hernández-Orallo mention and agree that the conclusion by Henderson *et al.,* is correct, but while FLOPs as a measurement of energy consumption is unreliable, this metric performed well in their study of *trends in energy consumption* [14, 27]. They also emphasize that a more accurate study would need to measure the energy consumption of

* Their paper has of now been cited almost a hundred times (97) according to Google Scholar

the individual inference instances and that this is rarely reported. Therefore in this thesis, the energy measurements use the methods of using FLOPs described by Henderson *et al.,* [27].

To end this section, we present two equations from the paper by Desislavov, Martínez-Plumed, and Hernández-Orallo, as these equations will be used later in Chapter 3. Please see Equations 2.2 and 2.3, and notice the difference between FLOPs, the total number of floating point operations, and FLOPS, floating point operations per second. The Watt variable refers to the processing unit's Thermal Design Power (TDP), which is its maximum generation of heat expressed in Watts when operating at full utilization. Note that 1 watt is $1\,\mathrm{J\,s^{-1}}$.

$$\text{Efficiency} = \frac{\text{FLOPS}}{\text{Watt}} = \frac{\text{FLOPs}}{\text{Joule}} \tag{2.2}$$

$$\text{Energy} = \frac{\text{FLOPs}}{\text{Efficiency}} = \frac{\text{FLOPs}}{\frac{\text{FLOPs}}{\text{Joule}}} = \text{Joule} \tag{2.3}$$

2.4.2 The Effect of Throughput and Precision

Floating point operations per second, *i.e.,* FLOPS, is a measurement of throughput, as it is the number of floating point operations executed by the hardware per second. Equation 2.4 shows how to calculate FLOPS. This equation will be necessary when deriving the efficiency calculated by Equation 2.2.

$$\text{FLOPS} = \text{cores} \cdot \frac{\text{cycles}}{\text{second}} \cdot \frac{\text{FLOPs}}{\text{cycle}} \tag{2.4}$$

The efficiency e_x for a processing unit x will have a substantial impact on the energy consumption, as a higher throughput (FLOPS) gives higher e, which will result in more work done per unit of energy, as shown in Equation (2.5).

$$\text{joules} = \frac{\text{FLOPs}}{e_x} \tag{2.5}$$

The hardware efficiency is not the only variable that affects the total energy consumption for a deployed AI model. FLOPS can occur with different precision – with the typical precision being Double-precision Floating-point (FP64) or Single-precision Floating-point (FP32). Of these two, the later is the most common precision. Today's GPUs and Tensor Processing Units

(TPUs) support Half-precision Floating-point (FP16) processing and are able to theoretically process two FP16 in the time it takes to perform one FP32 when using Tensor-Flow's mixed-precision [28]. Nvidia just released their new H100 GPU that is the followup from their earlier A100 GPU [29]. The H100's Hopper architecture supports 8-bit Precision Floating-point (FP8) processing and can perform FP8 computations with a throughput of 4000 TFLOPS compared to the A100's FP16 throughput of 78 TFLOPS resulting in a $6.4\times$ speedup [30]. Although, the H100's TDP is 700 W compared to the A100's 350 W, this still results in higher efficiency for the H100. For an efficiency comparison see Table 2.1 calculated with Equation 2.2.

Table 2.1: GPU Efficiency (TFLOPs per joule). 'NA' stands for 'Not applicable'

FP Format	**A100**	**H100**
FP8	NA	2.86
FP16	0.89	1.43
FP32	0.46	0.71
FP64	0.06	0.09

2.4.3 Monetary vs. Environmental Cost

In Section 2.4.2 we saw how a modern GPU improves efficiency. More modern GPUs drive down the energy cost but at a monetary cost. The expected price of the Nvidia H100 is not yet released, but we expect it to cost more than the A100 as the price of the A100 has already decreased by 50% [31].

Additionally, the cost will vary depending on the choice of cloud versus on-premises deployment. Data from Google for utilizing a set of Nvidia Tesla GPUs offered in Europe and the efficiency for single-precision FP32 are shown in Table 2.2 [32, 33]. These prices are based upon Google's Vertex AI service, which is a platform developed to help teams deploy AI models [34].

Table 2.2: Table describes cost per hour for GPU along with their Efficiency

GPU model	Price per hour ($)	Efficiency
A100	3.3741	0.89
T4	0.4025	0.12
P4	0.7475	0.08
V100	2.9325	0.06
P100	1.8400	0.04
K80	0.5635	0.03

2.5 Related work area

This section presents areas that indirectly solve the stated problem through other technologies. No other area tries to solve our problem, but the underlying wish to lower costs can be achieved in these areas.

2.5.1 Edge or Server-less Computing

In this thesis we mainly focus on looking at the costs needed when making inference. The idle time energy cost for that server is not considered as it is nothing the implementation of inference solution can change.

Lowering energy costs by using another type of server technology is however possible, and with Edge computing it can be achieved. The server only uses the resources to handle the inference request, and then simply gives the resources to the next request from a different client.

This is an interesting area to follow, but will not be of interest in this thesis as discussed above.

2.5.2 AI Hardware Chips

More chip-makers design chip architectures to specifically execute ML models as efficiently as possible, reducing the energy cost of execution.

In March this year, Nvidia released their new H100 GPU [29]. Its transformer Engine, part of the new Hopper architecture, will have a substantial effect on NLP training performance. As the inference model needs to be updated with new data, the model needs to be occasionally retrained. With dedicated AI chip-sets, we will get more cost efficient training.

2.6 Summary

To summarize this chapter, we will look closer at the progress made in other areas and the current development of measuring and accounting for ML model costs. In Table 2.3 we see the current solutions that tries to solve the energy accounting problem.

Table 2.3: Table of current solutions

Solution	**Comment**
Intel® Power Gadget [35]	Software that accurately measures energy consumption on systems with Intel CPU.
MLco2 [2]	Rough estimator with too few metrics. Good for understanding the effect of choosing a cloud service provider and region.
Experiment-Impact-Tracker [27]	Feasible tool to be tested. Measures energy cost of the whole system.

We could not find anything similar to a cost model for inferencing. We will therefore start with the objects in Table 2.3 to then expand beyond them. We will start with FLOPs as an energy predictor as explained in Section 2.4.1. It is not before we have established confidence in metrics to predict energy that we can look at the effect of other factors which are non-model specific.

In the coming chapters, we will establish if FLOP is a good predictor for energy consumption and examine the best ways to measure the energy consumption of an AI-system.

Chapter 3

Research Methodology

The purpose of this chapter is to provide an overview of the research method used in this thesis to answer the research question. Section 3.1 describes the research process. Section 3.2 details the research paradigm used to derive results. Section 3.3 focuses on the data collection techniques used for this research. Section 3.4 describes the experimental design to replicate the model data. Section 3.5 explains the techniques used to evaluate the reliability and validity of the data collected. Section 3.6 describes the method used for the data analysis. Finally, Section 3.7 describes the framework selected to evaluate the underlying cost model data.

3.1 Research Process

This thesis explores a complex area that is growing fast. The research question is dependent on both hardware and software, and both areas have made progress during the writing of this paper. It has been hard to explore all of the components that affect the cost fully, but with an extensive literature study, the area will be suitably covered. As some parts of the research area have not been explored in previous papers, the research strategy was to map out the unexplored areas and piece them together with what has been found. In Figure 3.1 the four research process components are presented.

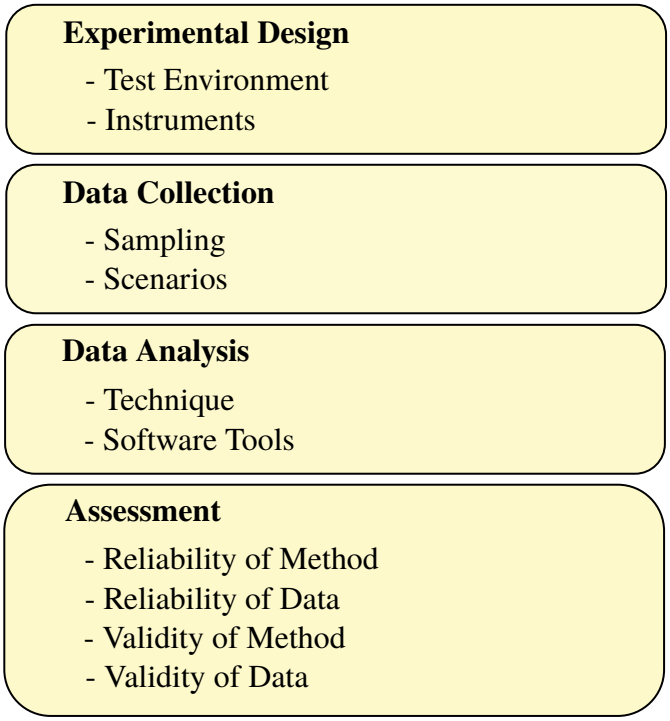

Figure 3.1: Research Process

3.2 Research Paradigm

We use the positivist research paradigm as this research project heavily depends on quantitative methods. We can only answer the research question by making measurements of actually deployed implementations of an inferencing engine. The set of these measurements was developed in regards to 'Green-IT' as mentioned in Section 2.2. These quantitative measurements will be relevant as a deployed implementation of an inferencing engine uses computational resources to execute inferencing requests.

3.2.1 Quantitative Research

A quantitative research methodology is used with a bottom-up approach. In the end, the cost model will be a simple calculation based upon a number of inputs. This bottom-up approach will help us gather data by testing different component configurations. The configurable components are shown in Table 3.1.

Table 3.1: Configurable Components

Component	Parameter
Model	Architecture
Tokenizer	Fast or Base
GPU	Efficiency
Precision	FP16 or FP32

3.2.2 Discarded Paradigm

A pragmatic view was considered as the AI model's code is dependent on the engineer. This pragmatic view should help us understand how the model's code is usually developed. Discussions with developers at Datatonic revealed that the deployed code is always kept to a minimum [15], and is only dependent on the web framework used to make inference.

3.3 Data Collection

The data collected consists of data points that describe energy consumption for three scenarios explained in Section 3.3.2. The scenarios will mimic inference periods and the data will later be processed and analysed to find a correlation between the work done by the inference machine and energy consumption.

The data collected is not connected to any specific social group and ethical concerns are not a factor in the collection. Although, client of Datatonic were the subject of the scenarios developed in Section 3.3.2, the names of these customers are kept confidential; hence, we can not map these scenarios to any specific organization.

3.3.1 Sampling

The area of NLP has seen a variety of different model architectures emerge. The SOTA architecture currently is the transformer. Due to time limitations the sampling will only consider a set of popular BERT based transformer architectures. These are listed in Table 3.2.

Table 3.2: Table of Transformer Models and Parameters [10, 23, 24, 25]

Model	**Parameters (millions)**
RoBERTa: A Robustly Optimized BERT	130
$BERT_{BASE}$: Pre-training of Deep Bidirectional Transformers	110
DistilBERT: Knowledge Distilled BERT	66
$ALBERT_{BASE}$:A Lite BERT for self-supervised learning	12

We want to find a correlation between models with similar accuracy but with different architectures. Therefore, we selected the models in Table 3.2. We can see by the number of parameters how they differ in size, which indicates their architectural differences.

3.3.2 Inference Scenarios

Hardware consumes energy when when it is idle; therefore, we need to take idle time in consideration when developing scenarios. A steady stream of requests without idle time would result in hardware working at all times with maximum energy consumption. The model requiring the least number of FLOPs is theoretically the best model as more requests can be handled per time unit with a given processor. Therefore, as previously mentioned, we will use models that gives high accuracy with a large custom data set.

Real life requests are tricky to model as requests arrive at random times. Therefore, the sample size and time duration is hard to determine, but in discussion with Datatonic three scenarios were selected based on a seeded random number generator to generate request arrivals [36]. The scenarios all handle between 100 and 500 requests. When all requests have been handled we stop our measurement as the scenario is done. In total we will have 1100 text classification tasks in 511 requests.

Scenario 1 A deployed language model for handling requests supporting software such as Apple's Siri is highly unpredictable. The service is used around the clock globally. This scenario will mimic this kind of deployment by sending 500 requests with a random idle time in the range of 0 ms to 1000 ms.

Scenario 2 Every morning, a news websites wants to gather all the latest news and classify the new items into different categories to make them easier to search. A news classifier service is used. The service

will get 10 articles to classify per request. There are 10 news websites that want to use this service. A total of 100 texts are processed in 10 requests.

Scenario 3 A stock trading firm wants to classify the latest news from different company analysis websites. They gather the top 500 news items to classify for further analysis. All 500 texts are sent in one request.

These scenarios will all be measured and average energy consumption per request for the scenarios will be derived and used in the cost model. In the discussion with Datatonic, it was established that it is hard do estimate the amount of requests. Therefore, we use the scenarios to mimic the various requests to derive an energy consumption estimation per request.

3.4 Experimental Design

A on-premises setup is used to monitor energy consumption. To minimize noise when measuring energy consumption, we turn off all components on the computer that could affect the energy consumption. We use scenario based tests and log energy used by our CPU, DRAM_{cpu}, and GPU.

Inference requests are sent by an external microcomputer connected via the local Wireless Local Area Network (WLAN). This is done to not interfere with the energy measurements and to mimic incoming requests coming from clients. The experimental setup is further explained in the following subsections.

3.4.1 Test environment

The test environment consists of two computers. One to process requests and one to generate them. The computer handling inference needs to be a Windows or a MacOS machine with an Intel©processor that supports RAPL [37]. Linux machines that supports RAPL can also be used, but the new version of Intel® Power Gadget that uses RAPL is not available for Linux. An older version is available for Linux machines, which has not been tested or used in this experiment.

The Bash script found in Appendix A is needed to replicate the scenarios used. The script can be executed from any machine that supports a Unix bash shell. The machine in question also needs to be connected to the local network

to be able to send requests to the inference handling machine described above. As we are not considering idle server energy consumption, any connection is sufficient. The only energy consumption we are interested in is the one on the machine handling the request.

Regarding the inference machine, all software that is not relevant to the experiment should be turned off, for example Bluetooth service, mail client, and the web browser. The software used on the inference machine is described in Section 3.4.2.

All models were fine-tuned on the Vertex AI platform on Google Cloud Platform (GCP). As no on-premises GPU was able to perform the fine-tuning of all the models, we will test fine-tuning with two models on the on-premises machine and see how well it fits the equation in Section 2.4.1.

3.4.2 Hardware/Software to be used

As the primary operating system (OS) used by Datatonic is the Darvin based macOS running on Apple computers, the test environment used was set up to find feasible tools that worked for the OS of interest for the stakeholders. The on-premises hardware monitoring was conducted with the software Intel® Power Gadget[35], and Istat Menues [38]. The first program monitors energy usage by the CPU and Dynamic Random Access Memory (DRAM), and the second program monitors the GPU.

The measurements were conducted with the request processing on an Apple MacBook pro 2017 using Intel® Core™i7-7920HQ CPU @ 3.10 GHz with 16 GB of 2133 MHz LPDDR3 memory with a AMD Radeon™Pro 560 4 GB GPU. The requests were sent over a local wireless 2.4 GHz connection by a Raspberry Pi 3 Model B+ (1.4GHz 64-bit quad-core processor, dual-band wireless LAN) that executed the Bash script in Appendix A.

The programming language used to deploy models was python. The choice of python was made as the Hugging Face python package gives good support to handle transformer models together with Tensor-Flow and is relativity easy to use [22]. Python also enables us to use TensorFlow and tensorflow-metal PluggableDevice, which is a macOS specific package to accelerate training on the discrete GPU on the machine in question [39]. The test environment used the python packages listed in Table 3.3. A description of how to set up the python environment is given in Appendix B as a requirements.txt file. The essential packages are shown in Listing B.1.

Table 3.3: Table of python environment packages and version number

Package	**Version**
datasets	2.1.0
Flask	2.1.1
huggingface-hub	0.5.1
keras	2.8.0
numpy	1.22.3
pandas	1.4.2
pip	21.2.4
tensorflow-macos	2.8.0
tensorflow-metal	0.4.0
tokenizers	0.12.1
torch	1.11.0

3.5 Assessing reliability and validity of the data collected

To avoid making the cost model too complex to use and to keep it in the context of interest, we will **not** include idle server energy costs. By subtracting the idle time energy consumption from the total energy used, we will find the energy usage of the deployed NLP model. Idle energy usage is unique for a specific server and will not be examined further in this paper (however, the idle time energy usage of the test environment will be measured and reported in the thesis). The energy data collected will then be mapped to work done by the NLP model using a regression model.

The validity of our data points is directly affected by the software used to make the measurements. There are two programs used to collect data, one for measuring energy and one that counts FLOPs. Our energy measurements are considered reliable as the methods used were promising, while FLOPs counting methods were found not to be supported on macOS. Third party options were available, but none worked.

As every server is unique, regarding the hardware components installed, it is close to impossible to predict an accurate idle time energy consumption without using a lot of input metrics to the cost model. Therefore, we will assume that readers measure the idle energy consumption of their server machine, with for example Intel® Power Gadget or a power meter, and then add this to their energy from the cost model. The data collected will then be processed to subtract the test environment's energy consumption, and the

systems' energy consumption when performing inference will be reported.

This section describes the measurement methods and assesses their reliability and validity. Additionally, we will explain what methods we used and discarded and why we discarded them.

3.5.1 Validity of Method

The method used for the hardware cost is simple, we sum up the cost of purchasing or renting the necessary hardware. As there are no previous studies found regarding NLP deployment, there are no existing values to validate our final cost model against. Energy estimation is the key issue, and the method is as valid as the tools used for data collection. These tools are described in Section 3.5.5 along with how the energy values were processed after collection.

As noted earlier, to avoid making the cost model too complicated, we will subtract the idle time energy consumption to isolate the energy used by the NLP model. If details of the idle time energy consumption were to be considered, we would have to deal with other factors such as machine temperature and power consumption of hardware components other than the ones executing the work needed for inference.

To see if the methods used to collect the data can be used used to derive a valid cost model, we will validate the cost model with new data and assess the model predictions by comparing them with the Google cloud cost calculator and their cloud carbon-footprint estimator [40, 41].

3.5.2 Reliability of Method

We repeated our data collection method three times for each model with each of the scenarios in Section 3.3.2. As noted above, the idle time energy consumption is subtracted from each measurement to isolate the energy consumption of the NLP model. We can ensure that the method is reliable if we can reproduce our data collection and get similar results when analyzing our data described in Section 3.6.

3.5.3 Discarded Method

Counting FLOPs is a problem on its own. In the literature study phase, the closest method to accurately calculate FLOPs found is by using Performance Monitoring Unit (PMU) events/counters or the Intel® Software Development

Emulator (Intel® SDE)*. These methods were not used due to time limitations. However, if this study were to be replicated, we would encourage others to use these tools.

3.5.4 Data Validity

In the first stage of testing the validity of measuring energy and counting FLOPs we repeated the data collection three times. We assessed the variation in results from these three repetitions. In the second stage, we used a transformer model that was **not** part of the data collection used to construct the cost model. This transformer model will give us new data that we will use to validate the output from the cost model.

3.5.5 Reliability of Data

Regarding energy measurements, the RAPL support gives us energy data about the CPU and DRAM that we can consider reliable as previous tests by others have shown that it gives inversely proportional results [37]. Regarding the GPU we use the iStats Menue software, which does not reveal how watts are measured. A request to the software developers was made to assess the validity. The answer was that the software just reports what the sensors shows on the machine, and that no specific testing has been made to assess if the values match what's happening on the hardware level [42]. The watts shown are in the range of what the GPU power should be (based upon the hardware specification), and we will assume that it is reliable.

Henderson *et al.,* have developed a measuring tool that should work for macOS [27], but the software did not yield any output when tested [43]. A test of the energy measuring software of Henderson *et al.,* was conducted on a Linux machine when considering the choice of OS. Still, the software yielded output with zero values, given an example by the developers that should give non-zero values. We encourage the developers of this software to fix the issue and readers to look into the software as it could be a valuable tool for energy measurement. A thread about this issue has been written in the GitHub repository †.

Regarding FLOPs counting, the reliability is harder to assess. Tensor Flow does not have any verified method to count FLOPs and an issue in the Tensor Flow repository on GitHub addresses this issue [44]. In the thread, there is a method that gives FLOPs of a model. This means that the cost model input

* https://intel.ly/3NybYgA † https://bit.ly/3wJKptY

FLOPs needs to be derived with the same method as the test environment. The code that yields FLOPs can be found in Appendix A and a description on setup and usage can be found in the documentation found in Appendix B.

3.6 Planned Data Analysis

We will first process the data and subtract the idle time energy consumption, resulting in the energy usage from the deployed model. This data will be the subject of a regression analysis where the correlation coefficient will be derived to assess the correlation of FLOPs and energy.

3.6.1 Data Analysis Technique

When we have accumulated the processed energy data for each model, we will map them to the FLOPs. We then apply a linear correlation between the two sets of data, work, and energy, to calculate the Pearson correlation coefficient. The coefficient is derived with Equation 3.1.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.1}$$

The coefficient r_{xy} is expected to be in the interval $[0 - 1]$, which would indicate that the work and energy grow larger together. A high correlation is given if r_{xy} is a value in the interval $[0.5 - 1]$. If we get a negative r_{xy}, we would know that the work done is *not* correlated to energy, and thus FLOPs is *not* a good metric to estimate energy costs.

3.6.2 Software Tools

We will read files generated with Intel® Power Gadget and the local database where iStat Menues stores sensor data. The data will be read with python and analyzed with the help of the python pandas package [45]. We will use the python package Matplotlib to generate graphs to show the linear correlation graph [46]. The source code for this analysis code is shown in Listing C.1.

3.7 Evaluation Framework

The evaluation framework will answer whether our metrics chosen to construct the cost model are suitable ones. Depending on the linear correlation analysis,

we will assess the correlation coefficient to derive conclusions about why we correlated or not. If not, we will need to look at what other metrics could have given a better correlation.

As energy measurements are considered valid and reliable, the evaluation will consider the metrics for the work done by the model. The questions that need to be answered are the following: (i) Is FLOPs a good work metric? (ii) Why is it, or not, a good metric? (iii), if it is not a good metric, which metrics give a better understanding of the work done by the chosen NLP model?

When we have answered these questions, we are going to develop the cost model. As there are various metrics in addition to energy and work, we will have to design the cost model so that others can use it in an intuitive way.

When we have answered the above questions, we will also need to validate the cost model output against other cost calculators, such as the ones Google has developed [40, 41].

3.8 System Documentation

Appendix A presents the script used to make request to the server presented in section 4.4. In appendix B we show which python packages were used in the test environment, and Appendix C presents the code used to analyse the measured data points and create the graphs presented in Chapter 5.

The code used to make all measurements and tests are found on my GitHub*.

* https://github.com/wolf019/NLP-Inference-Cost-Model

Chapter 4

Measurements and Data Processing

The initial test environment did not work as expected. Using the TensorFlow PluggableDevice Metal package for macOS [39] proved to be problematic and some adjustments needed to be made: when fine-tuning the model, utilising the discrete GPU worked as expected. However, when the server was tested, the time for making inference was $4\times$ as much with the discrete GPU compared to CPU. Therefore, we make measurements on CPU with Intel® Power Gadget when measuring inference, and together with Istat Menues to also measure the GPU when measuring fine-tuning.

When processing the data we will have to consider the two different subsystems separately. The fine-tuning is done with the main processing unit being the GPU, while inference is made with the CPU. The difficulty was that while Tensorflow-Metal supports AMD GPUs, as previous paragraph explains, it was problematic to use. This is *not* something that affects out results. The FLOPs can be executed on any hardware we can measure with known efficiency. It will just take more time to execute the tests.

When removing the hardware dependency from our energy cost equations, we will calculate the efficiency for each type of processor type. In the following sections we will show the collected data, how we process it, and calculate the linear correlation coefficient.

Chapter 7 show how to avoid this problematic in future work and which test environment that is needed to yield the best results.

We made measurements of three aspects to capture the deployment process:

1. before we can deploy a model we need to *fine-tune* it. Due to lack

of powerful hardware and measurement issues described above, we measure the steps in each epoch and approximate the full training process;

2. when a model has been *fine-tuned* and set up in a web framework the server has to be started. We therefore measure the *cold-start* energy consumption for each model; and

3. when the model has been deployed in the web framework the *inference* starts. We run each of the scenarios and measure the energy consumed.

4.1 Fine-Tuning Results

Fine-tuning in the test environment had some natural limitations. Due to time limitations, training time on the machine's GPU needed to be approximated to avoid the need to run for several days per model. The approximation was made through training for x number of steps in an epoch. Various number of steps were tested in the range $50-550$ for five tests. For each of these five test, Power Gadget measured the energy of the test environment except the discrete GPU that was measured with iStat Menues.

In Equation (4.1) we sum up the joules data from Power Gadget and subtract the idle time energy. The idle time energy j_{idle} is calculated as the average idle time watts for CPU and DRAM, and multiplied by the training time.

$$j_{tot} = \sum_{i=start}^{stop} (j_{[i]CPU} + j_{[i]DRAM_{CPU}}) - j_{idle} \tag{4.1}$$

As the data points from iStat Menues were gathered every 5 seconds, we approximate the integral in Equation (4.2) using the Trapezoidal Rule. When the integral has been approximated, we subtract the measured idle energy consumption of the GPU.

$$j_{gpu} = \int_{i=training\_start}^{stop} (W_{[i]GPU}) - W_{GPUidle} * (stop_s - training\_start_s) \tag{4.2}$$

The results are presented in Table 4.1. As we will see in Section 4.3, the *Fast* tokenizer was more energy efficient and Table 4.1 shows that training with *Fast* tokenizer as it was more energy efficient.

Table 4.1: Table of Models [10, 23, 24, 25], Work, fine-tuning consumption per step and standard deviation, Total training consumption, which of GPU used in percentage

Model	GFLOPs	Step Joules	kWh	GPU usage
DistilBERT	48.526	141.463±1.117	1.592	91.220%
$ALBERT_{BASE}$	97.178	266.527±0.293	2.999	91.834%
RoBERTa	97.042	269.748±3.312	3.035	91.231%
$BERT_{BASE}$	97.119	307.120±1.971	3.455	91.597%

From the table we can see that FLOPs is correlated with energy consumption. We can also see that the energy usage of the GPU comprises the majority of the total energy consumption. What is interesting is that the GPU energy usage seems to be somewhat consistent over the models. Studying this further would be interesting to see if there exists a percentage that could be used to predict energy consumption by measuring the GPU alone.

We are not interested in accuracy as the PLMs promise SOTA NLP results. But for the curious, a closer evaluation reveals that the accuracy for the selected models is 93.0 ± 0.4%.

4.2 Cold-Start Results

The cold start was measured with the web framework flask [47]. Before each cold start we made sure the memory was freed. We made three cold starts and the measurements in Table 4.2 shows the mean value of the measurements to give an average cold-start consumption.

Table 4.2: Table of Models and Parameters[10, 23, 24, 25], Work, and cold-start consumption

Model	Parameters (millions)	GFLOPs	joules
$ALBERT_{BASE}$	11	97.2	160.2
DistilBERT	66	48.5	188.8
$BERT_{BASE}$	110	97.1	237.9
RoBERTa	130	97.0	258.8

We can see that there is a correlation between the number of parameters and energy consumed in joules. This make sense as more parameters have to be loaded upon deployment.

4.3 Inference Results

Table 4.3 shows the inference results of the 1100 text classifications. The table show two important values for each model. Joules describes the energy needed to run the scenarios and FLOPs is the work needed to evaluate one request with the selected model.

Table 4.3: Table of Models and Parameters[10, 23, 24, 25], and Work made by one inference

Model	**Tokenizer**	**Parameters (millions)**	**GFLOPs**	**Joules**
DistilBERT	Fast	66	48.5	8774.91
DistilBERT	Base	66	48.5	9260.82
RoBERTa	Fast	130	97.0	17 262.12
RoBERTa	Base	130	97.0	18 156.30
$BERT_{BASE}$	Fast	110	97.1	17 649.13
$BERT_{BASE}$	Base	110	97.1	17 724.58
$ALBERT_{BASE}$	Fast	11	97.2	19 952.61
$ALBERT_{BASE}$	Base	11	97.2	20 143.18

As we do not consider server idle time in the energy consumption, we derive Joules by summing up the energy consumed by CPU and DRAM_{cpu} during the data collection stage and subtract the average energy consumption of the system when no inferences are evaluated. The calculation is made with Equation 4.3 where j_x is energy.

$$j_{tot} = \sum_{i=start}^{stop} (j_{[i]CPU} + j_{[i]DRAM_{CPU}}) - j_{idle} \tag{4.3}$$

We can see that the parameters do not play a role in estimating inference energy cost. This was expected. What was also expected is that FLOPs is a good metric to describe the model's architecture, while Parameters is not. Remember that the models selected are all transformer models and share architectural characteristics. If we were to include other kinds of models or more advanced transformer models we can not be sure that FLOPs is a good metric. This is further discussed in Chapter 6.

Table 4.3 shows that the choice of tokenizer also plays a role in energy consumption. This was also expected as the *Fast* tokenizer option for each model is implemented with the functional programming language Rust [48]. If the tokenizer processes the words more efficiently, which a Rust tokenizer implementation should do, the energy consumption should be lower, and our measurements shows that.

4.4 Test Environment Code

The tests were all made with the code presented. Due to time limitations we could not test more web frameworks than flask [47]. All tests were conducted on the specified test environment machine to process the requests.

Listing 4.1 shows an example of the fine-tuning written in python. In contrast, Listing 4.2 shows an example of the inference server code also written in python.

Listing 4.1: Fine-Tuning DistilBert

```
dataset_name = 'ag_news'
num_targets = 4
model_name = "distilbert-base-uncased"
max_length = 512

# Manually specify the number of unique targets
train_dataset = load_dataset(dataset_name,
    ↪ split="train[10%:]")
val_dataset = load_dataset(dataset_name,
    ↪ split="train[:10%]")
test_dataset = load_dataset(dataset_name,
    ↪ split="test")

# load the tokenizer (convert our text to sequence
    ↪ of tokens)
tokenizer =
    ↪ DistilBertTokenizerFast.from_pretrained(model_name,
    ↪ do_lower_case=True)

# tokenize the dataset, truncate when passed
    ↪ 'max_length' and pad with 0's when less than
```

```
    ↪ 'max_length'
train_tokenized = train_dataset.map(lambda x:
    ↪ tokenizer(x['text'], truncation=True,
    ↪ padding='max_length'), batched=True)
val_tokenized = val_dataset.map(lambda x:
    ↪ tokenizer(x['text'], truncation=True,
    ↪ padding='max_length'), batched=True)
test_tokenized = test_dataset.map(lambda x:
    ↪ tokenizer(x['text'], truncation=True,
    ↪ padding='max_length'), batched=True)

# get data in standard tf.data.Dataset and remove
    ↪ 'text' label as it is not longer needed
tf_train_dataset =
    ↪ train_tokenized.remove_columns(['text']).
      with_format('tensorflow')
tf_val_dataset =
    ↪ val_tokenized.remove_columns(['text']).
      with_format('tensorflow')
tf_test_dataset =
    ↪ test_tokenized.remove_columns(['text']).
      with_format('tensorflow')

# convert to tensors
train_features = {x: tf_train_dataset[x] for x in
    ↪ tokenizer.model_input_names}
train_tf_dataset =
    ↪ tf.data.Dataset.from_tensor_slices((train_features,
    ↪ tf_train_dataset["label"]))
train_tf_dataset =
    ↪ train_tf_dataset.shuffle(len(tf_train_dataset)).batch(8)

val_features = {x: tf_val_dataset[x] for x in
    ↪ tokenizer.model_input_names}
val_tf_dataset =
    ↪ tf.data.Dataset.from_tensor_slices((val_features,
    ↪ tf_val_dataset["label"]))
val_tf_dataset =
    ↪ val_tf_dataset.shuffle(len(val_tf_dataset)).batch(8)
```

```
test_features = {x: tf_test_dataset[x] for x in
    ↪ tokenizer.model_input_names}
test_tf_dataset =
    ↪ tf.data.Dataset.from_tensor_slices((test_features,
    ↪ tf_test_dataset["label"]))
test_tf_dataset =
    ↪ test_tf_dataset.shuffle(len(test_tf_dataset)).batch(8)

# train keras model
encoder = TFAutoModelForSequenceClassification
.from_pretrained(model_name, num_labels=num_targets)

input_ids = tf.keras.Input(shape=(max_length,),
    ↪ dtype=tf.int32, name='input_ids')
token_type_ids =
    ↪ tf.keras.Input(shape=(max_length,),
    ↪ dtype=tf.int32, name='token_type_ids')
attention_mask =
    ↪ tf.keras.Input(shape=(max_length,),
    ↪ dtype=tf.int32, name='attention_mask')

embedding = encoder([input_ids, attention_mask])
logits = embedding[0]

model = tf.keras.models.Model(
    inputs = [input_ids, attention_mask],
    outputs = logits,
    name='tf_distilbert_classification'
    )

model._name = 'tf_distilbert_classification'

model.compile(
    optimizer=tf.keras.optimizers.SGD(learning_rate=5e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True),
    metrics=tf.metrics.SparseCategoricalAccuracy()
)
```

```
model.fit(train_tf_dataset,
    ↪ validation_data=val_tf_dataset, epochs=3)

model.save_pretrained(f"../models/{model_name}-trained")
```

Listing 4.2: Inference Server

```
app = Flask(__name__)

# Parameters
dataset_name = 'ag_news'
num_targets = 4
model_name = "distilbert"
max_length = 512

# Class names
class_names = {
    0: "World",
    1: "Sports",
    2: "Business",
    3: "Sci/Tech"
}

# Load tokenizer
tokenizer = DistilBertTokenizer.from_pretrained(
    model_name, do_lower_case=True)

# Load model.
model =
    ↪ TFAutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=num_targets)
model.load_weights(f"../distilbert/tf_model.h5")


@app.before_request
def logging_before():
    # Store the start time for the request
    app_ctx.start_time = time.perf_counter()


@app.route("/isalive")
def isalive():
    return Response(status=200)
```

```
# Serve the model


@app.route('/predict', methods=['GET', 'POST'])
def prediction():
    # Receive request
    req = request.get_json(silent=True, force=True)

    if text is None:
        # No text to classify
        return jsonify(code=403, message="bad␣
            ↪ request")
    else:
        result = []
        # Classify all texts in request
        for i in range(len(req['instances'])):
            tokenized_text = tokenizer.encode(
            req['instances'][i]['text'],
                ↪ truncation=True, padding=True,
                ↪ return_tensors="tf")

            class_names[np.argmax(tf.nn.softmax(
            model(tokenized_text)[0],
                ↪ axis=1).numpy())]

            result.append(class_names[np.argmax(
            tf.nn.softmax(model(tokenized_text)[0],
                ↪ axis=1).numpy())])

        return jsonify(predictions=result)


@app.after_request
def logging_after(response):
    # Get terminal print out of total time in
        ↪ milliseconds
    total_time = time.perf_counter() -
        ↪ app_ctx.start_time
    time_in_ms = int(total_time * 1000)
```

```
    # Log the time taken for the endpoint
    current_app.logger.info('%s ms %s %s %s',
        ↪ time_in_ms,
          request.method, request.path,
            ↪ dict(request.args))
    return response


if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0", port=8080)
```

Chapter 5

Cost Model Construction

In this chapter, we present the results, discuss them, and construct the cost model that meets the first Subgoal 1. Subgoal 1 has been the main focus of this thesis as the complexity of measuring energy was greater than expected. As a result, no other subgoals have been met and will instead be brought up in Chapter 6.

The energy results had three natural parts, as discussed in Chapter 4. To construct the cost model, we will perform a regression analysis on each part with the data from Chapter 4 to then construct an energy cost predictor equation and validate it with data from a model outside the data collection phase.

5.1 Correlation

We analyze (i) Fine-tuning, (ii) Cold-start, and (iii) Inference one by one in the coming subsections. We analyze the correlation coefficient squared, R^2, to analyze the correlation between FLOPs and Joules. We will find a promising correlation, which gives confidence in constructing the cost model in Section 5.3

5.1.1 Fine-Tuning Analysis

We start with the energy measurements for the Fine-tuning step when deploying PLMs. We need to separate the measurements of the CPU and GPU to be able to remove the hardware dependency, which we will do in Section 5.3.

We first look at the correlation of the CPU. As a similar analysis can not be done on the DRAM_{cpu} we analyze its energy consumption together with the CPU to avoid making the cost model too technical.

In Figure 5.1 we see each model together with a null value for no work done. The fitted line reveals a strong correlation with $R^2 = 0.952$, which indicates that FLOPs is a good predictor for energy consumption regarding the CPU and DRAM_{cpu}.

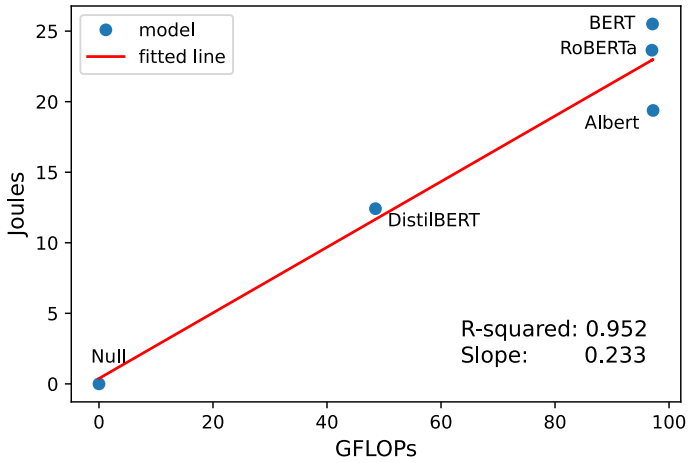

Figure 5.1: Regression line on CPU and DRAM energy consumption per step

Figure 5.2 shows an even stronger correlation with $R^2 = 0.990$. Again showing that energy can be predicted with the help of FLOPs regarding the energy consumption of the GPU. Both regression analysis above has a p-value smaller than 0.01, which makes us reject the null hypothesis. However, with just four data points, we want to validate the analysis with new data points. The validation will be made in Section 5.4.

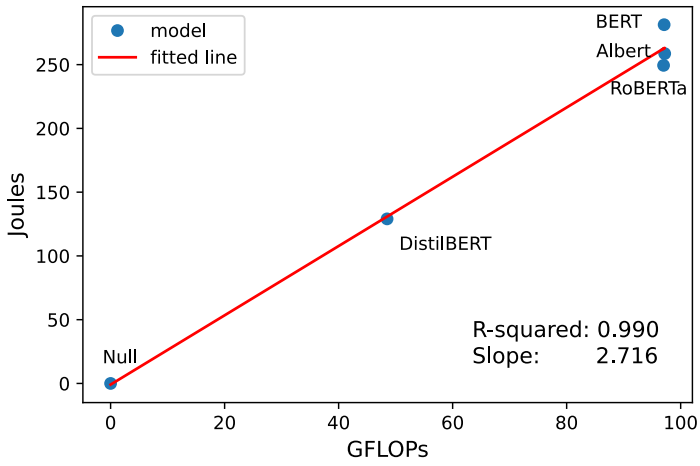

Figure 5.2: Regression line on GPU energy consumption per step

5.1.2 Cold-Start Analysis

Something expected regarding cold-start was that the correlation was weak. Starting up the inference server has little to do with the work done by the model - as model is not operating yet. However, we expected the models to have similar cold start energy consumption; however, there is a 40% difference between the lowest and highest values seen in Figure 5.3.

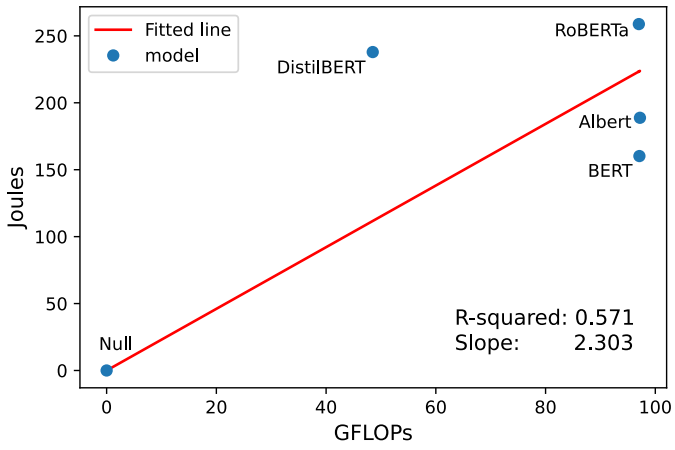

Figure 5.3: Regression line on energy needed to start up the inference server

Comparing the magnitude of the energy needed to handle the scenario-based tests with the cold-start energy consumption shows that the cold-start energy consumption is almost negligible. Still, we will add 250 joules to the predicted energy consumption to account for cold-start.

5.1.3 Inference Analysis

The energy values j_{request} shown in Figure 5.4 are derived from kWh values in Table 4.1 converted to joules j_{tot}. Together with Equation 5.1 we calculate the energy consumption per request as:

$$j_{\text{request}} = \frac{j_{\text{tot}}}{\text{requests}} \tag{5.1}$$

Figure 5.4 also shows that FLOPs are a good predictor regarding inference energy consumption. The strong correlation gives us confidence to proceed with the construction of the cost model.

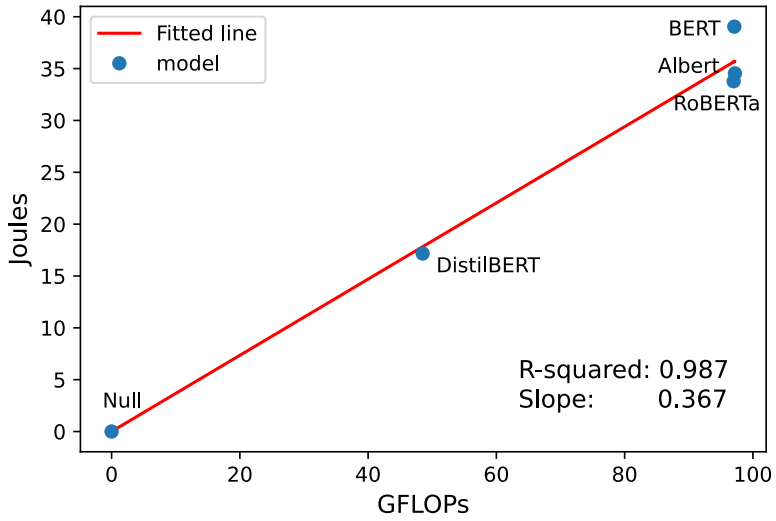

Figure 5.4: Regression line on inference energy consumption per request

5.2 Reliability Analysis

The reliability is understood by looking at the correlation coefficient squared. The strength of the relationship between FLOPs and Energy is considered

reliable for all energy accounting events except the cold-start. We have gathered the R-squared values in Table 5.1.

Table 5.1: Table of R-squared from Figures 5.1, 5.2, 5.3 and 5.4

Event	R-Squared
Fine-Tuning CPU	0.952
Fine-Tuning GPU	0.990
Cold-Start	0.542
Inference	0.984

Regarding cold-start, we have understood that the energy consumption has little to do with the FLOPs for each model as the model is not operating in this stage. As described in Section 5.1.2 we still account for cold-start by adding a fixed value that does not underestimate the consumption of energy.

At this stage, we have the reliability needed to proceed with the cost model construction, and we will do that in Section 5.3.

5.3 Cost Model Calculation

In Section 2.4.1 we presented efficiency, a commonly used theoretical value to express the hardware's ability to execute FLOPs per time unit. As the tests were performed on specific hardware, we now need to remove the hardware dependency from the results. This is because we want to use hardware efficiency as a metric to predict energy consumption. To remind us of the hardware efficiency variable we include the Equation 2.2 from Section 2.4.1 as Equation 5.2.

$$e = \text{Efficiency} = \frac{\text{FLOPS}}{\text{Watt}} = \frac{\text{FLOPs}}{\text{Joule}} \tag{5.2}$$

We have the correlation lines $joules = \alpha \cdot \text{FLOPS}$ for our events as intercept is anchored to 0. We now multiply the slope α with the efficiency e to get the coefficient β such that $joules = \frac{\beta \cdot \text{FLOPs}}{e}$. In this way we are able to input both the work needed to run the model and the efficiency for the chosen hardware. With efficiency $e_{\text{cpu}} = 8.82 \times 10^9$ we get $\beta_1 = \alpha_{\text{cpu}} \cdot 8.82 \times 10^9 = 2.05 \times 10^9$, and with efficiency $e_{\text{gpu}} = 24.77 \times 10^9$ we get $\beta_2 = \alpha_{\text{gpu}} \cdot 24.77 \times 10^9 = 67,29 \times 10^9$. These equations represents the Fine-tuning stage and constitutes the Equation 5.3.

$$\text{energy\_fine\_tune} = \text{epochs} \cdot \text{steps} \cdot \left(\frac{\beta_1 \cdot \text{FLOPs}}{e_{\text{cpu}}} + \frac{\beta_2 \cdot \text{FLOPs}}{e_{\text{gpu}}} \right) \tag{5.3}$$

With efficiency $e_{cpu} = 8.82 \times 10^9$ we get $\beta_3 = \alpha_{\text{inference}} \cdot 8.82 \times 10^9 = 26.25 \times 10^9$, which gives us Equation 5.4.

$$\text{energy\_inference} = \frac{\beta_3 \cdot \text{FLOPs} \cdot \text{requests}}{e_{\text{cpu}}} \tag{5.4}$$

To then construct the final cost model calculation needed to predict the inference energy, we need to know how many times the fine-tuning is going to be run on new data. We call this variable *maintenance*. Important to notice here is that the data set size needs to be consistent, and old data needs to be replaced with new data.

The final calculation to predict inference energy is shown in Equation 5.5.

$$\begin{aligned} \text{predicted\_energy} = &(\text{energy\_fine\_tune} + \text{cold\_start}) \cdot maintenance + \\ &\text{energy\_inference} \end{aligned} \tag{5.5}$$

5.4 Validity Analysis

To test the validity of the cost model, we will use a new larger model, specifically $BERT_{LARGE}$, to see how the predicted values from our cost model maps to the real values from the methods used for data collection. The results are shown in Table 5.2.

Table 5.2: Table of measured values of $BERT_{LARGE}$.

Model	**Parameters (millions)**	**GFLOPs**	**Fine-Tune (kWh)**	**Cold-Start (Joule)**	**Inference (Joule)**
$BERT_{LARGE}$	336	336.3	16.7	371.3	45886.2

Using the equations in Section 5.3 we get the predicted values shown in Table 5.3.

Table 5.3: Table of predicted energy consumption of $BERT_{LARGE}$

	Fine-Tune (kWh)	**Cold-Start (joule)**	**Inference (Joule)**	**Total (kWh)**
Real	16.7	371.3	45 886.2	16.7
Predicted	11.2	250.0	63 148.1	11.2

At this stage, we would like to state that the cost model predicts values with the equations stated Section 5.3. However, this validation of the model showed a fundamental flaw in our predictions. It simply does not predict the correct energy consumption. To understand why, we need to look closer at how the correlation line behaves with the new data of $BERT_{LARGE}$.

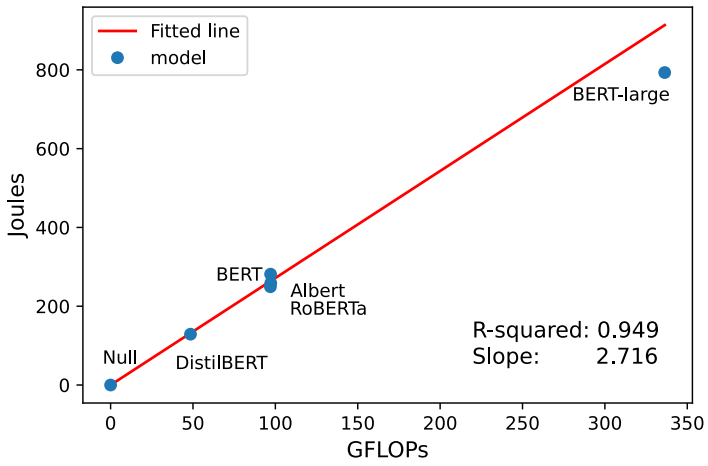

Figure 5.5: Regression line on GPU energy consumption per step

Figure 5.5 shows that FLOPs is still a pretty good predictor regarding the GPUs, so the problem is not there.

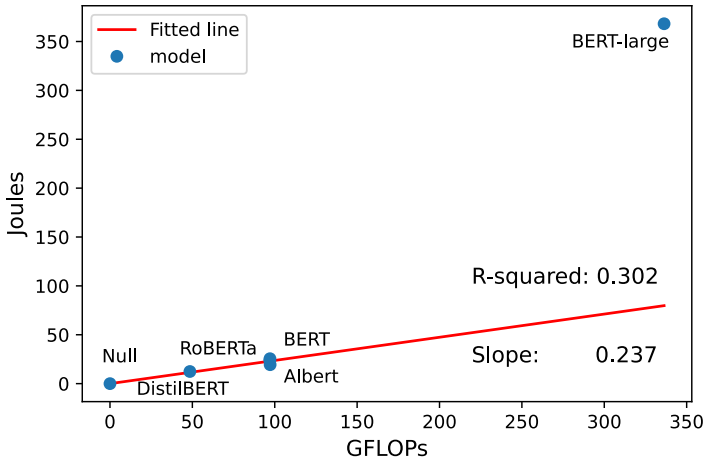

Figure 5.6: Regression line on CPU energy consumption per step

Looking at Figure 5.6, we see that the R^2 value has decreased substantially. This was unexpected. When we included the energy consumption for the DRAM_{CPU} into the CPU analysis, we did not understand how the work done by the memory changed with model size. The memory needs to work more than one might think when the model grows larger; however, this is not the case. If we compare how much energy the memory used for the total consumption of CPU and DRAM_{CPU} we find that memory used close to 22.0% of the energy when looking at the original data collected. For $BERT_{LARGE}$ that percentage is 15.8%. This means that the CPU is using *more* energy than expected for a larger model.

Further, the GPU usage had dropped to 75,9%. We discussed in Section 4.1, that the GPU usage percentage is interesting to examine more. Now it is clear that the question about CPU and GPU energy usage division is more complex that what our model considers.

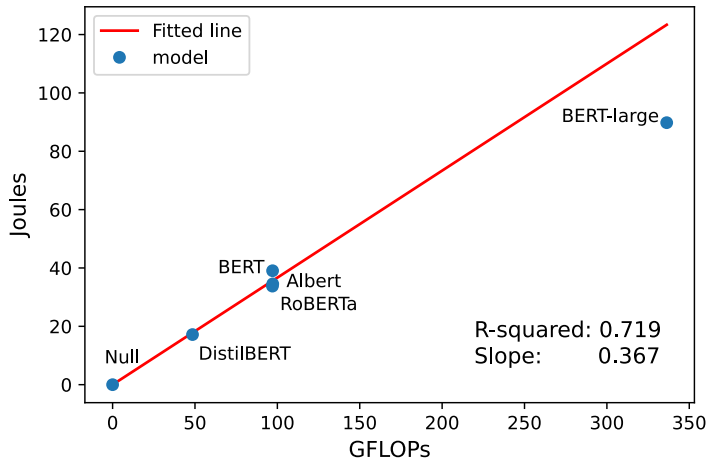

Figure 5.7: Regression line on CPU energy consumption per request

Last we look at the inference phase, and we see that the correlation has weakened with the new data point. R^2 value is somewhat large, but *not* as large as before (when excluding $BERT_{LARGE}$).

We must accept that the validity test has failed to give us the confidence that we have constructed a valid cost model. We will discuss this further in Chapter 7.

Chapter 6

Discussion

Deriving a deployment cost model proved to be a harder task than expected. We explored many sidetracks and rabbit holes to understand how to make a predictive cost analysis of an NLP solution.

6.1 Cost Center

One of the hardest problems was defining the cost center. Unfortunately, there is no clear answer, and what to consider to be a cost associated with NLP deployment could be another discussion. What we talk about here is the responsibility, *i.e.,* which part of a machine or Cloud Service Provider (CSP) is responsible for energy usage? Then the next question is, how can we make it more efficient? Who is accountable? To answer these questions is straightforward today: It is the developer. However, the developer can not solve all problems simultaneously, as this thesis made clear. An accounting model for who has the energy responsibility would solve this problem.

Exploring the area of energy predictions regarding NLP models gave rise to several questions that are essential to give a prediction on energy cost. When modeling the test environment, we need to decide which parts of the machine to measure. Where does one draw the line when accounting for machine learning costs? The inference is only interesting to analyze in an application. The application deployed uses several hardware components to handle requests, but there is a line to be drawn as to where the cost for the server is separate from the cost for the application. Applications primarily use processing units and memory, and these are the components we have handled in our model. One could also account for the Network Interface Controller (NIC) costs along with the idle time for the server, but when talking

about a cost model for NLP inference applications, the line needs to be drawn somewhere. A literature search for where this line should be drawn has not yielded any concrete findings. The debate about where the energy cost accounting limits should be drawn is an interesting and necessary debate in order to make energy accounting more concrete. For now, we have "Green-IT" and "Green-AI" to consider when developing, but these terms do not say anything concrete regarding energy cost accounting.

6.2 Predictor Metrics

As we have seen in this thesis, FLOPs is a good predictor of energy consumption. However, the feasibility of counting floating point operations is somewhat cumbersome. We need to train a specific model to use the method found by the AI community to count the FLOPs. We suggest that there is a need for an implementation of a FLOPs counting method into the various libraries that are used by the AI community to make analysis, such as the one in this thesis, more feasible to conduct.

We hope that this thesis will give incentives to others to explore the area of energy accounting as well as suggesting there is a need for support to collect important metrics relevant to an PLM model. Standardized metrics are of huge importance to be able to focus on the problem at hand. Using a cost model such as the one proposed in this thesis requires easy access to information about the inference-ready model.

Metrics that are a good predictor of energy consumption have been important in this thesis. We need to remember that the observed correlation of FLOPSs and energy only considers the Transformer model architecture. Introducing other architectures when examining this correlation could weaken the correlation; thus, we may need to consider other metrics. If this degree project had more resources, we would explore the various inputs like dimensions and layer types to see if we could apply a more general cost accounting. Essentially, we want to have metrics that give a good understanding of the calculation to be performed by the model. We suggest that there is still a need for a study which metrics could & should be used.

6.3 GPU Usage Observation

Working in a cloud environment has its limitations. Even if the cloud Virtual Machine (VM) has a CPU architecture that supports Running Average Power Limiting (RAPL), this does not mean that RAPL is accessible. Multiple attempts were made on GCP with no success in accessing the RAPL interface.

In Chapter 5 we saw that the percentage of CPU and DRAM_{cpu} were almost consistent over the tests when measuring the total energy consumption. A more extensive study would be interesting to conduct to examine if this percentage has a narrow range over various inference machines.

Regarding cloud computing and the Nvidia GPUs, there is usually a terminal-based interaction available. Using the command "nvidia-smi" we can learn the wattage that the GPU is using in real-time, and using a theoretical percentage value for predicting CPU and DRAM_{cpu} energy consumption would make it possible to easily measure the energy consumption of any machine learning inference solution. This would make it possible to construct a terminal-based program, which would be able to predict the cost in a cloud environment. However, our validation of the cost model calculation showed that the percentage varies a lot with model size. This area could still be interesting if one could predict the percentage of GPU usage from the AI model's metrics.

Chapter 7

Conclusions and Future work

With the limitations in time and resources, we could not conduct all of the tests needed to understand the inner workings of NLP inference. However, some primary outcomes suggest that there is value in the future to develop an accurate inference cost model. This chapter presents the conclusions of this thesis as well as suggests future work needed to meet the sub goals presented in Section 1.4.

7.1 Conclusions

Looking back at Chapter 1 we had high expectations for the outcome of this thesis. While none of the goals in 1.4 were meet; however, a lot of work was done toward reaching them. We found tools that measure energy on most systems from the extensive literature study. For RAPL supported Intel©processors, there is a straightforward measuring tool that is proven to give accurate results [37]. If I were to redo this thesis, I would have just focused on energy measurements and not a cost model. Actually constructing a complete cost model for AI inference is complex.

Using FLOPs as a metric mapped to energy is *not* the best approach. FLOPs are executed differently on different hardware, and some instructions are more efficient, such as Single Instruction/Multiple Data (SIMD), which makes FLOPs somewhat arbitrary. Other metrics that better describe the model's architecture will probably better map to energy. Unfortunately, we cannot explain the model's architecture in a single value as the value really has multiple dimensions, *i.e.,* akin to width, height, units, *etc*. A study of how to express each model's architectural characteristics would be needed to identify better metrics; however, this study has not found any metrics. We call

on the community to conduct more research in this area to map out common model architecture characteristics.

Recent papers presenting large transformer language models have shown promising results with high accuracy [49, 50]. These models have many parameters, and one might think that the number of parameters is correlated with high energy costs. However, even though these models have very many parameters, we have seen that the number of parameters alone are **not** a great predictor of energy consumption.

Knowledge distillation, used to develop DistilBERT, was shown to be a substantially efficient method to lower energy consumption. Our results in Chapter 5 show that DistilBERT is around 50% more efficient than the other non distilled models of similar size. We should remember that DistilBERT has 44 million parameters while Albert had 12 million. However, Albert reuses its parameters, and the number of individual parameter calculations are more than 12 million for Albert. We can conclude that the number of parameters does not say too much about energy consumption. Again, better model metrics that describe the actual work a model performs would help uncover more profound knowledge about energy consumption.

When conducting the data collection, we became familiar with the Tensorflow PluggableDevice package Metal for macOS [39]. We successfully installed the package and used the discrete AMD©GPUs to fine-tune the models. The relevant packages are described in Appendix B.

7.2 Limitations

Three main constraints limited the resulting cost model. The metric choice (*i.e.,* FLOPS), duration of the project, and the test environment.

As discussed in this thesis, we measured the work done in FLOPs, which is an arbitrary unit to use – as other factors also matter when executing FLOPs. We saw a good correlation when analyzing the results, but this was mainly due to the chosen models' constraints on the transformer. When validating our cost model, we saw that FLOPs was **not** a good predictor; therefore, we need other metrics to measure the work done to make a cost model applicable to models with different architectures. However, due to the limited duration of this project, we had only time to explore FLOPs as a metric.

Using FLOPs was also problematic as we could not calculate FLOPs of the whole inference program. This was partly due to the usage of macOS that did not support the python packages that do exist, and time constraints that limited the time that could be spent exploring other ways of counting FLOPs

(such as using Performance Monitoring Unit (PMU) events/counters or the Intel® Software Development Emulator (Intel® SDE)) [51].

The third constraint was the test environment. It became clear that using a laptop computer running macOS as a test environment limited which hardware components could be used and measured. Nvidia cards are preferable when measuring energy as we can easily measure them with the command line tool nvidia-smi. However, Nvidia drivers are not developed for macOS, and there is no official way to make Nvidia cards compatible with macOS unless the macOS High Sierra version from 2017 is installed. As risks of downgrading the test environment existed and backup disks were unavailable, we did use the Nvidia GPUs.

7.3 Future Work

Due to the complexity of the problem, only some parts of Subgoal 1 have been met. In this section, we will focus on the remaining issues that should be addressed in future work.

7.3.1 What has been left undone?

Creating a cost model is dependent on finding reliable metrics that predict energy consumption. We have seen in this thesis that FLOPs on its own is insufficient. Examining more complex metrics would be the next natural step.

Given the work presented in this thesis, the next step would also include more models in order to get more data points. All of the source code used in this project has been uploaded to GitHub to facilitate someone else continuing from where this thesis ended.

When suitable metrics have been found that pass the validation, then we can consider Subgoal 1 to be completed and proceed to Subgoal 2 and Subgoal 3.

7.3.2 Cost Analysis

A complete cost analysis has been left undone, as we do not have a reliable energy cost predictor. As soon as energy cost would be somewhat predictable, then the cost analysis could be finalized through data collection of monetary prices regarding on-premises servers or cloud services. When the price of running hardware has been predicted, then the energy cost is easily computed by multiplying the energy consumed by the energy price of the server's supply

of power. Additionally, the energy costs must be further multiplied by the server's PUE to give a realistically predicted GHG emissions for the inference period.

7.3.3 Next obvious things to be done

In particular, the author of this thesis wishes to point out that achieving Subgoal 1 requires finding valid metrics to predict energy consumption before it can be considered fully solved. This means that solving finding these metrics is the next thing that should be done. Using model architecture-specific metrics such as layers, transformer units, and similar characteristics could be suitable factors to consider.

7.4 Reflections

One of the most important results that this thesis tried to achieve is the ability to reduce the amount of energy required to handle the inference period of an AI-System life cycle through smarter implementation choices. A cost model to predict inference costs would give economic and environmental benefits globally. I hope that the AI-community will use my work to derive a reliable cost model.

The thesis contributes to the UN SDGs numbers 9 and 13 by making new AI innovations adopt "Green-AI" thinking and thus lower the energy consumption of the AI community.

This thesis has been a complex matter to handle for one Bachelor's student. Although I have found the subject very interesting as it deals with something that will benefit the global environment, I am glad to have falling into many traps by thinking that the work would be straightforward. A few parts are now clear that I wished I had known before I started: (i) Each model is unique in its architecture and will therefore give rise to a specific amount of work - as estimating this work is the key, spending more time on understanding a model's architecture characteristics would give a better understanding of choosing suitable metrics to estimate the work that must be done by the model; (ii) the test environment needs to be modern and effective, for example an Nvidia GPU with an Intel CPU with RAPL support on a Linux machine would have been a better test setup; and (iii) the experiment-impact-tracker would be an excellent tool when making measurements [27]. While the code for this tool works and gives no errors, it does not give any energy information that is greater than zero.

Before anyone continues my work, I strongly recommend carefully reading Chapters 3, 4, and 5; set up the test environment I recommended above; and try to make the experiment-impact-tracker work. The code on GitHub * makes energy measurements but is somewhat cumbersome to use, a working experiment-impact-tracker is recommended.

I want to end this thesis by encouraging interested readers to keep looking in the GitHub repository, where I will post updates regarding future work on a reliable NLP inference cost model.

---

* https://github.com/wolf019/NLP-Inference-Cost-Model

References

[1] United Nations Framework Convention on Climate Change UNFCCC), "ICT Sector Helping to Tackle Climate Change," Aug. 2016. [Online]. Available: https://unfccc.int/news/ict-sector-helping-to-tackle-climate-change [Page 1.]

[2] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the Carbon Emissions of Machine Learning," *arXiv:1910.09700 [cs]*, Nov. 2019, arXiv: 1910.09700. [Online]. Available: http://arxiv.org/abs/1910.09700 [Pages 2 and 20.]

[3] E. Strubell, A. Ganesh, and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP," *arXiv:1906.02243 [cs]*, Jun. 2019, arXiv: 1906.02243. [Online]. Available: http://arxiv.org/abs/1906.02243 [Pages 2, 6, 11, and 16.]

[4] O. Sharir, B. Peleg, and Y. Shoham, "The Cost of Training NLP Models: A Concise Overview," *arXiv:2004.08900 [cs]*, Apr. 2020, arXiv: 2004.08900. [Online]. Available: http://arxiv.org/abs/2004.08900 [Pages 2, 6, 11, and 12.]

[5] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, Dec. 2019. doi: 10.1016/j.jpdc.2019.07.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518308773 [Page 2.]

[6] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon Emissions and Large Neural Network Training," *arXiv:2104.10350 [cs]*, Apr. 2021, arXiv: 2104.10350. [Online]. Available: http://arxiv.org/abs/2104.10350 [Page 2.]

[7] E. Kern, L. M. Hilty, A. Guldner, Y. V. Maksimov, A. Filler, J. Gröger, and S. Naumann, "Sustainable software products—Towards assessment criteria for resource and energy efficiency," *Future Generation Computer Systems*, vol. 86, pp. 199–210, Sep. 2018. doi: 10.1016/j.future.2018.02.044. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17314188 [Pages 2 and 5.]

[8] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," *arXiv:1907.10597 [cs, stat]*, vol. 63, no. December 2020, pp. 54–63, Aug. 2019. doi: 10.1145/3381831 ArXiv: 1907.10597. [Online]. Available: https://dl.acm.org/doi/10.1145/3381831 [Pages 2, 5, and 16.]

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017, arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762 [Pages 2, 11, and 13.]

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv:1810.04805 [cs]*, May 2019, arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 [Pages 2, 13, 24, 35, and 36.]

[11] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, and J. Zhu, "Pre-Trained Models: Past, Present and Future," *arXiv:2106.07139 [cs]*, Aug. 2021, arXiv: 2106.07139. [Online]. Available: http://arxiv.org/abs/2106.07139 [Pages 2 and 6.]

[12] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, Oct. 2020. doi: 10.1007/s11431-020-1647-3. [Online]. Available: https://doi.org/10.1007/s11431-020-1647-3 [Pages 2 and 12.]

[13] A. Alvi and P. Kharya, "Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, the World's Largest and Most Powerful Generative Language Model," Oct. 2021. [Online]. Available: bit.ly/3HEb7IG [Page 2.]

[14] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Compute and Energy Consumption Trends in Deep Learning Inference," *arXiv:2109.05472 [cs]*, Sep. 2021, arXiv: 2109.05472. [Online]. Available: http://arxiv.org/abs/2109.05472 [Pages 3, 12, 15, and 16.]

[15] T. Axberg and A. Nieto, "AI-system validation," Feb. 2022, "personal communication". [Pages xi, 3, 12, 13, 14, 15, and 23.]

[16] K. Freund, "Google Cloud Doubles Down On NVIDIA GPUs For Inference," May 2019, section: Enterprise & Cloud. [Online]. Available: https://www.forbes.com/sites/moorinsights/2019/05/09/google-cloud-doubles-down-on-nvidia-gpus-for-inference/?sh=a32716067926 [Page 6.]

[17] S. I. Park, S. P. Ponce, J. Huang, Y. Cao, and F. Quek, "Low-cost, high-speed computer vision using NVIDIA's CUDA architecture," in *2008 37th IEEE Applied Imagery Pattern Recognition Workshop*, Oct. 2008. doi: 10.1109/AIPR.2008.4906458 pp. 1–7, iSSN: 2332-5615. [Page 11.]

[18] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct. 2016. doi: 10.1109/BDCloud-SocialCom-SustainCom.2016.76 pp. 477–484. [Page 11.]

[19] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai, and T. Chen, "Recent Advances in Convolutional Neural Networks," *arXiv:1512.07108 [cs]*, Oct. 2017, arXiv: 1512.07108. [Online]. Available: http://arxiv.org/abs/1512.07108 [Page 11.]

[20] D. Thomas, "Reducing Machine Learning Inference Cost for PyTorch Models - AWS Online Tech Talks," Apr. 2020. [Online]. Available: https://www.youtube.com/watch?v=ET2KVe2du3Y [Pages 12 and 15.]

[21] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. doi: 10.1126/science.1127647 Publisher: American Association for the Advancement of Science. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647 [Page 12.]

[22] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "HuggingFace's Transformers: State-of-the-art Natural Language Processing," *arXiv:1910.03771 [cs]*, Jul. 2020, arXiv: 1910.03771. [Online]. Available: http://arxiv.org/abs/1910.03771 [Pages 13 and 26.]

[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv:1907.11692 [cs]*, Jul. 2019, arXiv: 1907.11692. [Online]. Available: http://arxiv.org/abs/1907.11692 [Pages 13, 24, 35, and 36.]

[24] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," *arXiv:1909.11942 [cs]*, Feb. 2020, arXiv: 1909.11942. [Online]. Available: http://arxiv.org/abs/1909.11942 [Pages 13, 24, 35, and 36.]

[25] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv:1910.01108 [cs]*, Feb. 2020, arXiv: 1910.01108. [Online]. Available: http://arxiv.org/abs/1910.01108 [Pages 13, 24, 35, and 36.]

[26] C. Belady, S. Berard, M. Bramfitt, T. Cader, H. Coles, J. Cooley, L. Coors, T. Darby, J. Froedge, N. Gruendler, J. Haas, E. Jewitt, C. Long, B. MacArthur, P. Morris, S. Microsystems, Z. Ortiz, J. Pflueger, A. Rawson, J. Simonelli, H. Singh, R. Tipley, R. Tozer, G. Verdun, J. Wallerich, and R. Wofford, "PUE™: A COMPREHENSIVE EXAMINATION OF THE METRIC," *The Green Grid.*, p. 83, 2012. [Online]. Available: https://datacenters.lbl.gov/sites/default/files/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf [Page 16.]

[27] 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," *arXiv:2002.05651 [cs]*, Jan. 2020, arXiv: 2002.05651. [Online]. Available: http://arxiv.org/abs/2002.05651 [Pages 16, 17, 20, 29, and 62.]

[28] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," *arXiv:1710.03740 [cs, stat]*, Feb. 2018, arXiv: 1710.03740. [Online]. Available: http://arxiv.org/abs/1710.03740 [Page 18.]

[29] "NVIDIA A100 GPUs Power the Modern Data Center," Apr. 2022. [Online]. Available: https://www.nvidia.com/en-us/data-center/a100/ [Pages 18 and 19.]

[30] M. Andersch, G. Palmer, R. Krashinsky, N. Stam, V. Mehta, G. Brito, and S. Ramaswamy, "NVIDIA Hopper Architecture In-Depth," Mar. 2022. [Online]. Available: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/ [Page 18.]

[31] V. Anand, "NVIDIA DGX A100 supercomputer is half the cost and size with double the performance! How does it do it?" *https://www.hardwarezone.com.sg/tech-news-nvidia-dgx-a100-supercomputer-super-performance-fight-covid-19*, May 2020. [Online]. Available: https://www.hardwarezone.com.sg/tech-news-nvidia-dgx-a100-supercomputer-super-performance-fight-covid-19 [Page 18.]

[32] "Pricing | Vertex AI | Google Cloud," Apr. 2022. [Online]. Available: https://cloud.google.com/vertex-ai/pricing [Page 18.]

[33] "TechPowerUp," Apr. 2022. [Online]. Available: https://www.techpowerup.com/gpu-specs/ [Page 18.]

[34] "Vertex AI," Apr. 2022. [Online]. Available: https://cloud.google.com/vertex-ai [Page 18.]

[35] "Intel® Power Gadget," Apr. 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html [Pages 20 and 26.]

[36] A. Nieto, "Scenario construction," Apr. 2022, "personal communication". [Page 24.]

[37] M. Giardino and B. Ferri, "Correlating Hardware Performance Events to CPU and DRAM Power Consumption," in *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug. 2016. doi: 10.1109/NAS.2016.7549395 pp. 1–2. [Pages 25, 29, and 59.]

[38] “iStat Menus,” Apr. 2022. [Online]. Available: https://bjango.com/mac/istatmenus/ [Page 26.]

[39] Apple Corporation, “Getting Started with tensorflow-metal PluggableDevice,” Apr. 2022. [Online]. Available: https://developer.apple.com/metal/tensorflow-plugin/ [Pages 26, 33, and 60.]

[40] Google, “Carbon Footprint,” Apr. 2022. [Online]. Available: https://cloud.google.com/carbon-footprint [Pages 28 and 31.]

[41] ——, “Pricing | Cloud Storage,” Apr. 2022. [Online]. Available: https://cloud.google.com/storage/pricing [Pages 28 and 31.]

[42] T. Axberg, “Twitter Conversation,” 2022. [Page 29.]

[43] P. Henderson, “experiment-impact-tracker,” Mar. 2022, original-date: 2019-11-06T05:44:17Z. [Online]. Available: https://github.com/Breakend/experiment-impact-tracker [Page 29.]

[44] “TF 2.0 Feature: Flops calculation · Issue #32809 · tensorflow/tensorflow,” Apr. 2022. [Online]. Available: https://github.com/tensorflow/tensorflow/issues/32809 [Page 29.]

[45] “pandas - Python Data Analysis Library,” Apr. 2022. [Online]. Available: https://pandas.pydata.org/ [Page 30.]

[46] “Matplotlib — Visualization with Python,” Apr. 2022. [Online]. Available: https://matplotlib.org/ [Page 30.]

[47] “Welcome to Flask — Flask Documentation (2.1.x),” Apr. 2022. [Online]. Available: https://flask.palletsprojects.com/en/2.1.x/ [Pages 35 and 37.]

[48] “Tokenizer,” Apr. 2022. [Online]. Available: https://huggingface.co/docs/transformers/main_classes/tokenizer [Page 37.]

[49] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan,

H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "PaLM: Scaling Language Modeling with Pathways," *arXiv:2204.02311 [cs]*, Apr. 2022, arXiv: 2204.02311. [Online]. Available: http://arxiv.org/abs/2204.02311 [Page 60.]

[50] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," *arXiv:2205.01068 [cs]*, May 2022, arXiv: 2205.01068. [Online]. Available: http://arxiv.org/abs/2205.01068 [Page 60.]

[51] "Calculating "FLOP" using Intel® Software Development Emulator (Intel® SDE)," Apr. 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/calculating-flop-using-intel-software-development-emulator-intel-sde.html [Page 61.]

Appendix A

Bash Script to send requests in the experiment

Listing A.1: bash script used for generating requests

```
#!/bin/zsh

# Use $RANDOM for random number register

text_data='{"v1":"text0","v2":"text1","v3":"text2
↪ ","v4":"text3"}'

# Scenario 1,    Simulate a period of 500 requests
↪ to a Apples Siri like Application.
#                The scenario sends one text per
↪ requests with a sleep in between of
#                value 1000 - 0 milliseconds. 500
↪ inferences


RANDOM=42 # Make sure script use the same text
↪ sequence for all tests
for i in {1..500}
do
 #request
 json='{"instances":[{"text":'$(jq ".v$[${RANDOM
↪ }%4+1]" <(echo "$text_data"))'}]}'
```

```
 # Send request
  curl -H "Content-Type: application/json" -X POST
↪ -d "$json" 192.168.10.242:8080/predict

 # Sleep 1000ms - 0ms
 sleep $(echo "(($RANDOM%1000)+1)*0.001" | bc)
done


# Scenario 2, Simulate burst requests. 10 requests
↪ every 2 second for 10 batches. 100 inferences.

RANDOM=42
for i in {1..10}
do
 # init request
 echo '{"instances":[]}' > text.json

 # build reques of 10 texts to classify
 for i in {1..10}
  do
   data='{"text":'$(jq ".v$[${RANDOM}%4+1]" <(echo
↪ "$text_data"))'}'
   echo $(jq --argjson str "$data" '.instances[.
↪ instances | length] | = . + $str' text.json) >
↪ text.json
  done

 # Send request
 curl -H "Content-Type: application/json" -X POST -
↪ d @./text.json 192.168.10.242:8080/predict

 # Sleep 2 seconds
 sleep 2
done


# Scenario 3, Simulate one computation heavy
↪ request. 500 inferences made in one request.
```

```
RANDOM=42

# build the request with 500 texts to classify. Run
↪  this separate to not waste time waiting for
↪ request to build during testing.
#echo '{"instances":[]}' > text.json
#for i in {1..500}
#do
#         data='{"text":'$(jq ".v$[${RANDOM}%4+1]"
↪ <(echo "$text_data"))'}'
#        echo $(jq --argjson str "$data" '.instances
↪ [.instances | length] |= . + $str' text.json) >
↪ text.json
#done

curl -H "Content-Type: application/json" -X POST -d
↪  @./text.json localhost:8080/predict

# All scenarios are done
echo Done!
```

Appendix B

How to set up the python environment

Listing B.1: requirements.txt used for the inferencing computer

```
datasets==2.1.0
Flask==2.1.1
frozenlist==1.3.0
huggingface-hub==0.5.1
importlib-metadata==4.11.3
keras==2.8.0
Keras-Preprocessing==1.1.2
Markdown==3.3.6
MarkupSafe==2.1.1
matplotlib==3.5.2
multidict==6.0.2
numpy==1.22.3
pandas==1.4.2
pyarrow==7.0.0
pytz==2022.1
PyYAML==6.0
tensorboard==2.8.0
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow-macos==2.8.0
tensorflow-metal==0.4.0
termcolor==1.1.0
tf-estimator-nightly==2.8.0.dev2021122109
```

```
tokenizers==0.12.1
torch==1.11.0
torchvision==0.11.3
transformers==4.18.0
yarl==1.7.2
```

Appendix C

Data analysis source code

Listing C.1: Source code used for the data analysis

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
# Usage: Put energy measurements into data object
    ↪ and then select which feature to analyse as
    ↪ ys.
#
#
# 2022.04.22
#

from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
import pandas as pd

NONE = 0
DISTILBERT = 1
BERT = 2
ALBERT = 3
ROBERTA = 4
BERTLARGE = 5

# Put in data to be analysed
```

```
data = {
        'GFLOP': [0,...],
        'cpu_dram': [0,...],
        'gpu': [0,...],
        'cold-start': [0,...],
        'inference': [0,...]
}
# Create pandas data frame
energy_df = pd.DataFrame(data)

# Examine cpu_dram data
xs = energy_df['GFLOP']
ys = energy_df['cpu_dram']

# Linear Regression analysis
# Null value will have huge weight as we want
    ↪ intercept at 0.
# Prepare the coefficient matrix according docs
# https://docs.scipys.org/doc/numpys/reference/
    ↪ generated/numpys.linalg.lstsq.html
A = np.vstack([xs, np.ones(len(xs))]).T

# Then we prepare weights for these points. And we
    ↪ put all weights
# equal except the last one (for added anchor point
    ↪ ).
# In this example it's weight 1000 times larger in
    ↪ comparison with others.
W = np.diag(np.ones([len(xs)]))
W[NONE,NONE] = 1000.

# And we find least-squares solution
[m, c], [resid] = np.linalg.lstsq(np.dot(W, A), np.
    ↪ dot(W, ys), rcond=None)[:2]
# calculate R-squared
r2 = 1 - resid / (ys.size * ys.var())

# Create plot
plt.plot(xs, ys, 'o', label='model')
```

```
plt.plot(xs, m * xs + c, 'r', label='fitted line')
plt.figtext(.62, .2, f'Slope:         {m:.3f}',
    ↪ fontfamily='sans-serif', fontsize="large")
plt.figtext(.62, .25, f'R-squared: {r2:.3f}',
    ↪ fontfamily='sans-serif', fontsize="large")
plt.xlabel("GFLOPs", fontsize=12)
plt.ylabel("Joules", fontsize=12)
plt.legend()
# plt.savefig("cpu_energy.svg")
plt.show()
```

TRITA-EECS-EX- 2022:00

For DIVA

{
"Author1": { "Last name": "Axberg",
"First name": "Tom",
"Local User Id": "u1jnfm8d",
"E-mail": "taxberg@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
}
},
"Cycle": "1",
"Course code": "IA150X",
"Credits": "15",
"Degree1": {"Educational program": "Degree Programme in Information and Communication Technology"
,"programcode": "CINTE"
,"Degree": "Bachelors degree"
,"subjectArea": "Technology"
},
"Title": {
"Main title": "Deriving an Natural Language Processing inference Cost Model with Greenhouse Gas Accounting",
"Subtitle": "Towards a sustainable usage of Machine Learning",
"Language": "eng" },
"Alternative title": {
"Main title": "Härledning av en Kostnadsmodell med växthusgasredovisning angående slutledning inom Naturlig Språkbehandling",
"Subtitle": "Mot en hållbar användning av Maskininlärning",
"Language": "swe"
},
"Supervisor1": { "Last name": "Gindele",
"First name": "Oliver",
"E-mail": "oliver.gindele@datatonic.com",
"Other organisation": "Datatonic"
},
"Supervisor2": { "Last name": "Västberg",
"First name": "Anders",
"Local User Id": "u1ft3a12",
"E-mail": "vastberg@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"Examiner1": { "Last name": "Maguire Jr",
"First name": "Gerald Quentin",
"Local User Id": "u1d13i2c",
"E-mail": "maguire@kth.se",
"organisation": {"L1": "School of Electrical Engineering and Computer Science",
"L2": "Computer Science" }
},
"Cooperation": { "Partner_name": "Datatonic"},
"Other information": {"Year": "2022", "Number of pages": "xviii,80"},
"Series": { "Title of series": "TRITA-EECS-EX" , "No. in series": "2022:00" },
"Opponents": { "Name": "Emre Edward Leander"},
"Presentation": { "Date": "2022-06-02 13:00"
,"Language":"eng"
,"Room": "via Zoom https://kth-se.zoom.us/j/66357519795"
,"Address": "Isafjordsgatan 22 (Kistagången 16)"
,"City": "Stockholm" },
"Number of lang instances": "2",
"Abstract[eng]": €€€€


```
The interest in using \gls{SOTA} \glspl{PLM} in product development is growing. The fact that
developers can use \glspl{PLM} has changed the way to build reliable models, and it is the go-to
method for many companies and organizations. Selecting the \gls{NLP} model with the highest accuracy
is the usual way of deciding which \gls{PLM} to use. However, with growing concerns about negative
climate changes, we need new ways of making decisions that consider the impact on our future needs.
The best solution with the highest accuracy might \textit{not} be the best choice when other
parameters matter, such as sustainable development.

This thesis investigates how to calculate an approximate total cost considering \gls{OPEX} and
\coo~emissions for a deployed \gls{NLP} solution over a given period, specifically the inference
phase. We try to predict the total cost with \glspl{FLOP} and test \gls{NLP} models on a
classification task. We further present the tools to make energy measurements and examine the metric
\glspl{FLOP} to predict costs.

Using a bottom-up approach, we investigate the components that affect the cost and measure the energy
consumption for different deployed models. By constructing this cost model and testing it against
real-life examples, essential information about a given \gls{NLP} implementation and the relationship
between monetary and environmental costs will be derived.

The literature studies reveal that the derival of a cost model is a complex area, and the results
confirm that it is not a straightforward procedure to approximate energy costs. Even if a cost model
```

was not feasible to derive with the resources given, this thesis covers the area and shows why it is complex by examine \glspl{FLOP}.

€€€€,
"Keywords[eng]": €€€€
Machine Learning, Inference, Inferencing, Natural Language Processing, Pre-trained Language Model, Greenhouse Gas, Software Energy Measurement, Floating Point Operations, Green-AI, Green-IT, OPEX €€€€,
"Abstract[swe]": €€€€

Intresset att använda \gls{SOTA} \glspl{PLM} i produktutveckling växer. Det faktum att utvecklare kan använda \glspl{PLM} har förändrat sättet att träna tillförlitliga modeller på och det är den bästa metoden för många företag och organisationer att använda \gls{SOTA} Naturlig Språkbehandling (NLP). Att välja \gls{NLP}-modellen med högsta noggrannhet är det vanliga sättet att bestämma vilken \gls{PLM} som ska användas. Men med växande oro för miljöförändringar behöver vi nya sätt att fatta beslut som kommer att påverka våra framtida behov.

Denna avhandling undersöker hur man beräknar en ungefärlig totalkostnad med hänsyn till \gls{OPEX} och \coo~utsläpp för en utplacerad \gls{NLP}-lösning under en given period, dvs slutledningsfasen. Vi försöker förutspå den totala kostnaden med flyttalsoperationer och testar mot en klassificerings uppgift. Vi undersöker verktygen för att göra mätningar samt variabeln Flyttalsoperationer för att förutspå energiförbrukning.

€€€€,
"Keywords[swe]": €€€€
Maskininlärning, Slutledning, Naturlig Språkbehandling, Förutbildad språkmodell, Växthusgas, Energimätning av Mjukvara, Green-AI, Green-IT. OPEX €€€€,
}