MSc Artificial Intelligence

Exploring energy consumption, inference time and accuracy in code Large Language Models

by

Pepijn de Reus

12160849

August 13, 2024

Credits: 36 EC

January 2024 - June 2024

Supervisor:
Dr. A.M. Oprescu

Examiner:
Dr. W.H. Zuidema

Universiteit van Amsterdam

Contents

Abstract

Next to a significant increase in users, AI also has an increasing energy consumption. Microsoft and Google reported 30.9%, respectively 48% more carbon emissions last year due to increased energy requirements for AI. To reduce the dangerous consequences of climate change, the Paris Agreement aims to limit global warming to 1.5 degrees by limiting carbon emissions. Apart from ChatGPT, Large Language Models (LLMs) are widely adopted in the programming community with plug-ins such as GitHub Copilot. As very little work describes the energy consumption of code LLMs during inference, we formulate the research question: how can we reduce the energy consumption of code LLMs to limit global warming? In this thesis, we aim to reduce the energy consumption of StarCoder2 models with minimal harm to accuracy by reducing training time, compressing weights via quantisation and pruning the last layers. Our experiments indicate that none of our experiments succeeds in reducing energy consumption without compromising accuracy. Nevertheless, we see possibilities and provide suggestions to optimise compressing weights by quantisation.

Chapter 1

Introduction

From scheduling transport in logistics to diagnosing diseases: Artificial Intelligence (AI) has proven useful for many fields by extracting relevant patterns from big piles of data (Gutiérrez et al., 2024). Therefore in recent years AI, specifically generative AI, has gained unprecedented attention from the scientific community and the general public. With the introduction of ChatGPT (OpenAI, 2022), using AI became accessible for the general public who does not have coding experience. For many users, the interaction with ChatGPT was the first interaction with AI. As a result, ChatGPT gained over 180 million users in the first year (Ghassemi et al., 2023). Quickly, users discovered that the interaction with ChatGPT could help with making homework, formal emails and reporting. Some research suggests that Large Language Models (LLMs) can increase productivity and output (Noy and Zhang, 2023).

A special type of LLM can help programmers in their work by specialising the language model of programming languages. These models, known as code LLMs, are trained to develop code instead. of natural language. One of the biggest code-sharing platforms, GitHub, deployed its code LLM called GitHub Copilot in 2021. GitHub Copilot is one of the most-used code LLMs and users report increased satisfaction and productivity when working with GitHub Copilot (Kalliamvakou, 2022).

Besides many benefits, the recent advancements in AI also bring societal challenges. In the information age, truthfulness and transparency information are important to review information on its content. Since the wide adoption of chat LLMs such as Chat-GPT, there have been numerous articles that describe reviews, theses and even complete scientific articles that are written by generative AI. During these developments, the European Parliament drafted the first regulation on AI that categorises AI systems based on risk. The AI act aims "... to make sure that AI systems used in the EU are safe, transparent, traceable, non-discriminatory and environmentally friendly. AI systems should be overseen by people, rather than by automation, to prevent harmful outcomes." (European Parliament, 2023b). The AI Act identifies four risk types:

- **Unacceptable risk**, AI systems considered a threat to people, such as biometric identification and facial recognition, will be banned.
- **High-risk**, AI systems that negatively affect safety or fundamental rights, such as critical infrastructure and education, will be assessed before market release and during their lifecycle.
- **Limited risk**, AI systems that lack transparency, such as chatbots, need to ensure that humans are informed that content is AI-generated.
- **Low or minimal risk**, AI systems that pose low to minimal risk, such as spam filters and video games, are to be used freely.

Current research often focuses on mitigating risks and problems for LLMs, but little work is conducted on the energy consumption of LLMs that generate code. As code is a form of language which heavily relies on structure and syntax, these models are an interesting choice for our research. On top of that, governments are increasingly using code and AI for public administration (Höchtl et al., 2016), which makes it likely that code LLMs will fall under the high-risk category from the EU AI Act.

Increasingly, scientists call to reduce the energy consumption of AI, as the use of AI significantly impacts global energy consumption (Project, 2023; de Vries, 2023a). Alphabet reported that its energy bill increased over tenfold due to the required computational power to train AI models (Verhagen et al., 2024), and both Microsoft and Google reported an increased scope-3 carbon emissions related to the current AI development (Microsoft, 2024; Google Sustainability, 2024).

In the past year, 2023, global temperatures for the first time exceeded the 1.5 Celsius warming limit of the Paris Agreement for consecutive months (Poynting, 2024). Overall, global average temperatures were 1.48 degrees Celsius above pre-industrial, leaving a very thin margin to remain within the Paris Agreement (Copernicus, 2024). This makes it very likely that 1.5 degrees Celsius warming will not be achieved in 2050, as per the Paris Agreement, but already in this decade (Hansen et al., 2023). The World Meteorological Organisation assigns a probability of 80% that 1.5 degrees Celsius of warming will be reached before 2029 (WMO, 2024). Already this causes extreme heatwaves, which are now 35 times more likely than before the industrial age (Pinto et al., 2024). This global warming is then expected to exceed 2 Celsius before 2050 (Hansen et al., 2023), increasing the likelihood of irreversible changes such as large-scale death of coral reefs and permanent melting of the Arctic sea (Lenton et al., 2019). Rising energy consumption and rising global temperatures go hand-in-hand, hence there is an urgent demand to reduce the carbon emissions of AI.

To contribute to this, the EU AI Act requires all high-risk AI systems to report their energy consumption, use of resources and other impacts throughout their lifespan (European Parliament, 2023a). This thesis explores the interaction between accuracy and energy consumption in code LLMs. We analyse the entire pipeline: from training to usage. We formulate the following research question:

How can we reduce the energy consumption of code Large Language Models with minimal harm to accuracy?

We divide the main research question into the following sub-questions:

RQ1: How does training duration impact energy consumption and the accuracy of a code LLM?

RQ2: How does compressing model weights impact accuracy and energy consumption at inference time?

RQ3: How does removing layers in a code LLM impact accuracy and energy consumption during inference?

Outline

First, we present the foundation and relevant theoretical background in Chapter 2. Then, we provide our experimental design in Chapter 3 and the experimental setup Chapter 4, followed by the results in Chapter 5. We interpret and discuss the results in Chapter 6, where we also state our limitations and provide directions for future work. We then provide an overview of similar research in Chapter 7. Finally, we wrap up and summarise this thesis in the conclusion in Chapter 8.

Chapter 2

Theoretical background

We present the theoretical background forming the foundation of this thesis. First, we introduce Natural Language Processing and the recent developments that led to LLMs. We then discuss the impact of AI on climate change, followed by a section on measuring the energy consumption of code. We then list the three most-common methods to compress a model: distillation, pruning and quantisation.

2.1 Evolution of Natural Language Processing

Natural Language Processing (NLP) is one of the oldest areas of AI with the first paper dating back to 1948 where Claude Shannon (1948) presented ideas that would later form the basis of information theory. Shannon then presented the idea of N-gram language models (Shannon, 1951), allowing models to generate a coherent sequence of words. These frequency-based models were useful and simple to use, which made them widely used until the late 2000s (Noguer i Alonso, 2024). However, these models used sparse vectors to represent words, meaning that each word was added to the vector such that the representation of a word contained many zeros. This means that large vocabularies largely consist of zeros, making the computations unnecessarily complex. For example, if words occur very rarely we still store them in the vocabulary which leads to many words used rarely but making up a large size of the vocabulary. With a large vocabulary, the computations take more time and become more complex, known as the *curse of dimensionality*.

With the rise of neural models, words could be represented in a high-dimensional vector space such that related words have a similar representation in the vector space (Goodfellow et al., 2016). This allowed for compact vectors, in the context of NLP also known as *word embeddings*, overcoming the curse of dimensionality encountered for N-grams. As such, word embeddings are learned vector representations of words positioned in such a way that semantically related words are proximate in an embedding space. Word embeddings can lead to significant performance improvements because of reduced complexity in computations (Mnih and Kavukcuoglu, 2013). Word embeddings as dense vector representations outperform sparse vector representations in every NLP task (Jurafsky, 2000; Mnih and Kavukcuoglu, 2013). Often used is the representation of Global Vectors, or GloVe, that uses co-occurrence matrices to train word embeddings that can represent semantic relationships (Pennington et al., 2014).

Apart from word embeddings, neural networks, especially Deep Neural Networks (DNNs), learn to represent complex functions by finding optimal parameters (Sutskever et al., 2014). Though DDNs are powerful, they can only be applied to problems where the input and target (e.g. next-token in a sentence) are known *a priori*. In tasks such as speech recognition or machine translation, the sequence length is not known beforehand. The need to know the dimensionality of the input and target beforehand is a limitation of DNNs in these tasks.

Similar to this limitation, DNNs cannot use previous data for current predictions. For example in the sentence "I grew up in France, I speak fluent *French*", the context of the previous words matter for the prediction of the word French[1]. As the sequence becomes longer, DNNs have more and more trouble capturing these dependencies over time (Bengio et al., 1994), also known as *long-term dependencies*. This issue with long-term dependencies is linked to the issue of vanishing gradients. As neurons have a value between 0 and 1, updating the neuron with a small factor can lead to shrinking values over epochs. As this value decreases, the model becomes less able to update its parameters and to learn complex representations of the data (Qi et al., 2023). Exploding gradients on the other hand are caused by exponential growth in weight updating. Large gradients lead to large and volatile updates, again reducing the model's ability to represent complex data (Qi et al., 2023).

2.1.1 Long Short-Term Memory (LSTM)

With the introduction of the Long Short-Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, 1997) these limitations can be resolved (Sutskever et al., 2014). We briefly introduce the LSTM architecture, based on the renowned tutorial of Christopher Olah[1]. For a more extensive introduction, we refer to Staudemeyer and Morris (2019).

Figure 1 depicts a schematic overview of the LSTM architecture with three cells. At the top of the middle cell, a vector transfers the data from left to right as seen in a Recurrent Neural Network (RNN). Below this arrow, three operations use a sigma (σ) layer that controls how much information flows through the vector. The left arrow is the 'forget gate', and decides whether the previous information has to be used. For example, when a new object is introduced in the text the LSTM might forget the memory of the cell to initialise a new object. The gate right of the 'forget gate' is the input gate, which decides the current cell state and adds this to the flow of information on top of the cell. Finally, at the rightmost gate, the LSTM combines the current cell state and the filter by the forget gate to obtain the output of this cell. Then, in case of next-token prediction, the LSTM proceeds to the cell to the right to repeat the process.

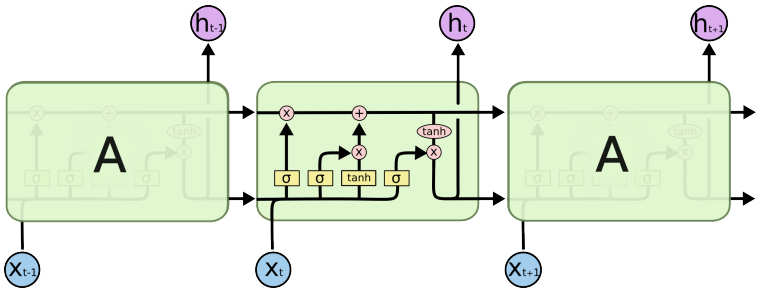


Figure 1: A schematic overview of the LSTM architecture with three sequential LSTM cells. Obtained from (Olah, 2015).

Using two different LSTMs, one LSTM reads the input sequence to generate (fixed) large dimensional vector representation and another LSTM extracts the output sequence from this large dimensional vector representation. This introduced the bidirectional Long Short-Term Memory (biLSTM) (Sutskever et al., 2014), resolving the limitation of DNNs with sequentiality. On the other hand, both architectures are prone to memory loss for long-range dependencies. By reversing the word order of the source sentence, the authors introduce "many short-term dependencies that made the optimization problem much simpler" (Sutskever et al., 2014).

The first LLM was ELMo (Embeddings from Language Models) (Peters et al., 2018a), a biLSTM architecture that allows taking context into account because it processes language both forward and backward. Previous architectures used a fixed word embedding and then predicted based on the output, with the biLSTM the model could access internal layers. Moreover, by using deep contextualized representations ELMo allows more context into the prediction. With

[1]Example obtained from `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

this approach, ELMo significantly outperformed previous state-of-the-art approaches. On top of that, this paper started the tradition of naming LLMs after Sesame Street characters.

2.1.2 Attention mechanism

Until 2015 the field of neural machine translation used an encoder-decoder architecture based on RNNs to translate a source sentence (Bahdanau et al., 2016). The encoder neural network encodes a source sentence into a fixed-length vector, after which the decoder outputs a translation of the encoded fixed-length vector. Because this encoder used a fixed-length vector, the architecture struggled with longer sequences. To solve this limitation of long-term dependencies, Bahdanau et al. presented the *attention mechanism* (2015) (Bahdanau et al., 2016). In essence, with the attention mechanism the model learns which tokens are relevant for the translation task. With this approach, the attention mechanism improved the performance on longer sequences.

The attention mechanism got in the spotlight with the release of the transformer architecture (Vaswani et al., 2017). The transformer architecture got rid of the recurrent neural networks and convolutions and solely used the attention mechanism, hence the name of the paper "Attention is all you need".

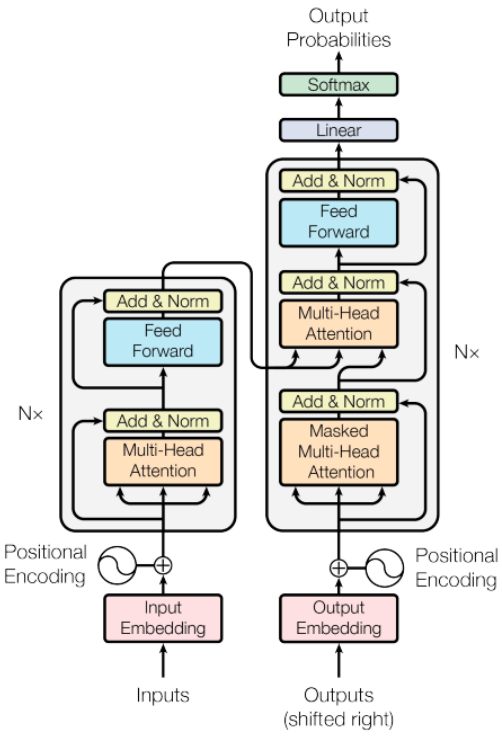


Figure 2: A schematic overview of the transformer model architecture, obtained from Vaswani et al. (2017).

Machine translation models consist of two parts: an encoder and a decoder. In Figure 2, the encoder is to the left and the decoder to the right. Suppose we wish to translate a sentence into another language. In this case, the encoder takes the source sentence as input whereas the decoder takes the target sentence as input. The target sentence will first be empty, and for each step in the process, we obtain a new word that we add to the target sentence. With the RNN approach, the sequence would be translated sequentially in one run as seen in Figure 1. But now with the transformer architecture, we model each word separately.

First, the input is embedded, as seen at the bottom-left of Figure 2. Continuing with the translation example, all words are converted into word vectors. After this step, the positions are lost in the embedding process and hence are added directly afterwards (Positional Encoding). The position provides information on where in the sentence a word belongs, consequently the positions are attached after encoding. Then, the embedding is processed in the Multi-Head Attention mechanism that defines a hidden state for the sentence. In this step, the model chooses which words in the sentence are important. Then these two hidden states of the source sentence (left) and the target sentence (right) are combined in the next Multi-Head Attention step on the right of the image. Here, a dot-product is computed between the keys, values (both source) and queries (target). Equation 1 defines the scaled dot-product attention used in the transformer.

$$\text{Attention}(Q, K, V) = \text{softmax}\frac{QK^T}{\sqrt{d_k}}V \tag{1}$$

As a dot product measures the angle between two vectors, we can then see whether the keys and values are aligned with the query. And so, at the end of the model, we select the keys and value pairs that best align with the queries. In other words, the words that best align the source sentence with the target sentence. This focus on attention removed the need for recurrence and convolutions entirely. Where an RNN would learn dependencies between words based on the sequence, a transformer learns dependencies based on the word itself. This makes the transformer superior in comparison to prior architectures such as LSTMs in both prediction accuracy as well as training time (Vaswani et al., 2017).

The scaled dot-product attention is thus very effective in learning representations. However, with the attention as defined in Eq. 1 we only form representations on a single word. To capture different aspects of the relationships between words, the authors linearly projected the queries, keys and values in different dimensions and then performed the attention operation in parallel. After this parallel processing, the representations are concatenated and then linearly projected again as depicted in Figure 3. Following these steps, we obtain the outcome of the multi-head attention. Because of this multi-headed attention, the model can divide its attention across these various representations and positions. When attention is used across multiple layers the model learns to attend to the relevant pieces of input in each layer, improving the representation while the input is processed. Because of this stacking of attention layers predictions become more accurate over time, leading to a significant improvement over earlier language models.

Because the authors limit the dimensionality for each head, the total computational cost for 8 heights in parallel is similar to the cost of single-head attention with full dimensionality. Nevertheless, the operation is still computationally costly which results in the user having to wait until the model finishes its prediction. Quickly after the release of the transformer authors tried to reduce this computational period, called the inference time, for example, by having multiple queries heads for one key head and one value head. This configuration is called Multi-Query Attention (MQA) and reduces the memory operations required for loading the key and value tensors (Shazeer, 2019). This makes inference up to ten times faster with a minimal loss in accuracy. A similar approach groups queries and approaches the performance of multi-head attention with the speedup of MQA (Ainslie et al., 2023). These developments show that simple modifications can significantly improve the efficiency of models with a minimal loss in performance.

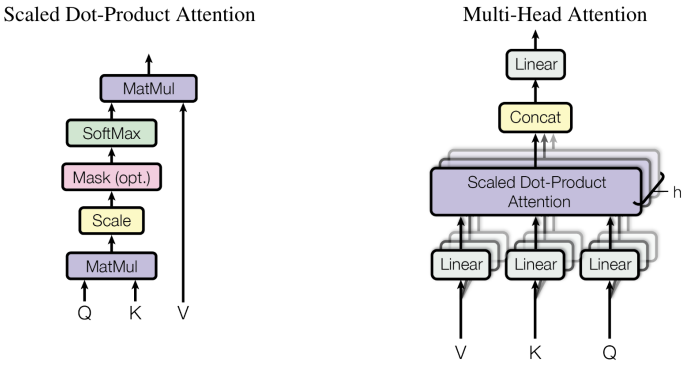


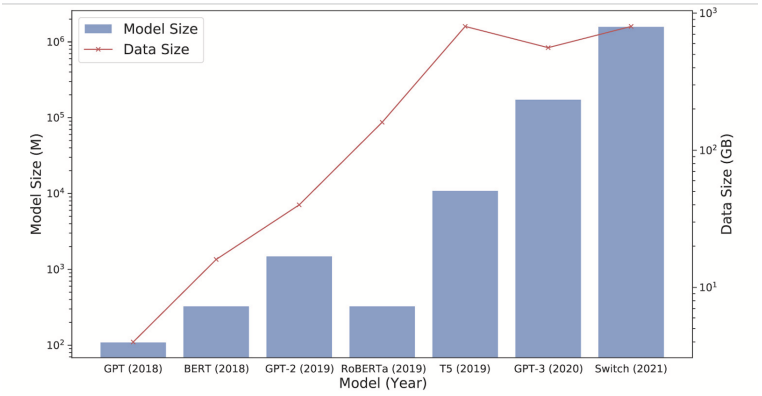
Figure 3: The Scaled Dot-Product Attention is displayed to the left and the Multi-Head Attention mechanism to the right. Figure obtained from (Vaswani et al., 2017).

2.1.3 Large Language Models

Where Large Language Models (LLMs) were trained on specific tasks, e.g. neural translation or language modelling, Embeddings from Language Models (ELMo) was the first model to generically pre-train the model and then perform task-specific fine-tuning (Peters et al., 2018a). ELMo is a pre-trained bidirectional LSTM which formed a new type of complex word representations. This approach allowed for pre-trained word embeddings to be shared for a variety

of tasks, improving the state-of-the-art performance on various benchmarks of that time. The development of ELMo had a huge impact on the field, in the beginning the bidirectional LSTM would outperform the transformer architecture because of the word representations (Peters et al., 2018b). After the transformer paper in 2017, OpenAI developed the first LLM using this transformer architecture: Generative Pre-trained Transformer 1 (GPT-1) (Radford et al., 2018). With 117 million parameters, the model was one of the biggest language models of that time (2018). A few months later, Google introduced BERT: Bidirectional Encoder Representations from Transformers (Devlin et al., 2019). BERT had over 340 million parameters and was, like GPT-1, generically pre-trained and then fine-tuned for specific tasks. Again, BERT improved the state-of-the-art for benchmarks and showed the importance of bidirectional training.

With the first LLM, GPT-1, released in 2018 having 'just' 117 million parameters, we see a clear trend in growing model size and data size over time. As depicted in Figure 4, the growing logarithmic scale shows a clear trend on larger and larger language models. Though this trend seemed to reach a limit with the release of GPT-3's 175 billion parameters (2020), Switch released the biggest LLM so far with 1.57 trillion (1.57 E12) parameters in 2021. However, this cannot be said with certainty as various developers of state-of-the-art LLMs such as GPT-4 do not disclose their model architectures anymore due to "the competitive landscape and the safety implications" (OpenAI, 2024). With these increasingly large models, accessibility to non-commercial users is at stake because the models are so big they cannot be used without access to powerful GPUs.



(b) The model size and data size applied by recent NLP PTMs A base-10 log scale is used for the figure.

Figure 4: Graph depicting the growth in base-10 log scale for both model size and data size of LLMs since GPT-1 in 2018 until 2021. Obtained from (Han et al., 2021).

While first used for classical NLP tasks, LLMs are being integrated into search engines. For example Perplexity allows users to "... directly pose your questions and receive concise, accurate answers backed up by a curated set of sources."[2]. Though these developments seem useful at first, integrating search engines with a chatbot also provides challenges regarding sustainability. Google did not deploy this earlier as the expected energy consumption would increase up to tenfold (de Vries, 2023a). During this thesis, in May 2024, Google announced that they would enable AI during search (Reid, 2024).

Countering this trend, some plea for training smaller models for a longer period to increase accuracy (de Vries, 2023b). Scaling laws describe the relationship between model size and performance. Using the Chinchilla scaling laws (Hoffmann et al., 2022), De Vries claims that smaller models trained on more tokens are faster and cheaper in training. Smaller models also increase access to researchers with a limited GPU budget. By analysing the asymptotic trend in the scaling laws, the author expects smaller models to be equally capable as their bigger LLM family. Though this is a non-peer-reviewed article, the author's hypothesis is backed by a recent pre-print that describes scaling laws for LLMs (Allen-Zhu and Li, 2024). The authors showed that as models are more exposed to data, they become more likely to recall. By increasing the

[2]As described in Perplexity's FAQ

number of exposures of the data from 100 times to 1000 times, the authors claim that GPT2 can store 2 bits per parameter instead of 1 (Allen-Zhu and Li, 2024). This means that a model trained 10 times longer could be as effective as a model twice its size.

2.1.4 Phases of Large Language Models

Roughly speaking we can identify three phases in the development of LLMs: the training phase, the inference phase and the fine-tuning phase. We briefly introduce each phase and its related literature.

Training

The **training phase** is the first step and is well-documented in literature, with half of the sustainable AI papers focusing on this phase (Verdecchia et al., 2023). In the training phase, the model learns to predict the target and hence optimises its parameters. Especially with LLMs, this process can take days or even weeks (Liu et al., 2024). Hence during this process the parameters and current loss and accuracy are stored. This allows developers to review these metrics over time during the training process, for instance, to determine when the model starts overfitting to the data. Also, by storing checkpoints the developers can review hyperparameter settings by comparing the various checkpoints to find the optimal values (hyperparameter tuning).

User interaction

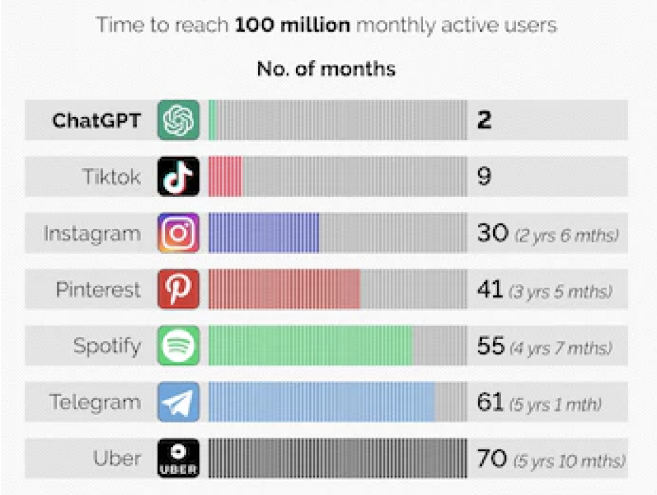


Figure 5: Time required in months for big applications to reach 100 million users. Image obtained from CNBC.

After this training phase, models are shared such that users can query the model. During this **inference phase**, the model generates output based on the user's queries. About 20% of the literature describes the inference phase in their examination of AI's energy consumption. Initially, the training phase was often the most consuming in terms of computation time and energy. Large neural networks require more computations as the models grow in size, a trend that has been seen in the last decade (Han et al., 2021). However, with the increasing use of models such as ChatGPT with over 100 million users (Ghassemi et al., 2023), the inference phase now covers millions of users that each perform multiple queries.

As depicted in Figure 5, the time in which ChatGPT gained 100 million users was also unprecedented, indicating how popular these AI applications are. Hence, with the increased use of AI comes a more substantial inference phase. Another example is Google Translate: with over one billion queries a day, the energy consumption of all these queries adds up to enormous amounts of energy (Luccioni et al., 2023a). Experts estimate that three days of ChatGPT consumes enough energy to match the training phase (de Vries, 2023a). On the other hand, there are reports of other LLMs such as BLOOM that have a significantly lower

energy consumption during inference. The author concludes that there is no general formula to reason on the energy consumption of the inference phase and that it may depend on various settings such as updating the weights during inference and parallelisation strategies. Further research should help clarify some differences reported across various LLMs (de Vries, 2023a).

Fine-tuning

Though generic training in the training phase is useful for capturing patterns in language and providing coherent text, next-token prediction does not necessarily align with human instructions. A model trained using next-token prediction likely predicts words related to the prompt, but this often leads to undesirable behaviour such as repeating the prompt, hallucination or ignoring instructions (Zamfirescu-Pereira et al., 2023; Ji et al., 2023). For example, a model prompted not to say ABC will likely output ABC because it learned during the training phase that these words are relevant. To overcome this, LLMs are fine-tuned after training.

Instruction-tuning is such a form of fine-tuning where models are learned to follow instructions. After instruction-tuning the model understands that, for example, it should not output words preceded by a negation. With instruction-tuning, zero-shot[3] performance can be improved significantly (Wei et al., 2022). Using human-based datasets such as the Internet Movie Database (IMDB), which contains actual movie reviews with all their jokes, sarcasm and exaggerations, models learn to follow instructions. By instruction-tuning using over 60 datasets, the authors improved the zero-shot performance of various LLMs (Wei et al., 2022).

However, a good text is hard to define and express in a training metric. Luckily, humans can judge whether a text is good and aligns with their desired outcome. User interaction with the model generates feedback, which can be used again to improve the model (Lambert et al., 2022). With the deployment of the model, user feedback is continuously incorporated into training such that the model learns to predict the output preferred by humans. We call this Reinforcement Learning from Human Feedback (RLHF). By constructing a reward function representing human preference, we can optimise the model after initial release by rewarding output preferred by humans (OpenAI, 2024). Human feedback is costly concerning time and money, therefore this phase only occurs after the model is trained and instruction-tuned.

2.2 AI and climate change

Climate change forms a major challenge to our future[4]. To understand climate change and the urgency we introduce current climate science, followed by the policies drafted to remain within 1.5 degrees of warming. We then link climate science to computer science, specifically regarding the energy consumption of AI.

By 2050, climate change will affect over 3.3 billion people and thereby also geopolitics, business and the economy (Delanoë et al., 2023). To fight these risks, the United Nations adopted the Paris Agreement in 2015 (UN, 2015). The Paris Agreement is an international framework that aims to reduce greenhouse gas (GHG) emissions such that global warming will be limited to 2 degrees Celsius, preferably 1.5 degrees Celsius, as this will minimise the risks of climate tipping points (Lenton et al., 2019). Climate tipping points are events that alter the state of the ecosystem in such a way that the change becomes irreversible (Lenton et al., 2008). For example, the melting of the Arctic ice will not only increase sea levels but also further

[3]Zero-shot performance is the performance of an LLM on the first encounter. Consequently, few-shot performance is the performance of an LLM with multiple prompts to steer the model's answer in the right direction.

[4]`https://www.nature.com/nclimate/`

increase global warming as there is less ice to reflect sunlight. Once this tipping point has been reached, ice will keep melting regardless of our efforts to stop it.

Most countries and sectors have adopted measures to comply with the Paris Agreement by reducing GHG emissions and hence attempting to limit global warming. Estimates of the ICT sector's carbon footprint range between 2% to 3.7% of global GHG emissions in 2021 (Freitag et al., 2021; Delanoë et al., 2023). With the recent advances in AI such as the release of ChatGPT, Midjourney and DALL-E these emissions are growing exponentially (Delanoë et al., 2023). Current reduction pathways of big tech companies are insufficient to reach the Paris climate goals (Project, 2023). Hence the energy consumption of, and GHG emissions attached to, AI are important and somewhat neglected. Current efforts to reduce the energy consumption of ICT and data processing are mostly targeted at academic readers (Verdecchia et al., 2023). Most literature on this topic presented scientific predictions, but as of now in 2024, the first evidence starts to confirm these claims. For example in 2023, Microsoft's scope-3 carbon emissions[5] increased by 30.9% since 2020 (Microsoft, 2024). Microsoft's biggest competitor Google also reported a 48% increase in greenhouse gas emissions since 2019 (Google Sustainability, 2024).

2.2.1 Energy consumption of AI

To zoom in from ICT to AI specifically, we provide literature regarding the energy consumption of AI. The negative external effects on climate are not limited to energy consumption, nevertheless, we narrow the scope of this research to the energy consumption of AI as scientists and machine learning and software engineers can impact this.

An often-cited paper is by Strubell et al. (2019) on the energy consumption of training large neural networks. The authors describe that the most 'computationally-hungry' models obtain the highest scores, as a result, most state-of-the-art models now require a vast amount of computational resources to train. The authors describe that ten years earlier a model could be trained on a laptop or server, but now requires specialised hardware which does not only require a lot of energy but also limits access. Using averages of data centre efficiency and the average amount of CO_2 emitted for a KWh, the authors computed the CO_2 emissions of training four LLMs: Transformer, ELMo, BERT and GPT-2. The authors found that a large transformer emitted around 284 000 kilograms of CO_2, a number often repeated in related work. The most important finding of the authors however is that they could not compute all emissions because many models lack transparency, not only on the hardware used but also on training time and hyperparameter search.

A systematic review of Green AI reports that most work on AI and environmental sustainability targets the training phase (Verdecchia et al., 2023). Of the 98 papers included in this review, 28 papers provide monitoring approaches to study the energy footprint of AI. 17 focus on tuning hyperparameters during training, where papers range from alternative training strategies to better GPU utilization during training. Other work includes model benchmarking, studying the deployment phase and the trade-off between energy and precision. Though the articles included in this review are heterogeneous and cover many aspects, about a third of the papers mention energy savings. In 17 out of 27 papers, these savings exceed 50%, indicating that significant reduction is possible by tweaking model architecture and training. However, most research on the energy efficiency of AI focuses on model training (Castaño et al., 2023; Verdecchia et al., 2023) and little is known about the energy consumption or emissions of using these large language models. One article reviewed the Hugging Face library and found that the reporting on carbon emissions is decreasing (Castaño et al., 2023). Less than 1% of the released models on Hugging Face report on their carbon emissions and none of the 1417 models

[5]Accounting for 96.5% of the total emissions of Microsoft.

analysed met certified energy efficiency criteria. One high-impact paper from 2021 mentions LLMs and their computational costs, the authors state that "There is a clear need to reduce the memory and computational requirements of large-scale DGMs to enhance accessibility and sustainability" (Bender et al., 2021).

By focusing on the energy consumption during training, we might find that general-purpose models have a lower energy consumption than training separate models. However, if the energy cost during inference is higher then we might still prefer fine-tuned models. One recent paper analyses the energy consumption of large general-purpose models (such as ChatGPT) during their inference for various ML tasks (Luccioni et al., 2023a). As expected, the authors conclude that models with more parameters require more energy during inference. Next to that, tasks that generate new content (e.g. text summaries or image generators) are the most energy-intensive. And, interestingly, the authors found that using multi-purpose models for discriminative tasks is more energy-intensive than using task-specific models. The authors therefore conclude that engineers should prefer task-specific models over general or multi-purpose models if the given task allows it.

The energy consumption of AI is increasing at such a steep pace that tech companies are considering other power sources to develop and deploy their models. The Wall Street Journal reported that in June 2024 about one-third of the nuclear power plants are negotiating with tech companies to buy nuclear power directly (Hiller and Herrera, 2024). With the increasing energy requirements of AI (Verhagen et al., 2024; Microsoft, 2024), there is an urgent call from scientists to study the energy consumption of AI. The use of AI can significantly impact global energy consumption (Project, 2023; de Vries, 2023a), which could sap the grid of critical resources (Hiller and Herrera, 2024).

2.3 Measuring energy consumption of code

As previously discussed, the energy consumption and the GHG emissions related to this energy consumption are often neglected. In this section we present various methods to measure the energy consumption of code. Before we dive in these methods we emphasise the difference between energy and power. Power can be measured at a given timestamp, energy is measured over a process. As we aim to reduce the energy consumption of LLMs we will stick with energy throughout this thesis.

2.3.1 Hardware-based

The most obvious method to measure energy consumption is by measuring the difference between the energy that goes in and comes out of a device. By measuring the difference during the execution of code, e.g. AI, one can derive the energy consumption. This hardware-based solution, also known as the *power plug method*, is often an expensive investment and inadequate for detailed analysis because background processes are also measured (Khan et al., 2018). Nevertheless, specifically for larger setups such as a data centre, the power plug method can be a suitable choice to measure the energy consumption of code.

2.3.2 Software-based

An alternative to the hardware-based approach is a software approach that measures the utilization of specific components such as the Central Processing Unit (CPU) and Graphics Processing Unit (GPU). If one knows the utilization percentage of such a component, this can be factored by the power consumption of this component to estimate the energy consumption. With this computation, we estimate the energy consumption but we do not directly measure the energy

consumption. Some papers indicate that software-based methods correlate up to 0.99 with hardware-based methods (Khan et al., 2018), but others suggest a deviation of 20% (Cao et al., 2020). So the estimates are not precise, nevertheless, they provide insight into the order of magnitude, allowing experts to contextualise the energy consumption of code. Two software-based methods are often discussed in related work: pyRAPL and CodeCarbon. We briefly introduce both.

pyRAPL

Using the Running Average Power Limit (RAPL) interface, available on Intel's Sandybridge chips from 2012 onwards, we can estimate the energy consumption of the chip (Weaver et al., 2012). Using this RAPL interface, an open-source software package built with Python (pyRAPL) reads these counters to estimate the energy consumption of Python code (Spirals, 2019). Khan et al. (2018) showed that these estimations correlate up to 0.99 with power plug measurements, making pyRAPL a reliable method to measure energy consumption (Khan et al., 2018). Using pyRAPL over the power plug method holds two advantages: the costs (in both time and financial aspects) are lower and the setup allows for measuring specific parts of the CPU. The CPU parts are depicted in Figure 6. For example, we can set pyRAPL to solely measure the energy consumption of the DRAM[6]. A disadvantage of pyRAPL is that it solely works on Intel CPUs, whereas most AI nowadays is trained on GPUs to speed up the machine learning processes (Lacoste et al., 2019).

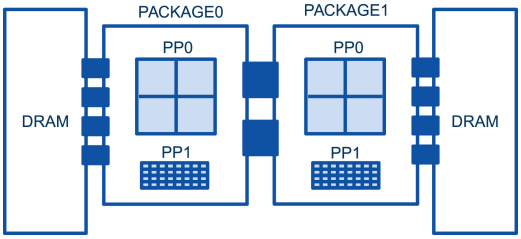


Figure 6: Overview of the Intel RAPL architecture for a dual-socket system, obtained from Veiga et al. (2018).

CodeCarbon

Another open-source software approach is CodeCarbon[7], which uses the RAPL (Weaver et al., 2012) interface to measure the energy consumption of code. Next to RAPL, CodeCarbon allows to measure the energy consumption of NVIDIA GPUs with System Management Interface (SMI). SMI is a command-line interface that is used to monitor the NVIDIA GPUs (NVIDIA, 2012), but can therefore also be used to query the GPU for its energy consumption. Therefore, CodeCarbon can measure the energy consumption of both CPUs and GPUs. Apart from measuring the energy consumption CodeCarbon also predicts the GHG emissions in CO_2-equivalents (CO2eq) (Schmidt et al., 2021). Next to the energy consumption, three aspects impact the carbon emissions of machine learning (Lacoste et al., 2019):

- **Location of the server and energy grid used**
  The location of the server is important in defining the energy mix[8]. As public data on the energy mix is often missing, the authors assume that the server is using the local energy grid for power. With this assumption, the authors use the public data on the local energy mix to estimate the emission of GHG for that server. The energy mix can vary greatly, leading to situations where some models can reduce their GHG emissions by a factor of 40 depending on the location of the data centre (Lacoste et al., 2019).

[6]Dynamic Random Access Memory, a method to temporarily store data close to the CPU.
[7]Available on Github.
[8]Following the definition by Wikipedia, we define energy mix as: "a group of different primary energy sources from which secondary energy for direct use - such as electricity - is produced."

- **Length of training**
  The amount of training time influences energy consumption. Often, fine-tuned models perform similarly or better than generic models (Howard and Ruder, 2018). With the training of LLMs often taking several days (Vaswani et al., 2017; Peters et al., 2018a), using pre-trained models could significantly reduce training time and thereby also energy consumption.

- **Computing infrastructure**
  With typical models often trained on various GPUs (Vaswani et al., 2017; Peters et al., 2018a), the hardware forms an important aspect of the GHG emissions. Not only does the number of GPUs matter, but as newer GPUs have more floating point operations per second (FLOPS) the architecture impacts energy (Lacoste et al., 2019).

Combining these elements, CodeCarbon provides the energy consumption and estimated GHG emissions related to the training task for both CPU and GPU. An advantage of CodeCarbon over the power plug method and/or pyRAPL is that CodeCarbon sets energy consumption in perspective by taking the source of energy (such as solar or gas) into account when this information is provided by the user.

2.4 Model compression

As environmental impact scales with model size (Bender et al., 2021), compressing models might be an effective way to reduce code LLM energy consumption with minimal harm to accuracy. We examine the three most common post hoc methods[9] that compresses model size.

2.4.1 Knowledge distillation

Together with the size of neural networks, the computation requirements to run a model have increased. Additionally, inference speeds tend to increase and become less efficient with bigger models (Li et al., 2023b; Bender et al., 2021). These limitations obstruct deploying DNNs to smaller devices such as phones or smartwatches. To resolve this problem, Hinton et al. (2015) compressed a model by extracting its learned knowledge into a smaller model (Hinton et al., 2015): distillation. The authors state that training AI is a different objective than deployment (or inference) of AI, as during training the algorithm will explore various paths in the network. However, during inference, these paths may not be needed as the knowledge is already in the network. With the softmax activation function, higher logits tend to push the lower values even lower. While this causes the model to be very accurate, the information on relatedness between classes is lost. To include this knowledge, the authors propose a parameter T (Temperature) to smoothen the logits in the same order of magnitude. These smoothed logits are then called *soft labels* whereas the original logits are named *hard labels*.

Then the authors train a distilled model using two objective functions: the regular softmax function and a function that compares the soft labels with the hard labels. Using this approach, the authors used a speech recognition task to prove that a distilled model can even outperform a baseline model regarding accuracy. And recent work shows that a distilled model consumes less energy than its original counterpart (Yuan et al., 2024). The authors also describe that apart from the energy consumption, the memory utilisation of distilled models is more efficient.

[9]Meaning that the method compresses the model *after* training

2.4.2 Pruning

As previously stated, deep learning models often have many complex paths to come to their prediction. The multi-headed transformer architecture is such an example where we observe an impressive performance on machine translation tasks (Vaswani et al., 2017). However, an analysis of why multi-headed work so well is challenging. To examine the effect of specific heads, Voita et al. (2019) identified the functionality of the various heads (Voita et al., 2019). Doing so, the authors found that heads have different roles such as positional heads (attending adjacent tokens), synthetic heads (attending tokens for a syntactic relationship) and attention to rare words. By removing, *pruning*, the heads they found the role of each head, after which the authors removed the heads without specialised roles. Surprisingly, this had minimal impact on model performance. In one task (WMT), the authors pruned nearly 75% of the encoder heads and 33% of the decoder heads "without any noticeable loss in translation quality".

Do fewer heads in the transformer architecture also lead to a more efficient model? Recent work suggests that indeed, with careful selection of parameters, pruning can reduce the model size (Gholami and Omar, 2023). The authors claim that significant reductions can be made without considerable compromise on performance. On top of that, the authors define "post-pruning fine-tuning strategies" that allow the model to enhance its potential to generalise.

2.4.3 Quantisation

Where knowledge distillation and pruning are techniques that modify the architecture of the model, quantisation leaves the architecture as is. Model parameters are often stored as floating point (FP32), meaning that 32 bits store the weight, as depicted in Figure 7.

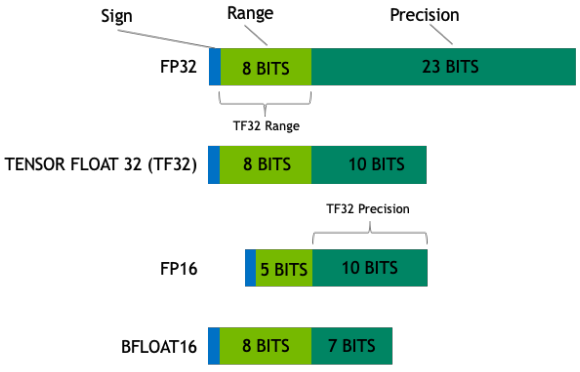


Figure 7: Overview of the representation of datatypes FP32, FP16, TF32 and BFLOAT16. Image obtained from NVIDIA.

But as discussed in Section 2.1.3, current LLMs exceed a billion parameters. Storing parameters as an 8-bit integer (LLM.int8) or even 4-bit float (FP4) can thus reduce the required storage memory for the parameters by 1/4 or 1/8, which allows the user to run the model on non-GPU devices such as personal computers or even mobile phones (Almeida et al., 2021). There is also an indication that quantisation makes the model more efficient during inference (Wang et al., 2024). Apart from reducing the model size during inference to reduce its energy footprint, quantisation is applied during regular machine learning operations as well. To increase training speed, 32-bit floating point (FP32) weights are updated using 16-bit floating point (FP16) gradients (Belkada and Dettmers, 2022), making the training twice as fast compared to using FP32 for gradients. Though initially, it seems that the model loses precision as we reduce the parameters, a recent paper shows that the performance of a quantised model using QLoRA can match the performance of ChatGPT up to 99.3% (Dettmers et al., 2023).

Absmax quantization

Following the example from the QLoRA paper (Dettmers et al., 2023), we provide an overview of the quantisation process. Suppose we have a tensor $\mathbf{X}^{\text{FP32}}$ stored in 32 bits, which we want to quantise into a tensor stored in Int8, $\mathbf{X}^{\text{Int8}}$ spanning from [-127, 127]. We then have to compress the tensor in such a way that we cover the entire range of the 32-bit tensor. Using normalisation, we can ensure this as follows:

$$\mathbf{X}^{\text{Int8}} = \text{round}\Big(\frac{127}{\text{absmax}(\mathbf{X}^{\text{FP32}})}\Big) = \text{round}(c^{\text{FP32}}, \mathbf{X}^{\text{FP32}}) \tag{2}$$

As Eq. 2 shows, we ensure that all numbers are normalised between the maximum of the Int8 format (127) and the absolute maximum of the tensor. This factor can then be stored as a quantisation constant c^{FP32}. Conversely, to retrieve the 32-bit tensor $\mathbf{X}^{\text{FP32}}$ from the quantised tensor $\mathbf{X}^{\text{Int8}}$, we use the inverse to dequantise the tensor as depicted in Eq. 3.

$$\text{dequantise}(c^{\text{FP32}}, \mathbf{X}^{\text{Int8}}) = \frac{\mathbf{X}^{\text{Int8}}}{c^{\text{FP32}}} = \mathbf{X}^{\text{FP32}} \tag{3}$$

Though this representation allows to store tensors in a format that requires less memory, there is also a limitation. Suppose we have a tensor with values between 0-10 and one outlier at 127. Using Eq. 2, we can immediately see that one outlier disproportionally impacts the quantisation constant. To overcome this, the input tensor can be divided into blocks with a quantisation constant for each block. This process is called **block-wise k-bit quantisation** (Dettmers et al., 2022).

For this, the model employs a 4-bit float to store weights and then performs *double quantisation* by quantising the quantisation constants. On top of this, the authors employed fine-tuning to enhance performance. The result is a model that used over 780GB of GPU memory to less than 48GB, without degrading the runtime or predictive performance compared to a 16-bit fully fine-tuned baseline. This paper shows the potential of quantisation not solely as a technique to reduce required memory for LLMs, but also the ability to do so without harming model performance.

Chapter 3

Methodology

To answer our research question **how to reduce the energy consumption of code LLMs with minimal harm to accuracy** we design experiments for each sub-question. Unfortunately, some setbacks disturbed our initial method which led to changes. We mention these in this section, the discussion follows in Chapter 6. In this section, we describe our approach and preferences. We aim to make this thesis reproducible using open-source models and methods discussed in scientific literature. If available, we prefer deploying methods that report on CO_2 emissions and methods that respect copyright over methods that do not report on these. Based on these principles, we provide an overview of our method in this section. The details and implementation follow in the experimental setup, Chapter 4.

3.1 Finding models and data

With the preference for open-source models, we consider models from the Hugging Face library. Preferably, the selected model shares its (intermediate) weights and architecture so that we can compress the model. To answer our first research question, whether we can reduce model training, we would have to train these models. However, with data sets often exceeding 30 terabytes (Lozhkov et al., 2024) and thousands of hours required for training (Bender et al., 2021; Rothe et al., 2020), we think training ourselves is infeasible both due time and budget constraints. Nevertheless, if we gain access to the intermediate checkpoints during training, we could still study the impact of training duration on the development of the model's accuracy.

3.2 Code evaluation

Code completion is often used to test a model's accuracy or performance. The model is then prompted with a piece of code, for example, the documentation of a function. With this description, the model generates the code of this function. For illustration, suppose we prompt the model to write a function that prints hello world. An example is given in Figure 8 where code snippet 3.1 represents a prompt and snippet 3.2 provides the expected output. The model copies the prompt and then predicts each next token in the sequence.

Where 'regular' LLMs are often evaluated with BLUE scores (Papineni et al., 2002), which measure how close machine-generated output is to human-generated output. The score rates between 0 and 1 and gets closer to 1 when a machine translation gets closer to a professional human translation. BLUE does not work well with code as BLEU measures the similarity between the output and the correct answer. For code, often many correct solutions are available for the same problem set. So, instead, we use pass@k (Chen et al., 2021) which measures the probability that for a given coding query, at least one of the top k outputs passes all unit tests.

```python
def print_hello_world():
    """This function prints the text
    'Hello World' in the terminal.
    """
```

Listing 3.1: Example of a docstring which could serve as a prompt for the model.

```python
def print_hello_world():
    """This function prints the text
    'Hello World' in the terminal.
    """
    print("Hello World")
```

Listing 3.2: Example of the output that would follow this prompt.

Figure 8: An imaginary example of a code completion task where the left snippet illustrates a prompt and the right snippet provides the output of a code LLM.

Consequently, we report the pass@1 for all models. Though unlikely with open-source models from Hugging Face, there is a risk that malicious code is generated. Compartmentalization prevents running malicious code on the server that can leak to other parts. Evaluating the code in a Docker container can thus reduce the odds of hacking.

3.3 Evaluation framework

While some papers draft large data sets to test their models, this does not guarantee that these models are also accurate. Rather, would we prefer having a smaller, but very accurate test, that truly examines the meaning of a given task and prompt? As such, we use EvalPlus (Liu et al., 2023) as a framework to evaluate the generated code. EvalPlus is based on Google's Mostly Basic Programming Problems (MBPP) (Austin et al., 2021) and OpenAI's HumanEval (Chen et al., 2021), which are the most widely studied code evaluation frameworks (Liu et al., 2023). These code evaluation frameworks are both based on code completion and provide a function description together with a few tests to steer the model in the right direction. EvalPlus expands on these two frameworks by providing a more rigorous evaluation: the authors of EvalPlus found that previous benchmarks were often insufficient on corner cases and provided imprecise problem descriptions for the prompt. EvalPlus gives a clear, unambiguous, instruction in natural language and then queries the model for code. We then sanitise the code by removing all outside of the function definition and checking for a return statement. The output is then subject to various unit tests that define the right data structures and outputs. If the code passes all of these tests, the output is labelled as correct. For a failure on one of these tests, the output is labelled incorrect. In the end, EvalPlus gives a pass@k score that depicts the mean success rate. A schematic overview of this process is depicted in Figure 9.

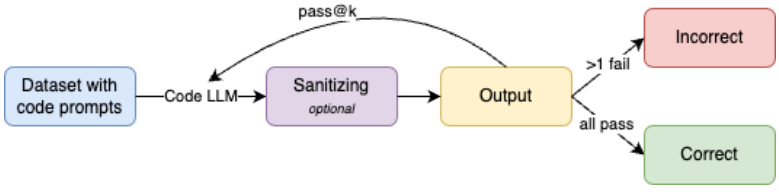


Figure 9: Schematic overview of a code evaluation framework. Prompts from the dataset are fed into the code LLM, the output is then often sanitised (or pre-processed) after which the sanitised output is evaluated with various tests. Depending on the variable k, several outputs can be ranked and if the top-k has at least one correct implementation the pass@k score will be 1.

The authors show that for nearly all tested LLMs, the pass@1 score is lower when evaluated with EvalPlus than with MBPP or HumanEval (Liu et al., 2023). Besides that, EvalPlus has

a public leaderboard that displays the pass@1 score for each model, making the framework accessible to the general public. We show the first test from EvalPlus in code snippet 3.3 to illustrate how code is tested. We converted $\backslash n$ to the start of a new line.

```
{
  "task_id": "HumanEval/0",
  "prompt": "from typing import List
  def has_close_elements(numbers: List[float], threshold: float) -> bool:
  """ Check if in given list of numbers, are any two numbers closer to each
   other than given threshold.
  >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
  False
  >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
  True
  """",
  "entry_point": "has_close_elements",
  "canonical_solution": "    for idx, elem in enumerate(numbers):
    for idx2, elem2 in enumerate(numbers):
        if idx != idx2:
            distance = abs(elem - elem2)
            if distance < threshold:
                return True

        return False",
  "test": "METADATA = {'author': 'jt', 'dataset': 'test'}

  def check(candidate):
  assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.3) == True
  assert candidate([1.0, 2.0, 3.9, 4.0, 5.0, 2.2], 0.05) == False
  assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.95) == True
  assert candidate([1.0, 2.0, 5.9, 4.0, 5.0], 0.8) == False
  assert candidate([1.0, 2.0, 3.0, 4.0, 5.0, 2.0], 0.1) == True
  assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 1.0) == True
  assert candidate([1.1, 2.2, 3.1, 4.1, 5.1], 0.5) == False"}
}
```

Listing 3.3: The first evaluation as provided in the EvalPlus framework. The original test sample is available on GitHub.

The first items in the test are the task identifier and the prompt. For the prompt, we see that a function is provided with its respective inputs and expected output, boolean in this example. The prompt then provides a small doc-string that explains what the function should do, followed by some examples of input and expected output. The code LLM then generates a solution, stored in the item *canonical_solution*. In the end, the task specifies some assertions that the solution should pass for the solution to be correct.

3.4 Compressing model weights

At least 8 methods compress model weights into smaller datatypes and are supported by the Hugging Face transformer library. Though there is some overlap, a method should be selected based on the use case. Figure 10 provides an overview of which method to pick in what use case. For example, of these 8 methods, 3 methods are not available 'on the fly', meaning that the models need to be calibrated after quantisation. Depending on the hardware, quantisation can take 5 minutes for a 350M parameter model but up to 4 hours to quantise a 175B parameter model. All methods are supported by transformers, but bytsandbytes (BNB) is the most accessible option which is why we stick to using BNB as quantisation method in this thesis.

Quantization method	On the fly quantization	CPU	CUDA GPU	RoCm GPU (AMD)	Metal (Apple Silicon)	torch.compile() support	Number of bits	Supports fine-tuning (through PEFT)	Serializable with 🤗 transformers	🤗 transformers support
AQLM	🔴	🟢	🟢	🔴	🔴	?	1 / 2	🟢	🟢	🟢
AWQ	🔴	🔴	🟢	🟢	🔴	?	4	🟢	🟢	🟢
bitsandbytes	🟢	🔴	🟢	🔴	🔴	🔴	4 / 8	🟢	🟢	🟢
EETQ	🟢	🔴	🟢	🔴	🔴	?	8	🟢	🟢	🟢
GGUF / GGML (llama.cpp)	🟢	🟢	🟢	🔴	🟢	🔴	1 - 8	🔴	See GGUF section	See GGUF section
GPTQ	🔴	🔴	🟢	🟢	🔴	🔴	2 - 3 - 4 - 8	🟢	🟢	🟢
HQQ	🟢	🟢	🟢	🔴	🔴	🟢	1 - 8	🟢	🔴	🟢
Quanto	🟢	🟢	🟢	🔴	🟢	🟢	2 / 4 / 8	🔴	🔴	🟢

Figure 10: An overview of the available quantisation methods compatible with the transformers module. Figure obtained from Hugging Face.

Chapter 4

Experimental setup

Following our methodology, we provide the experimental setup in this chapter. We first discuss the selected models, then the method for weight compression, and our experiments. To allow the reproduction of our experiments, we provide our system specifications.

4.1 Model selection

StarCoder2 is an open-source code LLM, generated by collaborating scientists in the BigCode project. StarCoder2 uses the Stack v2 as the training set for the model. StarCoder2 is transparent about training by mentioning both the computational costs as well as the energy impact in CO_2-equivalents[10]. StarCoder2 builds upon the previous StarCoder model but uses group-queried attention instead of multi-query attention. Details on the architecture are provided in the StarCoder2 paper and on GitHub. With StarCoder2's statements on open-source development and contributing to research, we picked their model for our work.

Several months into this thesis the authors informed us that they lost access to the server and could not provide the training checkpoints. To be able to answer our first research question, we reached out to the developers and corresponding authors for other models on Hugging Face with around 3B parameters to ask them for access to these checkpoints[11]. We reached out to the authors of the coding LLMs Phi-2 and Granite-3b, both did not reply to our request. Later, we found the Pythia Scaling Suite, a collection of models developed to facilitate interpretability research. Pythia (Biderman et al., 2023) releases intermediate checkpoints for each 1000 iterations. Pythia has a 2.8B model that we use to substitute for StarCoder2-3B as the number of parameters is alike. Likewise, StarCoder2-3B's performance was poor during pruning, so we included Phi-2 to see how a similar coding LLM behaves in this experiment.

Initially, StarCoder2 seemed the best-performing open-source model. But the field is developing at an enormous pace fast such that in the few months of this thesis, many models have been released of which many claim to improve StarCoder2's performance. These claims are not always a fair comparison as the tests are often included in training data (Zhou et al., 2023). New models will likely be released throughout the coming years. Therefore, we decide to stick to the StarCoder2-3B and StarCoder2-7B models to continue our analysis. The StarCoder2-15B model required parallelisation and would change our experimental setup, we therefore exclude this model from our experiments. We elaborate on this choice in Appendix C.

[10] CO_2-equivalents is "a standardized measure used to express the global-warming potential of various greenhouse gases as a single number, i.e. as the amount of CO_2which would have the equivalent global warming impact." (Lacoste et al., 2019)

[11] For completion, we asked the authors to either provide us with the checkpoints of these intermediate training weights or to run our code snippets on these weights on their server. We hoped the latter option would convince authors to share the results without having to share the intermediate training checkpoints.

4.2 Compressing the model weights

To optimise the model inference time and energy consumption, we deploy quantisation using the bitsandbytes library (Dettmers et al., 2021), available on PyPi and GitHub. We use quantisation over knowledge distillation and pruning as quantisation is relatively easy because it does not modify the model architecture. By obtaining the weights from StarCoder2 from Hugging Face, we apply quantisation using bitsandbytes to reduce the weights in both 4-bit and 8-bit format. We then compare these compressed models with their original counterparts to study the impact on energy consumption and accuracy (pass@1).

4.3 Experiments

Combining all these settings and design choices, we design three experiments:

- **Experiment 1**, we use the Pythia 2.8B model[12] with its various checkpoints during training to run evaluations on the model. We use the LM Evaluation Harness framework to perform 250 tasks in batches of 2 for the code2python task[13] from CodeXGLUE (Lu et al., 2021). Pythia is not a code LLM, but the code2python task queries the model to construct natural language comments for Python code. The model did not fit on one GPU but, contrary to StarCoder2-15B, the model can be parallelised to 4 GPUs by a parameter in the evaluation framework. By examining the smoothed BLEU-4 score, inference time and energy consumption we wish to analyse the development of these metrics over training time. Note that, as described in Section 3.1, we do not train the model ourselves but rather simulate the training behaviour by using intermediate checkpoints. We then measure the energy consumption and accuracy of these intermediate models during inference as if it was the definitive model to answer **RQ1: How does training duration and energy consumption impact the accuracy of a code LLM?**

- **Experiment 2**, we apply quantisation using bitsandbytes (Dettmers et al., 2021) to reduce the weights in 4-bit and 8-bit formats. Likewise, we evaluate the compressed models on pass@1 accuracy, inference time and energy consumption. The outcome answers **RQ2: How does compression model weights impact accuracy and energy consumption at inference time?**

- **Experiment 3**, we prune the last layers of StarCoder2-3B, StarCoder2-7B and Phi-2 until the pass@1 metric reaches 0. As pruning does not necessarily lead to reduced performance (Gholami and Omar, 2023), we reduce the model by pruning the last layers as these often contain context where earlier layers encode knowledge (Ju et al., 2024). With each pruned layer we evaluate the compressed models on pass@1 accuracy, inference time and energy consumption on the HumanEval+ task. Answering **RQ3: How does pruning layers of a code LLM impact accuracy and energy consumption during inference?**

[12]Nota bene, we anticipated using the StarCoder2-3B model instead as the authors declared sharing the entire model for research purposes. When we got in touch with the authors they said to have lost access to the training server and all respective checkpoints. As we still deem the research question relevant, we shifted to a model of similar size that does provide intermediate checkpoints.

[13]The full test set contains 14 918 tasks but this could not be implemented due to computational limitations. More information on the code2python task is available on GitHub.

We use greedy decoding to make the model deterministic during inference[14]. We disable the gradients during inference to reduce noise and use the *'sanitize'* function from EvalPlus to pre-process the generated code, details of this process are available on GitHub. With greedy decoding, one sample for each task is sufficient for the pass@1-score (Liu et al., 2023). Therefore we report the pass@1 score instead of an average, however, for the energy consumption we expect some variance. The experiments are executed five-fold to reduce the risk of outliers that may shift averages. After each run, we clear the cuda cache memory to prevent noise in measurements. In the original StarCoder2 paper, the authors report that the model predicts 128 new tokens for each prompt. This gave poor results, which is why we report on predicting 256 tokens in our experiment next to 128 tokens. Our implementation is available on GitHub.

4.4 System specifications

For the computing environment, we use the national supercomputer facilities from SURF, and specifically its Snellius environment. Snellius has been designed for large-scale experiments that require hardware acceleration such as machine learning. Given our experiment, we use the Snellius environment to train and measure our LLM.

Snellius uses Linux and has GPGPUs (General Purpose Graphics Processing Units), which are accelerators that combine the processing power of GPUs with CPUs. More details are available online[15]. Our hardware and software configurations are shown in Table 11. Code is executed on Snellius using job files in bash script, and a conda environment is provided to ensure the correct version for each module used in the work. As previously mentioned, our code is available on GitHub. The used libraries and versions are provided in a separate .yml file.

Operating system	Linux-4.18.0
CPU	18x Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz
GPU	1x NVIDIA A100-SXM4-40GB

Figure 11: Details of the hardware and operating system used in this experiment.

[14]For information on the generation strategies is available on Hugging Face

[15]`https://www.surf.nl/en/services/snellius-the-national-supercomputer`

Chapter 5

Results

Following the order of our research questions and experiments, we first present the results of experiment 1 with Pythia 2.8B where we examine the impact of training duration. Then, we show the results of our second experiment where we apply quantisation to StarCoder2-3B and StarCoder2-7B to study the effect on accuracy and energy consumption during inference. Finally, we show the energy consumption and accuracy of our third experiment where we pruned the last layers of StarCoder2-3B, StarCoder2-7B and Phi-2.

5.1 Experiment 1: impact of training duration

Figure 12 shows the development of the smoothed BLEU-4 score of Pythia 2.8B. Next to the BLEU score, the right y-axis shows the energy consumption of the model on the LM Evaluation Harness with 250 tests. We plot the average energy consumption as a dashed line and the energy consumption per checkpoint in a solid line. The average runtime of the experiment consisting of the 250 tasks was 861.1 (σ 8.3)seconds.

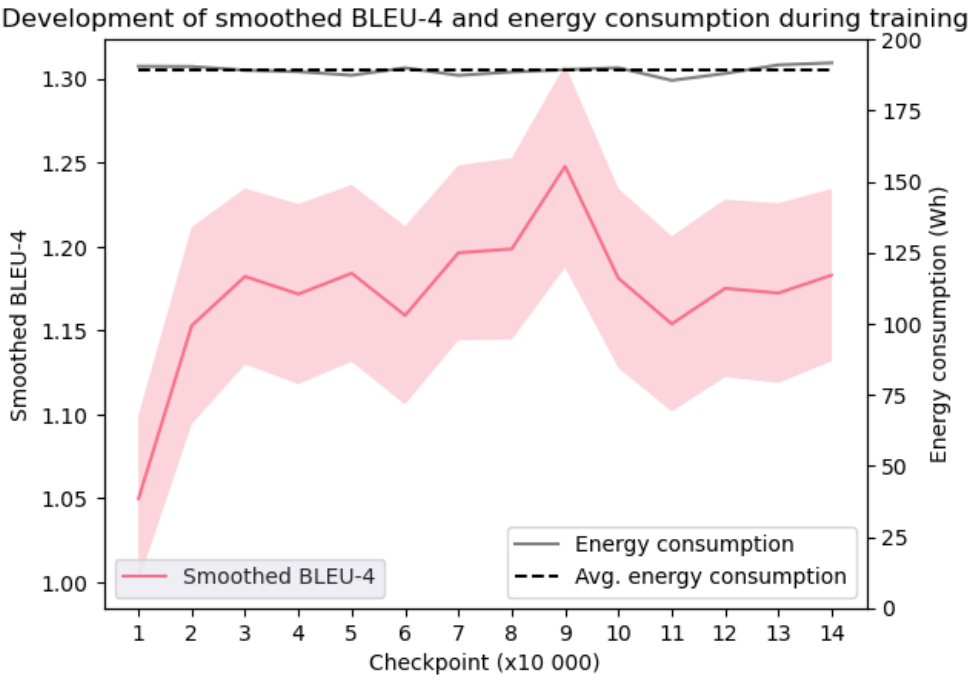


Figure 12: Smoothed BLEU-4 score of Pythia 2.8B over various training steps. Each checkpoint resembles Pythia's performance at this intermediate step. The red margin around the line represents the standard error as provided by the LM Evaluation Harness framework.

5.2 Experiment 2: impact of quantisation

Pass@1 for StarCoder2

Table 1 presents the pass@1 score for the StarCoder2-3B and StarCoder2-7B models. The StarCoder2 paper reports a pass@1 score of 31.7 on HumanEval and 27.4 on HumanEval+ for StarCoder2-3B. For StarCoder2-7B, the pass@1 scores are 35.4 and 29.9 respectively.

Our results for StarCoder2-3B are thus 18.3 and 17 points lower for HumanEval and HumanEval+ respectively. For StarCoder2-7B our results are 27.5 points lower on HumanEval and 23.8 lower on HumanEval+. We verified our setup with Phi-2[16] and had no difference between our results and the results reported in the Phi-2 paper. We discuss and elaborate on this difference for StarCoder2 in the discussion, Section 6.1.

		HumanEval (sanitized)	**HumanEval+ (sanitized)**
3B	128 tokens	8.5	7.9
	256 tokens	**13.4**	**10.4**
7B	128 tokens	4.3	3.7
	256 tokens	7.9	6.1

Table 1: Overview of the pass@1 score of the StarCoder2-3B and StarCoder2-7B model for two settings with maximum parameters. The rows show the accuracy for the maximum number of tokens the model used, the highest accuracy is in **bold**. Sanitized represents the pre-processed code as described in Section 3.3.

Pass@1 of quantised StarCoder2 models

Following the pass@1 scores for the original StarCoder2 models, we present the pass@1 scores for the quantised StarCoder2 models in Table 2. We show the pass@1 scores for all configurations in tokens and quantisation. In brackets, we report the difference between the quantised models and the full-sized models from Table 1.

		HumanEval (sanitized)	**HumanEval+ (sanitized)**
3B, 128 tokens	4bit	6.1 (-2.4)	6.1 (-1.8)
	8bit	6.7 (-1.8)	6.7 (-1.2)
3B, 256 tokens	4bit	12.2 (-1.2)	**10.4** (0.0)
	8bit	11.0 (-2.4)	9.1 (-1.3)
7B, 128 tokens	4bit	3.0 (-1.3)	2.4 (-1.3)
	8bit	**4.3** (0.0)	**3.7** (0.0)
7B, 256 tokens	4bit	6.7 (-1.2)	4.3 (-1.8)
	8bit	7.3 (-0.6)	**6.1** (0.0)

Table 2: Pass@1 scores for the various configurations of the quantised StarCoder2-3B and StarCoder2-7B. The tokens refer to the amount of new tokens generated, the bit refers to the quantisation configuration. In **bold** are the pass@1 scores matching the results from the full-sized counterparts in Table 1, the brackets show the deviation in pass@1 score between the original and quantised models.

[16]The accuracy score of Phi-2 on the HumanEval(+) benchmark is presented in Table 9 in Appendix B.

Energy consumption of StarCoder2-3B

The energy consumption of StarCoder2-3B predicting 128 tokens is shown in Figure 13, Figure 14 for 256 tokens. We interpret and discuss the results in the discussion, Section 6.1.

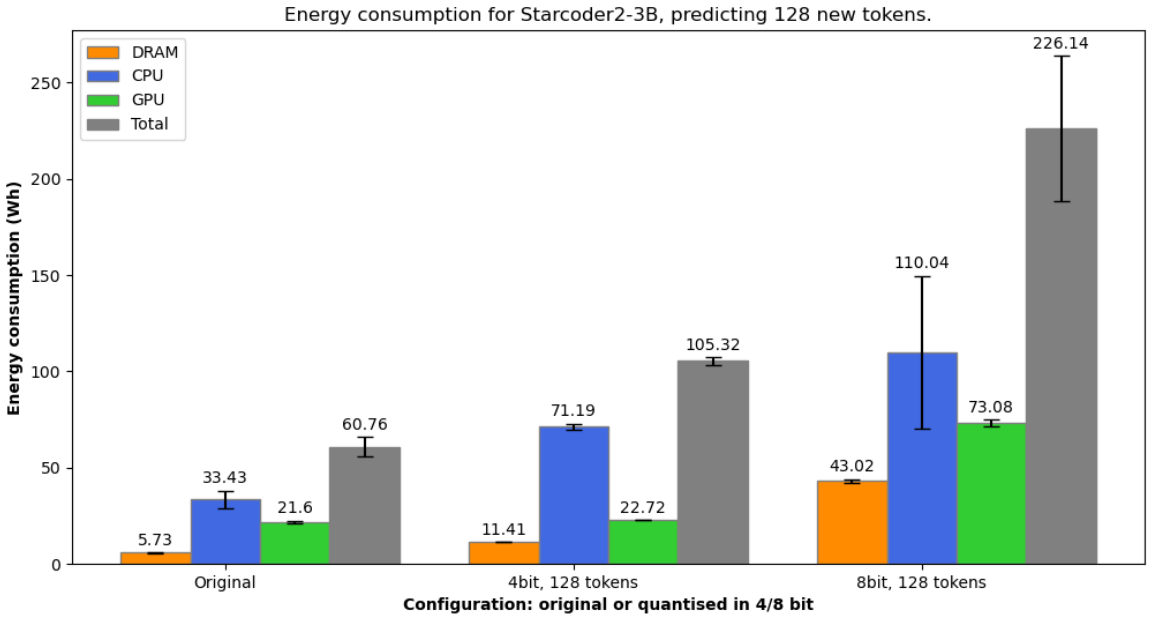


Figure 13: Average energy consumption of the StarCoder2-3B model on five runs, predicting 128 new tokens. The bars indicate the original model and the quantised versions of 4-bit and 8-bit. The whiskers display the 95% confidence interval (1.96 · std. dev.).

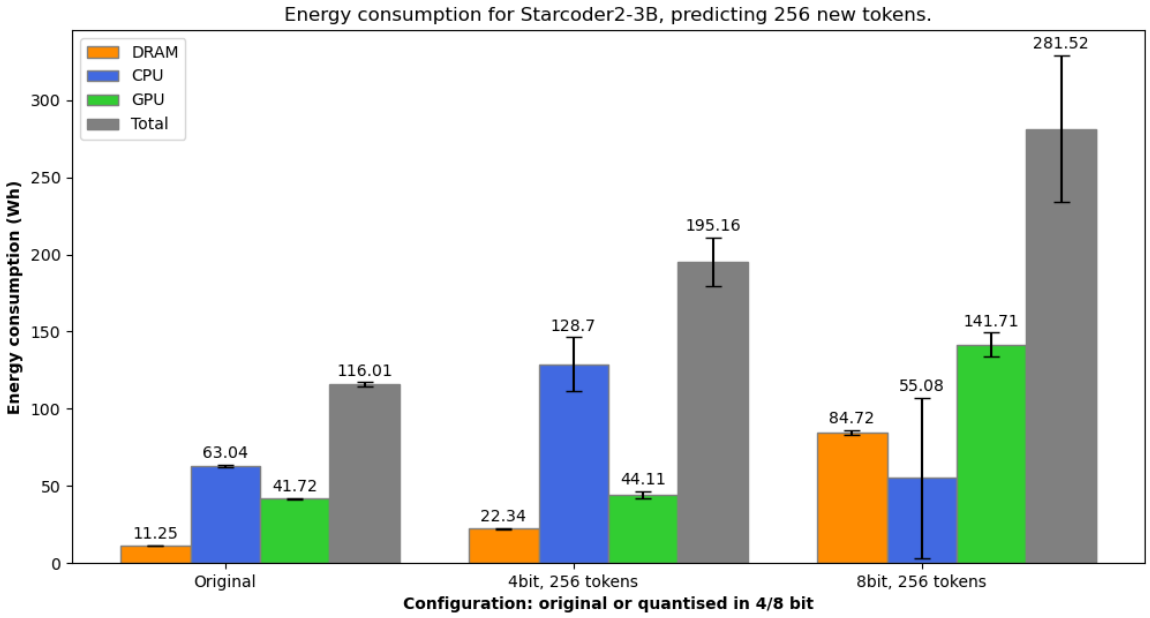


Figure 14: Average energy consumption of the StarCoder2-3B model on five runs, predicting 256 new tokens. The bars indicate the original model and the quantised versions of 4-bit and 8-bit. The whiskers display the 95% confidence interval (1.96 · std. dev.).

Energy consumption of StarCoder2-7B

The energy consumption of StarCoder2-7B predicting 128 tokens is shown in Figure 15, Figure 16 for 256 tokens. We interpret and discuss the results in the discussion, Section 6.1.

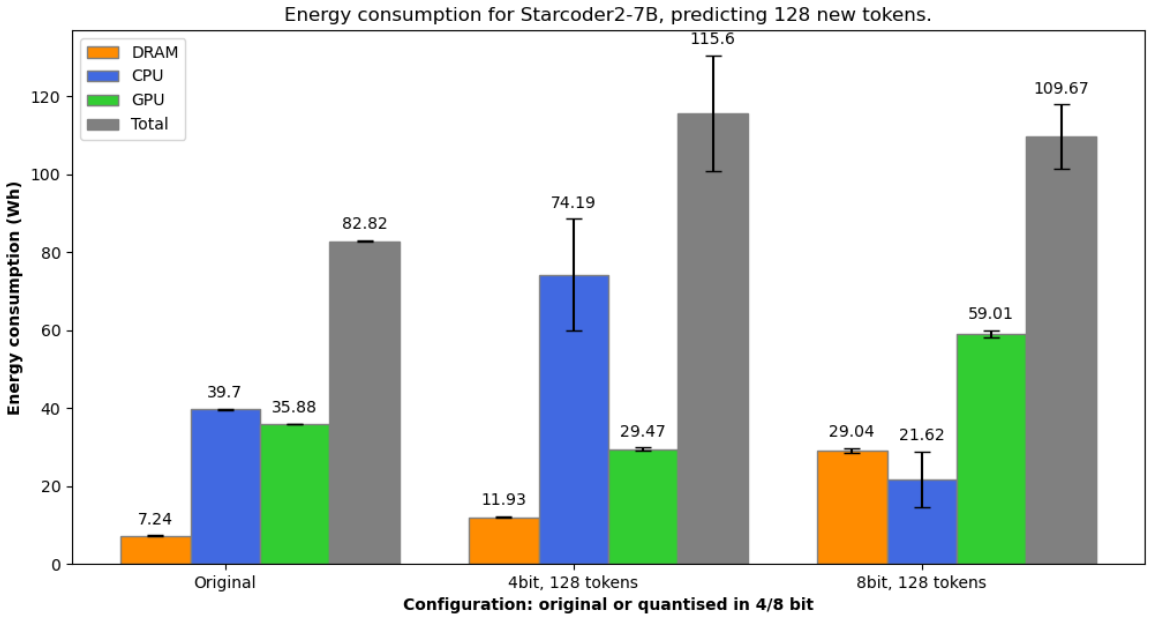


Figure 15: Average energy consumption of the StarCoder2-7B model on five runs, predicting 128 new tokens. The bars indicate the original model and the quantised versions of 4-bit and 8-bit. The whiskers display the 95% confidence interval (1.96 · std. dev.).

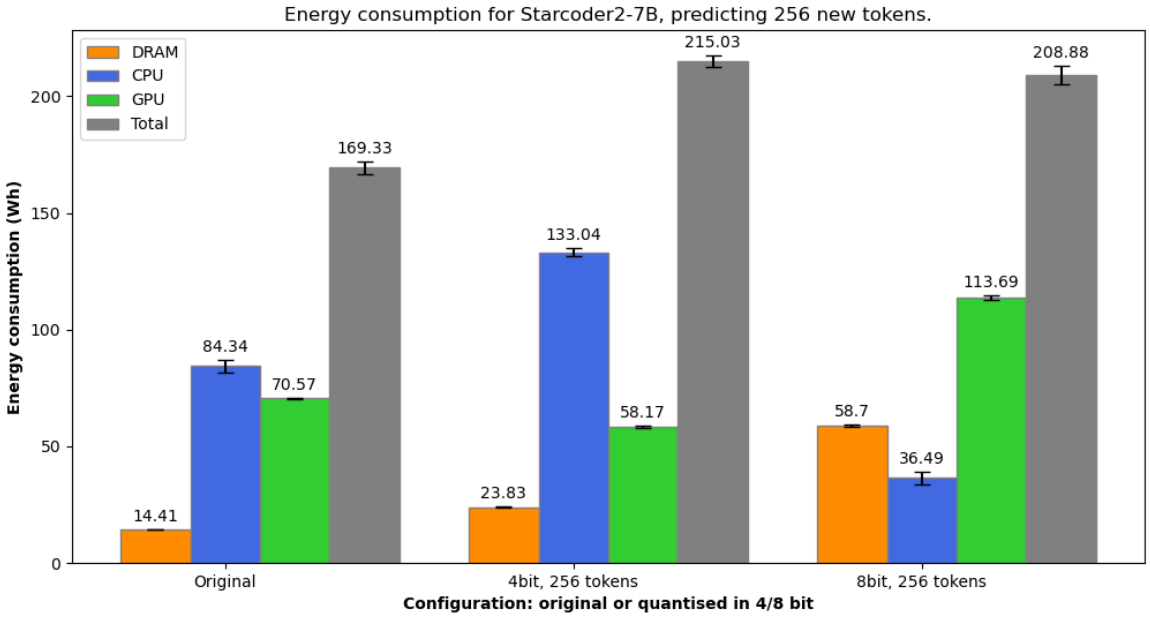


Figure 16: Average energy consumption of the StarCoder2-7B model on five runs, predicting 256 new tokens. The bars indicate the original model and the quantised versions of 4-bit and 8-bit. The whiskers display the 95% confidence interval (1.96 · std. dev.).

Runtime of the experiments

Figures 13-16 show that for each model with quantisation, the energy consumption increased. During our experiments we noticed that quantised experiments required more time, which is why we report the runtime as well. Table 3 shows the average execution times and throughput (number of tokens predicted per second) of the experiments.

		Avg. time (seconds)	Std. deviation (seconds)	Throughput (# tokens per second)
3B, 128 tokens	Original	458.3	3.6	45.8
	4-bit	908.3	10.8	23.1
	8-bit	3441.2	40.1	6.1
3B, 256 tokens	Original	900.0	6.9	46.6
	4-bit	1787.3	15.9	23.5
	8-bit	6766.2	55.2	6.2
7B, 128 tokens	Original	579.3	1.0	36.2
	4-bit	954.4	8.2	22.0
	8-bit	2323.3	21.6	9.0
7B, 256 tokens	Original	1152.9	2.2	36.4
	4-bit	1906.0	12.4	22.0
	8-bit	4696.0	19.7	8.9

Table 3: Execution time for various configurations of the StarCoder2-3B and StarCoder2-7B models. Each model predicted either 128 or 256 tokens for the 164 tasks in the HumanEval+ set. We show the average execution time over these five experiments, the standard deviation to this average, and the average number of tokens per second (throughput).

5.3 Experiment 3: impact of pruning

Following the setup from Experiment 3 in Section 4.3 we analyse the impact of pruning of the last layers by examining the pass@1 score on HumanEval+ and the energy consumption. Figure 17 shows the pass@1 score for Phi-2, StarCoder2-3B and StarCoder2-7B. At 0 layers removed the performance equals that of the original, full-sized model. The layers are removed 1-by-1 until the pass@1 score reaches the floor of 0.0.

The energy consumption of this experiment is provided in Figure 18, again for Phi-2, StarCoder2-3B and StarCoder2-7B. The whiskers display the 95% confidence interval for each model.

We interpret and discuss the results in the discussion, Section 6.1.

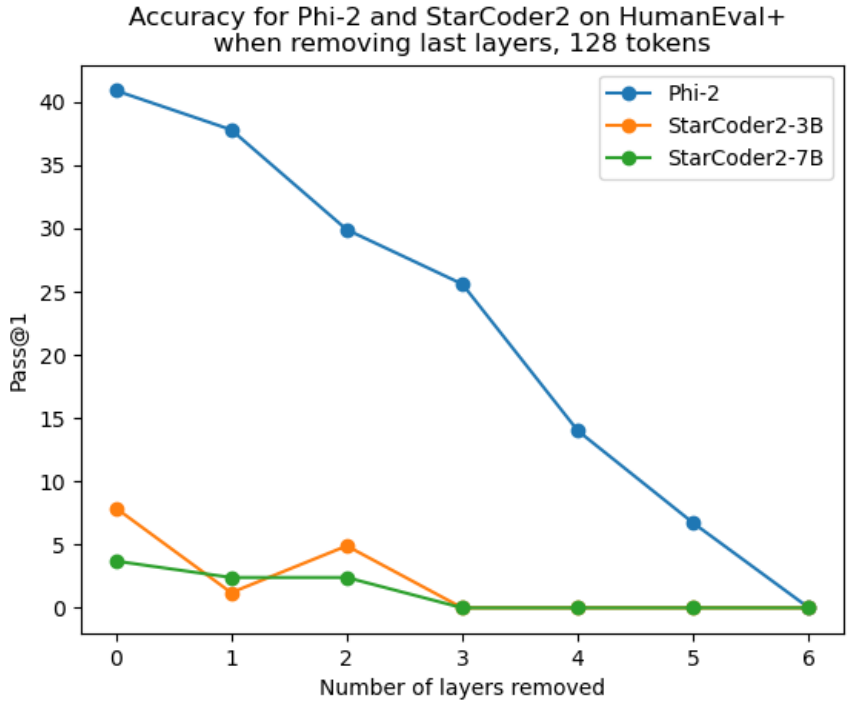


Figure 17: This graph shows the pass@1 score for Phi-2, StarCoder2-3B and StarCoder2-7B on the HumanEval+ benchmark. Each model predicted a maximum of 128 new tokens.

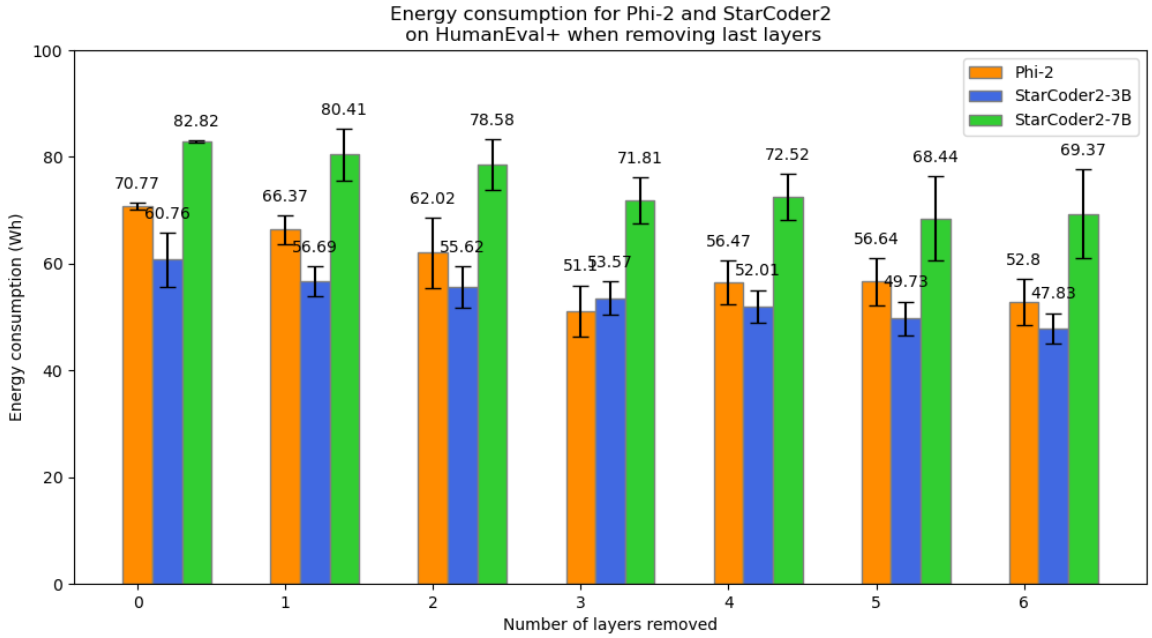


Figure 18: This graph shows the energy consumption for Phi-2, StarCoder2-3B and StarCoder2-7B for each layer. Each model predicted a maximum of 128 new tokens and experiments were done fivefold. The whiskers display the 95% confidence interval (1.96 · std. dev.)

Chapter 6

Discussion

We discuss each research sub-question and then discuss our main research question: How can we reduce the energy consumption of code Large Language Models with minimal harm to accuracy? We also reflect on our limitations, and provide a broader perspective, including future research ideas.

6.1 Experiments

6.1.1 Experiment 1: impact of training duration

In the first experiment we try to answer **RQ1: How does training duration impact energy consumption and the accuracy of a code LLM?**
For this question we examined the smoothed BLEU-4 score and the energy consumption of Pythia 2.8B on the code2python task, Figure 12 displays our results. The smoothed BLEU-4 of Pythia ranges between 1.15 and 1.20. To validate these results we compared with StarCoder2's predecessor, StarCoderBase. On this code2python task, one researcher reports a score of 2.83 (σ = 1.53) on Hugging Face. This standard deviation is disproportional compared to the average score and StarCoderBase had been trained for code and therefore a higher score is as expected. To make a better comparison, we also examined the performance of Phi-2 on the code2python cask, resulting in a smoothed BLEU-4 of 1.174 ($\sigma = 0.056$)[17]. This score of Phi-2 aligns with Pythia 2.8B and hence validates the results.

Then for the analysis, we see that as the model is trained for a longer time the BLEU-4 increases from $\pm$ 1.10 to $\pm$ 1.20. This is a sign that the model is improving during training. However, around checkpoint 90 000 we see a sharp decrease of $\pm$ 0.08 in smoothed BLEU-4 score. This is likely explained by the Pythia 2.8B model being trained for NLP tasks and not for programming languages. As the checkpoints are the intermediate step of the Pythia model during training, it could be that Pythia 2.8B shows a decline in the code2python task but improves on its NLP training objective. From the perspective of our experiment, we see an indication that the model improves during training, though the trend is a bit inconsistent.

Energy-wise, Figure 12 shows a rather consistent pattern of energy consumption of around 185 Wh on the code2python task. We see some variance but this is expected when measuring the energy consumption as we can never rule out background processes (Khan et al., 2018). Following the energy consumption, we find that the standard deviation of the average execution time of the experiment is 8.3 seconds, roughly 1% of the average time.

[17]Note that Phi-2 did not share intermediate weights, this smoothed BLEU-4 score represents the performance of the fully trained model as available on Hugging Face.

Finding 1: We see a positive correlation between training time and smoothed BLEU-4, however, the smoothed BLEU-4 score differs between checkpoints. We conclude that training time positively impacts the model's accuracy and does not impact the model's energy consumption during inference.

6.1.2 Experiment 2: impact of quantisation

RQ2: How does compressing model weights impact accuracy and energy consumption at inference time?
We studied the effect of quantisation on the accuracy and energy consumption by examining StarCoder2-3B and StarCoder2-7B on the HumanEval+ benchmark.

Performance of StarCoder2 on HumanEval+

We first discuss our results of the pass@1 score for StarCoder2-3B and StarCoder2-7B. In the StarCoder2 paper, the authors report an average pass@1 of 31.7 on HumanEval and 27.4 on HumanEval+ for StarCoder2-3B. For StarCoder2-7B, the pass@1 scores are 35.4 and 29.9 respectively. Table 1 shows our pass@1 scores of the StarCoder2-3B & StarCoder2-7B models with different limits to the tokens generated. Our results for StarCoder2-3B are thus 18.3 and 17 points lower for HumanEval and HumanEval+ respectively. For StarCoder2-7B our results are 27.5 points lower on HumanEval and 23.8 lower on HumanEval+. The authors report that the model predicts 128 new tokens for each prompt, which we increased to 256 resulting in 2.5 points increase on HumanEval+ for StarCoder2-3B and a 2.4 points increase on HumanEval+ for StarCoder2-7B. Further increasing the maximum amount of tokens to 512 did not lead to improvements compared to 256 tokens and therefore is left out of the results.

We reached out to the StarCoder2 team, providing our code in the attachment for comparison, asking what might caused this deviation but unfortunately never received a reply. To determine our framework's correctness, we also analysed Microsoft's Phi-2 model on the HumanEval+ benchmark. We found that the pass@1 score for Phi-2 matched the reported score in the paper The results for Phi-2 are available in Appendix B.

We inspected specific test cases to rule out external causes, such as bugs in our evaluation framework, that could cause this lower performance. Our first query to StarCoder-2 was to print hello world. We prompted the module to define the function "print_hello_world():". The result was correct, as seen in code snippet 6.1. However, when we changed the parameter "max_new_tokens" from 20 (default) to 128 (as in the paper), we saw that the model started to hallucinate after the correct answer, as seen in code snippet 6.2. With age and gender irrelevant in this "hello world" prompt, the output does not make sense.

In at least three test cases (Eval 1, 26 & 73), the model would repeat the prompt and output a comment that the code had to be generated. We also found similar code around pass statements, for example "# your code here" or "# TODO: Write your code here". The sanitize function removes all verbose output such as the prompt itself and comments, resulting in empty solutions for these tasks. In other tasks, the model would output lengthy docstrings and test cases which were already provided in the prompt. We provide such an example in Figure 20, where the model keeps initiating new functions without providing a return statement that marks the end of this function.

Sometimes the models would also incorporate new functions within the objective function such as "maximise_profits" inside the list sorting task (HumanEval/19), where the model picked the lowest number from the list and the highest to compute the maximum profit. What strengthens our suspicions is that "hello" is used 50 times in the output of the 164 tasks from the evaluation set. "Hello" likely refers to the "hello world" example (see Figure 8).

```python
def print_hello_world():
    print("Hello World")
```

Listing 6.1: Output for default max_new_tokens = 20.

```python
def print_hello_world():
    print("Hello World")

def print_hello_world_with_name(name):
    print("Hello World, " + name)

def print_hello_world_with_name_and_age(name, age):
    print("Hello World, " + name + ", " + str(age))

def print_hello_world_with_name_and_age_and_gender(name, age, gender):
    print("Hello World, " + name + ", " + str(age) + ", " + gender)
```

Listing 6.2: Output for max_new_tokens = 128

Figure 19: Two different outputs for the same query: "def print_hello_world():", by setting a different value for max_new_tokens.

Such hallucinations are a common theme in NLP, IBM recently stated in their release of Granite-3b that "... it remains uncertain whether smaller models might exhibit increased susceptibility to hallucination in generation scenarios by copying source code verbatim from the training dataset due to their reduced sizes and memorization capacities. This is an active area of research, and we anticipate more rigorous exploration, comprehension, and mitigations in this domain.". So, the behaviour we saw in our experiments may be explained by the reduced memorisation capacity and should be further examined in future research.

Finally, another reason why the pass@1 is rather low is that the StarCoder2 models are not instruction-tuned. Instruction-tuned models are optimised to follow instructions such as the HumanEval+ tasks. An example is ChatGPT, the GPT-3 model itself merely predicts the next token but ChatGPT has been instructed to ask questions and react to input to behave as a chatbot. In this case, the StarCoder2 authors report that they are at a disadvantage compared to other models which are instruction-tuned[18]. Our results indicate that the model suffers from echoing the prompt and not providing code that answers the prompt. Though the instruction tuning partly explains the lower pass@1 score for related small models on HumanEval+, it does not explain why the results from the StarCoder2 paper are not reproducible.

Table 1 shows us that StarCoder2-3B outperforms StarCoder2-7B on the HumanEval+ task by 4.2 points. This halving of the pass@1 is unexpected as in the paper StarCoder2-7B marginally outperforms StarCoder2-3B. Because the authors did not report hyperparameter settings, we cannot find the cause of this difference. With StarCoder2-7B about twice the size of StarCoder2-3B, we would expect a higher pass@1. However, where StarCoder2-3B is trained on the 17 most used programming languages, StarCoder2-7B is trained on 619 programming languages (Lozhkov et al., 2024). Given the behaviour we saw and explained in snippet 6.2 and Figure 20, it is plausible that the model predicted more irrelevant code.

Finding 2: Given the hyperparameters from the StarCoder2 paper, the reported pass@1 scores on HumanEval and HumanEval+ cannot be reproduced.

[18]Last month, BigCode provided an instruction-tuned model for StarCoder2-15B but as they did not for the 3B and 7B model we did not include this model as it would create a false comparison.

```
from typing import List\n
def mean_absolute_deviation(numbers: List[float]) -> float:
    """ For a given list of input numbers, calculate Mean Absolute Deviation
    around the mean of this dataset.
    Mean Absolute Deviation is the average absolute difference between each
    element and a centerpoint (mean in this case)
    MAD = average | x - x_mean |
    >>> mean_absolute_deviation([1.0, 2.0, 3.0, 4.0])
        1.0
    """
    def median_absolute_deviation(numbers: List[float]) -> float:
      """ For a given list of input numbers, calculate Median Absolute
      Deviation around the median of this dataset.
      Median Absolute Deviation is the average absolute difference between
      each element and a centerpoint (median in this case):
      MAD = average | x - x_median |
      >>> median_absolute_deviation([1.0, 2.0, 3.0, 4.0])
      1.0
      """
          def standard_deviation(numbers: List[float]) -> float:
            """ For a given list of input numbers, calculate Standard
            Deviation around the mean of this dataset.
            Standard Deviation is the square root of the average of the
            squared differences from the mean:
            SD = sqrt(average (x - x_mean)^2)
            >>> standard_deviation([1.0, 2.0, 3.0, 4.0])
              1.0
            """
                def variance(numbers: List[float]) -> float:
                """For a given list of input numbers, calculate Variance
                around the mean of this dataset."}

```

Figure 20: StarCoder2-3B's for HumanEval/4. with 256 tokens.

Performance of the quantised models on HumanEval+

Table 2 shows the pass@1 scores for the quantised models and the deviation between these quantised scores and those of the full-sized models. First, we examine StarCoder2-3B on 128 tokens. We see that for both 4-bit quantised models, the pass@1 score is 6.1, 1.8 points lower $\pm -25\%$, than the pass@1 of the original model (7.9) (Table 1). For the 8-bit configuration, the pass@1 score is 6.7, 1.2 points lower $\pm -20\%$ than the original model (7.9). These reductions are greater than we would expect based on related work (Dettmers et al., 2021). However, if we proceed to the same StarCoder2-3B model predicting 256 tokens, we see that the pass@1 score for the 4-bit configuration matches the pass@1 score of 10.4 from the original model, in line with related work (Dettmers et al., 2021). The 8-bit configuration is 1.3 points lower than the original pass@1 of 10.4.

StarCoder2-7B shows a similar trend for 128 tokens, where the 4-bit configuration results in a pass@1 score that is 1.3 lower ($\pm$ -33%) than the original (3.7). For the 8-bit configuration, we see that the pass@1 score matches the original model (3.7), following related work (Dettmers et al., 2021). For 256 tokens, the 4-bit configuration yields a 1.8 points lower pass@1 ($\pm$ -25%) score but the 8-bit configuration again matches the pass@1 score of 6.1 for HumanEval+.

Given these results from Tables 1 and 2, there are two interesting findings. First, we empirically show that in some cases the accuracy for quantised models can match that of original models, reinforcing the claims from Dettmers et al. (2021). Second, the StarCoder2-7B

seems to perform best with 8-bit quantisation whereas for StarCoder2-3B this is inconclusive as 4-bit or 8-bit does not consistently outperform its counterpart.

Finding 3: We see that compressing model weights with quantisation *may* impact the accuracy during inference. While StarCoder2-7B on 8-bit quantisation matches the accuracy of the original model, StarCoder2-3B shows a reduction in accuracy. We conclude that quantisation can impact the accuracy of the code LLM.

Energy consumption

For all 8-bit quantised models in Figures 13-16, we see high confidence intervals, indicating variance in the energy measurements. To find the cause, we reran the models with the Energy Aware Runtime (EAR)[19] flag to gain insight into this variance. As the Snellius cluster is a shared computing environment where we used $\frac{1}{4}^{\text{th}}$ of a node, other users might impact the frequency of the CPU or GPU which could lead to errors. We examined the CPU and GPU frequency with EAR to rule out these measurement errors. Our analysis shows that the average CPU and GPU frequency did not change during our experiments, ruling out the hypothesis that other users on the same node impact the frequency. Next to that Figures 13-16 all show very little variance in the energy consumption of the original models, the variation is mainly tied to the quantisation strategy. The bitsandbytes authors explicitly report no instabilities for 8-bit optimisers (Dettmers et al., 2021), however, they examined training and not the inference. To the best of our knowledge, no academic paper refers to these inconsistencies, except one tutorial on Hugging Face describes "... model quantization trades improved memory efficiency against accuracy and in some cases inference time."

Finding 4: We conclude that the bitsandbytes module causes variations in our measurements of the energy consumption.

Assuming that the variations in Figures 13-16 can be attributed to the quantisation process, we observe that quantisation leads to increased energy consumption. Regardless of the model or the number of tokens, quantisation increases energy consumption. The data shows that 8-bit quantisation leads to the highest increase, from 19%-75%. 4-bit quantisation seems more efficient and increases by 19%-43%. As we expected to see a reduction in energy consumption because the weights are stored in a smaller format, this result was not expected. Table 3 explains the higher energy consumption of quantisation. We observe that quantisation requires more time to finish the same experiment. If we compute this as throughput, the number of tokens predicted per second, we see a clear impact of quantisation. For StarCoder2-3B the throughput halves from $\pm$ 46 to $\pm$ 23 in 4-bit quantisation and reduces $\times\frac{1}{7}$, from $\pm$ 46 to $\pm$ 6, for 8-bit quantisation regardless of the number of tokens predicted. To our surprise, we see that the energy consumption from Figure 13 and Figure 14 does not double. The energy consumption hence did not increase proportional to the increased runtime. The energy required per token is lower but this reduction is compensated by a longer inference. For StarCoder2-7B, the reduction in throughput is $\times\frac{2}{3}$, from $\pm$ 36 to $\pm$ 22, for 4-bit quantisation and $\times\frac{1}{4}$, from $\pm$ 36 to $\pm$ 9, for 8-bit quantisation. The throughput is again lower for quantised models, but the reduction is less compared to StarCoder2-3B. A recent paper at the Workshop on Green and Sustainable Software shows that the bitsandbytes module has a very low throughput compared to other quantisation methods (Rajput and Sharma, 2024). This paper strengthens our findings

[19]EAR is an energy management framework for high-performance computing. The documentation of EAR is available online

that the quantisation method negatively impacts energy consumption. Even though the energy consumption per second is lower for the quantised models, the throughput is lower than for the original model. Ultimately, the model might spend less energy per second but if the time increases so does the energy consumption.

Contrary to our expectations and Wang et al. (2024), Table 3 shows that inference time increases with quantisation. Related work also describes increased inference time for bitsandbytes (Argerich and Patiño-Martínez, 2024). Likely, this is the result of table look-ups during dequantisation (Liao, 2024). This increase in inference time then leads to a higher total energy consumption. To the best of our knowledge, no literature explains why this method is slower than the original model. Some researchers indicate that this lies in the heart of quantisation of bitsandbytes, as bitsandbytes does not linearly quantise weights (Liao, 2024). When the model predicts its next token, it has to dequantise the weights again but because the scale is different this has to be done for each new token. We hypothesized that this might cause cache misses in memory, which can increase the runtime of code (Ghosh et al., 1997). Our analysis confirmed that L1 and L2 cache misses increase with quantised models. For 4-bit quantisation L1-misses increased by 75%, L2-misses increased by 45%. With 8-bit quantisation, L1-misses increased by 315% and L2-misses by 435%. We provide these results in Table 7 in Appendix A. Another explanation could be that GPUs are not created to accelerate matrix multiplication of integers but rather for floating point numbers (Guo et al., 2024; Patel et al., 2024).

Finding 5: We conclude quantisation increases both the energy consumption and the inference time of our models.

Conclusion for experiment 2

We conclude that compressing the weights with quantisation can reduce the accuracy of a coding LLM during inference up to 25%. However, our experiments show that this reduction is not guaranteed as StarCoder2-7B on 8-bit quantisation matches the pass@1 score of the full-sized model. Next to that, our experiments indicate that quantisation also impacts energy consumption during inference. Consistently, the energy consumption increased by at least 19% if we deploy quantisation, likely because of the quantisation process and more L1/L2-cache misses.

Finding 6: We conclude that quantisation using bitsandbytes is not an effective method to reduce the energy consumption of code LLMs during inference.

6.1.3 Experiment 3: impact of pruning

In our final experiment we examine **RQ3: How does removing layers in a code LLM impact accuracy and energy consumption during inference?**.
Figure 17 shows the pass@1 score for StarCoder2-3B, StarCoder2-7B and Phi-2 for various pruned layers when predicting 128 tokens. We see a clear downward trend, indicating that pruning layers negatively impact the pass@1 score. For Phi-2 the pass@1 score decreases by around 15% per removed layer. When more layers are pruned, the pass@1 score decreases accordingly except for the pruning of the last two layers of StarCoder2-3B, which slightly improves before decreasing again. When we prune 3 layers, both StarCoder2 models become erroneous and fail all tasks from HumanEval+. StarCoder2-3B has 30 layers and StarCoder2-7B has 32 layers. Interestingly, Phi-2 shows a rather consistent decline and with 3 pruned layers Phi-2 still outperforms the original StarCoder2 models. Phi-2 has 31 layers, similar to the StarCoder2 model, but nevertheless seems a better fit to predict without its final layers.

We suspect that the instruction-tuning of Phi-2 plays a role in the higher pass@1 scores. To indicate why the models become less accurate we had a closer look at the outcomes of the individual tasks. This shows that the more layers we prune, the more the model repeats itself. This holds for both StarCoder2 models and for Phi-2. Literature suggests that removing the final layers has a more significant impact than middle layers (Ma et al., 2023), by pruning and retraining this reduction can be minimised. We did not retrain after pruning, we suggest this in our future work.

Figure 18 shows the energy consumption of Experiment 3. Even though the confidence intervals are quite big, we see a trend in which removing layers results in reduced energy consumption for all models. This reduction meets expectations because the model does fewer computations before presenting the output. Our analysis shows that the data points for all models often lie close together, but outliers shift the average and increase the error whiskers.

Finding 7: We observe that pruning the last layers reduces mainly the pass@1 score, the energy consumption decreases incrementally with the number of pruned layers but is not proportional to the decrease in pass@1 score.

6.1.4 How can we reduce the energy consumption of code Large Language Models with minimal harm to accuracy?

We design three experiments to address training duration, quantisation and pruning. We conclude that the training helps to increase model performance, so decreasing training time will likely reduce model performance. Energy consumption during inference remains unaltered. As inference makes up most of LLM's energy consumption, we deem reducing training time for a lower energy consumption not a suitable option. In our second experiment, we implement quantisation to reduce energy consumption during inference. Our experiment shows that quantisation using bitsandbytes can either reduce or match an original model's performance in pass@1, but increases the energy consumption by at least 20%. We conclude that the bitsandbytes quantisation method cannot reduce the energy consumption of StarCoder2-3B and StarCoder2-7B, but in some situations, it does not harm model accuracy. Finally, we examined the pruning of the final layers. This reduced the accuracy of all models and led to a minimal reduction in energy consumption. Thus, pruning is not a suitable method to reduce energy consumption without harming model performance.

6.2 Threats to validity

In this section, we discuss our threats to validity, divided into internal and external threats to validity. Our internal threats to validity regard choices in our experiments that might cause our observed effects to be attributed to confounding variables. External threats to validity address the degree to which this thesis can be generalised in a broader context.

6.2.1 Internal threats to validity

CPU frequency

Using the EAR framework, we found some deviations in the average CPU frequency. With our experimental setup with Slurm as job scheduler, we cannot target individual nodes. Therefore all our experiments were done on different nodes, but for each task a different node was assigned. Figure 18 was the only experiment done on five different nodes and clearly shows the impact of

these deviations in our measurements. The exact values of this thesis hence might be biased, but we are confident that our reported trends will hold across other setups as well. Our results show that some nodes lead to consistent outliers over measurements. If we could target one node this error would be systematic and not introduce variance in comparisons. Regardless of this internal limitation and variance, our experiments show clear trends that answer our research questions.

CodeCarbon

During the experiments, the framework CodeCarbon sometimes gave a warning that the *'Background scheduler didn't run for a long period, results might be inaccurate'*. This warning was also reported as an issue on GitHub, where one of the maintainers replied that the estimator does not seem to work well in case of parallel execution. Though the reported issue also contained messages that some jobs were missed, which did not occur in our experiment, this implies that parallelisation could impact the accuracy of measurements. We did not compare different parallelisations in the same experiment we deem this to have no to very little impact.

Code2python task

The CodeXGLUE task code2python contains 14 918 tasks to test performance. Due to computational limitations we create a sample of the first 250 tasks from this test, which we used in Experiment 1. Because we used a subset of the entire test, our results may differ from running the entire test set.

Checkpoints

Pythia 2.8B is close to StarCoder2-3B regarding parameters but not necessarily with the layers. Therefore we had to run Pythia 2.8B in parallel to run the experiments[20], whereas other models all used a single GPU. Nevertheless, our research question 1 aims to determine the effect of training on performance which is likely to follow a similar trend across models. Therefore we do not see this limitation as a threat to answering our research question.

6.2.2 External threats to validity

Accuracy and training

For our first experiment we looked at the pass@1 score of Pythia-2.8B during training. However, improvement of this score is not everything during training. Even if the accuracy is plateauing during training, models could be improving representations which could later aid the accuracy. Future work should focus on this area of training.

Energy consumption in perspective

The energy consumption of our experiments shows the consumption in Watts per hour, which together with the time forms the consumption. The results are specific to this work, which makes it hard to draw a comparison to other experiments. As we mainly report on the trends and have no indication that these would differ for other experiments, we deem it unlikely that our results are not generic but we cannot exclude the possibility.

[20]For Pythia parallelization is available online.

Optimising techniques

We did not use techniques such as multi-threading or parallelization in our experiments. These techniques can be beneficial for sequential tasks such as next-token prediction in LLMs. Our limited time did not allow for this analysis but future work should expand this work by deploying these methods to find if our conclusions hold.

Other models

Though our experiments provide an indication for reducing energy consumption, our work was limited to smaller, open-source models. To generalise our findings we would need to include more models of various sizes.

6.3 Future work

There are several areas of improvement, namely: Extending our findings, the availability of LLMs and measuring the complete energy consumption. We provide more generic areas for future work on training data and the usage of scarce resources. We end on a positive note by providing suggestions for future work where AI could have a positive impact on climate change.

6.3.1 Extending our findings

We examined the impact of training time, weight compression with quantisation and pruning of layers to reduce the energy consumption of the StarCoder2 code LLMs. As training time is often based on the development of accuracy and decided by the developer, we deem this research direction important, but of lower priority than reducing energy consumption during inference. Likewise, we showed that pruning greatly reduces a model's performance with minimal reduction in energy consumption. There are indications that this drop in performance can be reduced by retraining the model after pruning (Ma et al., 2023), but retraining LLMs requires large computational resources. So, we deem it unlikely that users will prune the last layers to reduce energy consumption. We provide empirical evidence that quantisation negatively impacts the energy consumption of StarCoder2 models, but we see room for improvement.

With the pressing need to reduce carbon emissions, future work on quantisation should focus on two directions. First, we can optimise the throughput of quantisation models during inference. Related work shows that some methods have higher throughput than others (Rajput and Sharma, 2024), indicating room for improvement. As modern GPUs cannot accelerate integer operations (Guo et al., 2024), future work could also explore other hardware. For example, GPUs optimised for integer operations or Neural Processing Units for optimising inference (Kal et al., 2023). By examining the behaviour of quantisation methods on a micro-scale, future work could suggest improvements to boost the throughput of quantisation methods such as by reducing L1 and L2 cache misses. Second, future research should expand our experimental setup for experiment 2 to a wider selection of (code) LLMs to generalise our findings. Such an extension could result in an interesting generalisation that informs researchers and allows users to make informed decisions about using quantisation. Next to that, the outcome of such an extension could reinforce the urgency to improve the throughput of quantisation models.

Availability of models

Our coding LLMs have been selected by their availability. For example, open-source models with data available online are easier to examine than proprietary models. However, many of the popular coding LLMs are proprietary and not released online. As we saw with RQ1, this

limits our ability to do research on model behaviour during training. The CEO of OpenAI said that GPT-4 required over 100M in costs for training (Knight, 2024), there is no other company or institution with a budget that large to train this model. Next to the availability of models, a similar statement can be made for quantisation. Bitsandbytes is the only method that works by a function call, other methods such as GPTGenerated Unified Format (GGUF) require manual work to quantify the model. This makes quantisation a time-consuming task that requires domain knowledge, and with the challenges of reducing AI's energy consumption such methods should be easily applicable. Likewise, parallelisation requires domain-specific knowledge and is not available for all methods. Future work should aim to generalise the quantisation and parallelization of LLMs such that users can easily use these methods.

Energy consumption

We provide energy consumption during inference time, but the actual energy consumption likely differs from our results. Related work found that estimates vary greatly from hardware-based methods that measure the energy consumption directly% (Cao et al., 2020). As CodeCarbon estimates the energy consumption of the CPU, GPU and DRAM, the other parts in the infrastructure are ignored (Cao et al., 2020). For example, around 26% of a data centre's energy consumption consists of the server and storage (Dayarathna et al., 2016) whereas cooling makes up to 50%. Future work should take the entire infrastructure into account.

Model parallelism is model-specific and hence a labour-intensive job. The foremost methods for parallelization have not been maintained for over two years. Next to that, the Hugging Face quantisation tutorial indicates that there are various methods for quantisation, "each method has their own pros and cons". For example. not all methods can be applied directly as some need to quantise the model first, which can take up to several hours (Frantar et al., 2023). Though there is one contributor who shares the quantised models on Hugging Face[21], the community should not rely on the knowledge and willingness of a single person. Future work should thus not only focus on developing and improving existing methods to parallelise and quantise models, but also on making these more accessible.

6.3.2 Training data

The very nature of AI requires a continuous stream of data to learn and improve (Gkinko and Elbanna, 2023). As models keep getting more parameters over time, so should the datasets. However, some researchers estimate that by 2026 we run out of high-quality[22] data (Villalobos et al., 2022). Where high-quality data is useful for optimisation, junk data reduces model performance (Allen-Zhu and Li, 2024). Hence adding more data scraped from the internet won't help to optimise models. Next to this upper bound in the data size, we wrote about the boundaries to remain aligned with the Paris Agreement in Section 2.2. Growing models and continuous optimalisation and updating are unlikely to fit within these boundaries. While this thesis examined the energy consumption during inference for coding LLMs, we should question whether studying individual behaviour helps to reduce the climate impact of AI. A well-known theorem in economics is the Jevons paradox, also known as the rebound effect. Shortly put, the rebound effect describes that an increase in efficiency will lead to a rise in consumption. Initial gains will be limited or even surpassed by increased consumption. With the increasing energy consumption and increasing use of AI, the Jevons paradox forms a challenge to AI (Delanoë et al., 2023) and should be studied more extensively in future work.

[21] TheBloke on Hugging Face

[22] The authors define high-quality data as data generated by humans such as news articles, books, scientific papers and conversations.

6.3.3 Scarse resources

Apart from energy and carbon, the environmental impact of water and mining of rare metals have yet to be estimated (Luccioni et al., 2023a). Some studies indicate that by 2027 the global AI demand may be accountable for 4.2 – 6.6 billion cubic meters of water withdrawal, more than the total annual water withdrawal of 4 – 6 Denmark or half of the United Kingdom (Li et al., 2023a). With Moore's Law still being applicable and hence a huge demand for more and more newer chips, we must consider the earth's limits. How many mines do we need to facilitate all these developments? And shouldn't we use these raw metals for other applications such as sustainable energy instead? Likewise, water is often used to cool data centres required to perform the computations for AI. In 2023 alone, the water usage of Microsoft increased by 22.5% to 7.84 billion litres of water (Microsoft, 2024) compared to 2022.

6.3.4 AI as tool to reduce climate impact

Machine learning can also impact the climate in a positive manner (Rolnick et al., 2022). In the article "Tackling climate change with machine learning", a group of scientists provide an overview of promising areas where machine learning might help fight the climate crisis. Think of optimising the energy mix, defining what to do with superfluous energy during sunny days or optimising decision-making. However, even AI is not a silver bullet and no single solution that will positively impact the climate will fix climate change. Alex de Vries (2023a) writes on this behalf "it would be advisable for developers not only to focus on optimizing AI, but also to critically consider the necessity of using AI in the first place, as it is unlikely that all applications will benefit from AI or that the benefits will always outweigh the costs.".

Chapter 7

Related work

Throughput of various quantisation methods

A recent paper by Rajput and Sharma (2024) analyses five quantisation methods on LLAMA-2-7B. The authors compare Gradient-based PostTraining Quantization (GPTQ), Activation-aware Weight Quantization (AWQ), GPT-Generated Model Language (GGML), GPTGenerated Unified Format (GGUF) and Bits and Bytes (BNB). The authors find that BNB is the worst-performing quantisation method in efficiency and perplexity, GGUF is the most efficient method. The authors hint that GGUF is optimised for GPU work, compared to BNB which is generic. The authors call for energy optimization an explicit design criterion alongside accuracy. The authors use perplexity on the wikitext-2 dataset. Our work differs from this work as it focuses on coding tasks and studies the StarCoder2 models instead of LLAMA-2.

Apart from this paper, we could not find literature discussing the slower inference of quantisation methods. Some non-academic sources report on a slower inference speed when using quantisation, such as the Hugging Face LLM optimisation tutorial and a GitHub issue. One open-source contributor even dedicated a page on GitHub to the reduced inference of quantised models, though unfortunately, the StarCoder2 and Phi-2 models are not part of this repository.

Estimating the carbon footprint of BLOOM

Based on sustainability papers and workshops, Luccioni et al. (2023b) estimated the Carbon Footprint of BLOOM (BigScience Large Open-science Open-access Multilingual Language Model), a 176B Parameter Language Model. The authors present an introduction similar to our work, emphasising the need for reducing carbon emissions to limit global warming. By examining the impact of equipment manufacturing up until model deployment, the authors cover the tangible part of the Life Cycle Assessment (LCA). To estimate the impact of inference the authors simulated 18 days with 558 real-time queries per hour. They estimate that the model used around 689 842 kW for training and inference, accounting for 39.3 tonnes of CO_2-emissions. With around 11.2 tons of CO_2-emissions for the hardware manufacturing this totals at 50.5 tonnes of CO_2-emissions. To set this number in perspective, to remain within 1.5 degrees of warming each person can emit up to 2.3 tonnes of CO_2 per year[23].

This is one of the first papers that assesses most stages in the LCA of LLMs, nevertheless, it greatly underestimates the impact of inference. Popular LLMs have millions of queries, making the inference phase a lot more expensive than model training (Luccioni et al., 2023a; de Vries, 2023a). Even though the estimated carbon footprint of 50 tonnes is impressive, it falls short of generalising this study to popular LLMs.

[23]See this publication by Oxfam.

Our work differs as we try to reduce the energy consumption of the inference phase, the work of Luccioni et al. (2023b) estimates the carbon footprint of the majority of BLOOM's lifecycle.

Reporting on energy consumption

Even though research on this topic is increasing, there seems to be a divide between the technical experts developing AI and the critics who examine energy consumption. For example, a high-impact position paper by Han et al. (2021) describing the past, present and future of pre-trained models does not contain the words 'climate' or 'energy'. This shows that even amongst AI experts sustainability is not incorporated in daily practice. To include this topic in the development and publishing of models, Lottick et al. (2019) state that the carbon footprint of algorithms must be reported such that users can make informed decisions. To enhance this reporting, high-impact conferences such as NeurIPS, ICLR and ICML could force their submissions to include their energy and carbon footprint in papers, to set a standard.

Another way to consider energy consumption during the development of AI is by predicting the energy consumption. If a model architecture is known, each part can be measured by itself using CodeCarbon and this information can be stored in a database (Getzner and Charpentier, 2023). This database can then be used to predict the energy consumption of a constructed model, based on its architecture. Even though the authors predict the parts and not the entire model, their framework provides an interesting direction for future work. Predicting and optimising the energy consumption of models could prevent energy consumption and carbon emissions in the first place, instead of merely limiting them after design.

Chapter 8

Conclusion

The recent developments of AI have led to mass adoption of LLMs. More than 100 million people interact with ChatGPT and programmers using code LLMs describe increased productivity and more joy in their work. But this mass adoption comes at a cost: AI consumes a significant amount of electricity, causing the CO_2-emissions of companies such as Microsoft and Google to increase by double digits. With global temperatures now 1.48 degrees Celsius above the pre-industrial age, irreversible climate change becomes more likely unless the limit the global warming by reducing greenhouse gas emissions. To contribute to this reduction in greenhouse gasses, we construct three experiments to answer our research question: **How can we reduce the energy consumption of code Large Language Models with minimal harm to accuracy?**

In our first experiment, we used the Pythia-2.8B model to see if we could reduce training time by using intermediate weights of the training phase. Our experiment shows a positive correlation between training time and performance, the energy consumption during inference did not change. Given the increasing performance with training time and the higher impact of the inference compared to training, we deem reducing training time to reduce energy consumption. In our second experiment, we use quantisation to compress the weights of the 3B and 7B StarCoder2 models. First of all, we could not reproduce the scores of the authors and they did not reply when we asked them. We then evaluated these quantised models on HumanEval+ and found that the pass@1 score for StarCoder2-7B with 8-bit quantisation matches the full-sized model. For all other configurations, the pass@1 score on HumanEval+ is lower than their full-sized counterparts. The energy consumption increased by at least 19% for all configurations, presumably because quantised models have a lower throughput. However, the reduced throughput is not proportional to the energy consumption. This shows that the average amount of energy per second was reduced, but the longer execution times surpass this reduction. In our third and final experiment, we pruned the last layers of Phi-2, StarCoder2-3B and StarCoder2-7B. We found that the performance dropped significantly with the number of layers that were pruned, whereas the energy reduction was minimal. We therefore deem the pruning of layers unsuitable for reducing the energy consumption of code LLMs. For future work, we deem an improved throughput for quantised models a promising field to reduce the energy consumption of code LLM inference.

Acknowledgement

First and foremost, I wish to thank my supervisors Ana Oprescu and Jelle Zuidema. My initial idea to combine research on AI with climate change may have been a risky direction, but your guidance and questions helped me to scope this research to something tangible. I believe the experiments provide interesting options for future work and I hope we can extend this work and contribute to reducing energy consumption of LLMs. Thank you! I was fortunate to have conversations with PhD students which helped to shape my experiments. Marco Brohet, Jaap Jumelet, Anna Langendijk, Jelle van Dijk, Baohao Liao and Oskar van der Wal, thank you for your time and for answering my detailed, and sometimes stupid, questions. I am also grateful to SURF and the Faculty of Science (FNWI, UvA) for providing access to the National Supercomputer Snellius.

Bibliography

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023). GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. arXiv:2305.13245 [cs].

Allen-Zhu, Z. and Li, Y. (2024). Physics of Language Models: Part 3.3, Knowledge Capacity Scaling Laws. arXiv:2404.05405 [cs].

Almeida, M., Laskaridis, S., Mehrotra, A., Dudziak, L., Leontiadis, I., and Lane, N. D. (2021). Smart at what cost? characterising mobile deep neural networks in the wild. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, pages 658–672, New York, NY, USA. Association for Computing Machinery.

Argerich, M. F. and Patiño-Martínez, M. (2024). Measuring and Improving the Energy Efficiency of Large Language Models Inference. *IEEE Access*, 12:80194–80207. Conference Name: IEEE Access.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. (2021). Program Synthesis with Large Language Models. arXiv:2108.07732 [cs].

Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473 [cs, stat].

Belkada, Y. and Dettmers, T. (2022). A Gentle Introduction to 8-bit Matrix Multiplication for transformers at scale using transformers, accelerate and bitsandbytes.

Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? . In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, pages 610–623, New York, NY, USA. Association for Computing Machinery.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166. Conference Name: IEEE Transactions on Neural Networks.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., Skowron, A., Sutawika, L., and Wal, O. V. D. (2023). Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling. In *Proceedings of the 40th International Conference on Machine Learning*, pages 2397–2430. PMLR. ISSN: 2640-3498.

Cao, Q., Balasubramanian, A., and Balasubramanian, N. (2020). Towards Accurate and Reliable Energy Measurement of NLP Models.

Castaño, J., Martínez-Fernández, S., Franch, X., and Bogner, J. (2023). Exploring the Carbon Footprint of Hugging Face's ML Models: A Repository Mining Study. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs].

Copernicus (2024). Global Climate Highlights 2023. Technical report, European Commission.

Dayarathna, M., Wen, Y., and Fan, R. (2016). Data Center Energy Consumption Modeling: A Survey. *IEEE Communications Surveys & Tutorials*, 18(1):732–794. Conference Name: IEEE Communications Surveys & Tutorials.

de Vries, A. (2023a). The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194. Publisher: Elsevier.

de Vries, H. (2023b). Go smol or go home.

Delanoë, P., Tchuente, D., and Colin, G. (2023). Method and evaluations of the effective gain of artificial intelligence models for reducing CO2 emissions. *Journal of Environmental Management*, 331:117261.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. (2021). 8-bit Optimizers via Block-wise Quantization.

Dettmers, T., Lewis, M., Shleifer, S., and Zettlemoyer, L. (2022). 8-bit Optimizers via Block-wise Quantization. arXiv:2110.02861 [cs].

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs].

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

European Parliament (2023a). Artificial Intelligence Act.

European Parliament (2023b). EU AI Act: first regulation on artificial intelligence.

Frantar, E. (2023). How to run on multi GPUs?

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2023). GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs].

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G. S., and Friday, A. (2021). The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, 2(9):100340.

Getzner, J. and Charpentier, B. (2023). Accuracy is not the only Metric that matters: Estimating the Energy Consumption of Deep Learning Models.

Ghassemi, M., Birhane, A., Bilal, M., Kankaria, S., Malone, C., Mollick, E., and Tustumi, F. (2023). ChatGPT one year on: who is using it, how and why? *Nature*, 624(7990):39–41. Bandiera_abtest: a Cg_type: Comment Publisher: Nature Publishing Group Subject_term: Machine learning, Computer science, Technology, Policy.

Gholami, S. and Omar, M. (2023). Can pruning make Large Language Models more efficient? arXiv:2310.04573 [cs].

Ghosh, S., Martonosi, M., and Malik, S. (1997). Cache miss equations: an analytical representation of cache misses. In *Proceedings of the 11th international conference on Supercomputing - ICS '97*, pages 317–324, Vienna, Austria. ACM Press.

Gkinko, L. and Elbanna, A. (2023). The appropriation of conversational AI in the workplace: A taxonomy of AI chatbot users. *International Journal of Information Management*, 69:102568.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. Google-Books-ID: omivDQAAQBAJ.

Google Sustainability (2024). Environmental Report. Technical report.

Guo, H., Brandon, W., Cholakov, R., Ragan-Kelley, J., Xing, E. P., and Kim, Y. (2024). Fast Matrix Multiplications for Lookup Table-Quantized LLMs. arXiv:2407.10960 [cs].

Gutiérrez, M., Calero, C., García, F., and Moraga, M. (2024). The Effects of Class Balance on the Training Energy Consumption of Logistic Regression Models. In Araújo, J., de la Vara, J. L., Santos, M. Y., and Assar, S., editors, *Research Challenges in Information Science*, pages 324–337, Cham. Springer Nature Switzerland.

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., Zhang, L., Han, W., Huang, M., Jin, Q., Lan, Y., Liu, Y., Liu, Z., Lu, Z., Qiu, X., Song, R., Tang, J., Wen, J.-R., Yuan, J., Zhao, W. X., and Zhu, J. (2021). Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

Hansen, J. E., Sato, M., Simons, L., Nazarenko, L. S., Sangha, I., Kharecha, P., Zachos, J. C., von Schuckmann, K., Loeb, N. G., Osman, M. B., Jin, Q., Tselioudis, G., Jeong, E., Lacis, A., Ruedy, R., Russell, G., Cao, J., and Li, J. (2023). Global warming in the pipeline. *Oxford Open Climate Change*, 3(1):kgad008.

Hiller, J. and Herrera, S. (2024). Tech Industry Wants to Lock Up Nuclear Power for AI. Section: Business.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv:1503.02531 [cs, stat].

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. Conference Name: Neural Computation.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., Driessche, G. v. d., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. (2022). Training Compute-Optimal Large Language Models. arXiv:2203.15556 [cs].

Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. arXiv:1801.06146 [cs, stat].

Höchtl, J., Parycek, P., and Schöllhammer, R. (2016). Big data in the policy cycle: Policy decision making in the digital era. *Journal of Organizational Computing and Electronic Commerce*, 26(1-2):147–169. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10919392.2015.1125187.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12):248:1–248:38.

Ju, T., Sun, W., Du, W., Yuan, X., Ren, Z., and Liu, G. (2024). How Large Language Models Encode Context Knowledge? A Layer-Wise Probing Study. In Calzolari, N., Kan, M.-Y., Hoste, V., Lenci, A., Sakti, S., and Xue, N., editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 8235–8246, Torino, Italia. ELRA and ICCL.

Jurafsky, D. (2000). *Speech & language processing*. Pearson Education India, 3 edition.

Kal, H., Choi, H., Jeong, I., Yang, J.-S., and Ro, W. W. (2023). A convertible neural processor supporting adaptive quantization for real-time neural networks. *Journal of Systems Architecture*, 145:103025.

Kalliamvakou, E. (2022). Quantifying GitHub Copilot's impact on developer productivity and happiness.

Khan, K. N., Hirki, M., Niemi, T., Nurminen, J. K., and Ou, Z. (2018). RAPL in Action: Experiences in Using RAPL for Power Measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 3(2):1–26.

Knight, W. (2024). OpenAI's CEO Says the Age of Giant AI Models Is Already Over. *Wired*. Section: tags.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the Carbon Emissions of Machine Learning. arXiv:1910.09700 [cs].

Lambert, N., Castricato, L., von Werra, L., and Havrilla, A. (2022). Illustrating Reinforcement Learning from Human Feedback (RLHF).

Lenton, T. M., Held, H., Kriegler, E., Hall, J. W., Lucht, W., Rahmstorf, S., and Schellnhuber, H. J. (2008). Tipping elements in the Earth's climate system. *Proceedings of the National Academy of Sciences*, 105(6):1786–1793. Publisher: Proceedings of the National Academy of Sciences.

Lenton, T. M., Rockström, J., Gaffney, O., Rahmstorf, S., Richardson, K., Steffen, W., and Schellnhuber, H. J. (2019). Climate tipping points — too risky to bet against. *Nature*, 575(7784):592–595. Bandiera_abtest: a Cg_type: Comment Number: 7784 Publisher: Nature Publishing Group Subject_term: Climate change, Climate sciences, Environmental sciences, Policy.

Li, P., Yang, J., Islam, M. A., and Ren, S. (2023a). Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models. arXiv:2304.03271 [cs].

Li, X., Fang, Y., Liu, M., Ling, Z., Tu, Z., and Su, H. (2023b). Distilling Large Vision-Language Model with Out-of-Distribution Generalizability. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2492–2503, Paris, France. IEEE.

Liao, B. (2024). Personal communication.

Liu, D., Wang, Z., Wang, B., Chen, W., Li, C., Tu, Z., Chu, D., Li, B., and Sui, D. (2024). Checkpoint Merging via Bayesian Optimization in LLM Pretraining. arXiv:2403.19390 [cs].

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. (2023). Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. 37, New Orleans (USA).

Lottick, K., Susai, S., Friedler, S. A., and Wilson, J. P. (2019). Energy Usage Reports: Environmental awareness as part of algorithmic accountability. arXiv:1911.08354 [cs, stat].

Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., Pykhtar, D., Liu, J., Wei, Y., Liu, T., Tian, M., Kocetkov, D., Zucker, A., Belkada, Y., Wang, Z., Liu, Q., Abulkhanov, D., Paul, I., Li, Z., Li, W.-D., Risdal, M., Li, J., Zhu, J., Zhuo, T. Y., Zheltonozhskii, E., Dade, N. O. O., Yu, W., Krauß, L., Jain, N., Su, Y., He, X., Dey, M., Abati, E., Chai, Y., Muennighoff, N., Tang, X., Oblokulov, M., Akiki, C., Marone, M., Mou, C., Mishra, M., Gu, A., Hui, B., Dao, T., Zebaze, A., Dehaene, O., Patry, N., Xu, C., McAuley, J., Hu, H., Scholak, T., Paquet, S., Robinson, J., Anderson, C. J., Chapados, N., Patwary, M., Tajbakhsh, N., Jernite, Y., Ferrandis, C. M., Zhang, L., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. (2024). StarCoder 2 and The Stack v2: The Next Generation. arXiv:2402.19173 [cs].

Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., Li, G., Zhou, L., Shou, L., Zhou, L., Tufano, M., Gong, M., Zhou, M., Duan, N., Sundaresan, N., Deng, S. K., Fu, S., and Liu, S. (2021). CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. volume 35.

Luccioni, A. S., Jernite, Y., and Strubell, E. (2023a). Power Hungry Processing: Watts Driving the Cost of AI Deployment? arXiv:2311.16863 [cs].

Luccioni, A. S., Viguier, S., and Ligozat, A.-L. (2023b). Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model. *Journal of Machine Learning Research*, 24(253):1–15.

Ma, X., Fang, G., and Wang, X. (2023). LLM-Pruner: On the Structural Pruning of Large Language Models. *Advances in Neural Information Processing Systems*, 36:21702–21720.

Microsoft (2024). Microsoft 2024 Environmental Sustainability Report. Sustainability Report, Microsoft.

Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Noguer i Alonso, M. (2024). Key Milestones in Natural Language Processing (NLP) 1950 - 2024.

Noy, S. and Zhang, W. (2023). Experimental evidence on the productivity effects of generative artificial intelligence. *Science: A machine-intelligent world*, 381(6654):187 – 192.

NVIDIA (2012). NVIDIA System Management Interface.

Olah, C. (2015). Understanding LSTM Networks.

OpenAI (2022). Introducing ChatGPT.

OpenAI (2024). GPT-4 Technical Report. arXiv:2303.08774 [cs].

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a Method for Automatic Evaluation of Machine Translation. In Isabelle, P., Charniak, E., and Lin, D., editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Patel, P., Choukse, E., Zhang, C., Goiri, , Warrier, B., Mahalingam, N., and Bianchini, R. (2024). Characterizing Power Management Opportunities for LLMs in the Cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, volume 3 of *ASPLOS '24*, pages 207–222, New York, NY, USA. Association for Computing Machinery.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global Vectors for Word Representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018a). Deep contextualized word representations. arXiv:1802.05365 [cs].

Peters, M. E., Neumann, M., Zettlemoyer, L., and Yih, W.-t. (2018b). Dissecting Contextual Word Embeddings: Architecture and Representation. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J., editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium. Association for Computational Linguistics.

Pinto, I., Barnes, C., Kleeman, M., and Otto, F. E. L. (2024). Extreme heat killing more than 100 people in Mexico hotter and much more likely due to climate change. Technical report.

Poynting, M. (2024). World's first year-long breach of key 1.5C warming limit.

Project, T. S. (2023). Energy, climate: which virtual words for which real world? page 73.

Qi, X., Wang, J., and Zhang, L. (2023). Understanding Optimization of Deep Learning via Jacobian Matrix and Lipschitz Constant. arXiv:2306.09338 [cs, math, stat].

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training.

Rajput, S. and Sharma, T. (2024). Benchmarking Emerging Deep Learning Quantization Methods for Energy Efficiency. volume 21.

Reid, L. (2024). Generative AI in Search: Let Google do the searching for you.

Rolnick, D., Donti, P. L., Kaack, L. H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A. S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A., Luccioni, A. S., Maharaj, T., Sherwin, E. D., Mukkavilli, S. K., Kording, K. P., Gomes, C. P., Ng, A. Y., Hassabis, D., Platt, J. C., Creutzig, F., Chayes, J., and Bengio, Y. (2022). Tackling Climate Change with Machine Learning. *ACM Computing Surveys*, 55(2):42:1–42:96.

Rothe, S., Narayan, S., and Severyn, A. (2020). Leveraging Pre-trained Checkpoints for Sequence Generation Tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280. arXiv:1907.12461 [cs].

Schmidt, V., Kamal, G., Joshi, A., Feld, B., Conell, L., Laskaris, N., Blank, D., Wilson, J., Friedler, S., and Luccioni, S. (2021). CodeCarbon.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423. Conference Name: The Bell System Technical Journal.

Shannon, C. E. (1951). The redundancy of English. *Cybernetics; Transactions of the 7th Conference*.

Shazeer, N. (2019). Fast Transformer Decoding: One Write-Head is All You Need. arXiv:1911.02150 [cs].

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2020). Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs].

Spirals (2019). pyRAPL documentation.

Staudemeyer, R. and Morris, E. (2019). *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

UN (2015). Paris Agreement. *Paris Agreements United Nations*, 55(4):743–755.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, , and Polosukhin, I. (2017). Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Veiga, J., Enes, J., Expósito, R. R., and Touriño, J. (2018). BDEv 3.0: Energy efficiency and microarchitectural characterization of Big Data processing frameworks. *Future Generation Computer Systems*, 86:565–581.

Verdecchia, R., Sallou, J., and Cruz, L. (2023). A systematic review of Green AI. *WIREs Data Mining and Knowledge Discovery*, 13(4):e1507. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1507.

Verhagen, L., Frijters, S., Sabel, P., and Mohren, E. (2024). Hoeveel water en energie gebruikt uw favoriete chatbot? *de Volkskrant*.

Villalobos, P., Sevilla, J., Heim, L., Besiroglu, T., Hobbhahn, M., and Ho, A. (2022). Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning. arXiv:2211.04325 [cs].

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019). Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Publisher: Association for Computational Linguistics.

Wang, W., Chen, W., Luo, Y., Long, Y., Lin, Z., Zhang, L., Lin, B., Cai, D., and He, X. (2024). Model Compression and Efficient Inference for Large Language Models: A Survey. arXiv:2402.09748 [cs].

Weaver, V. M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., and Moore, S. (2012). Measuring Energy and Power with PAPI. In *2012 41st International Conference on Parallel Processing Workshops*, pages 262–268. ISSN: 2332-5690.

Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2022). Finetuned Language Models are Zero-Shot Learners.

WMO (2024). World Meteorological Organization Global Annual to Decadal Climate Update (2024-2028). Technical report.

Yuan, Y., Zhang, J., Zhang, Z., Chen, K., Shi, J., Stoico, V., and Malavolta, I. (2024). The Impact of Knowledge Distillation on the Energy Consumption and Runtime Efficiency of NLP Models.

Zamfirescu-Pereira, J., Wong, R. Y., Hartmann, B., and Yang, Q. (2023). Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery.

Zhou, K., Zhu, Y., Chen, Z., Chen, W., Zhao, W. X., Chen, X., Lin, Y., Wen, J.-R., and Han, J. (2023). Don't Make Your LLM an Evaluation Benchmark Cheater. arXiv:2311.01964 [cs].

Appendix A

Energy consumption tables StarCoder2

Energy consumption of full-sized StarCoder2-3B & StarCoder2-7B.

	3B		7B	
	128 tokens	**256 tokens**	**128 tokens**	**256 tokens**
DRAM (Wh)	5.73 (±0.04)	11.25 (±0.09)	7.24 (±0.01)	14.41 (±0.03)
GPU (Wh)	21.60 (±0.33)	41.72 (±0.13)	35.88 (±0.03)	70.57 (±0.07)
CPU (Wh)	33.43 (±2.3)	63.04 (±0.44)	39.70 (±0.07)	84.34 (±1.32)
Total (Wh)	60.76 (±2.57)	116.01 (±0.64)	82.82 (±0.10)	169.33 (±1.32)
Est. CO_2in grams	21.53 (±0.91)	40.88 (±0.38)	29.34 (±0.04)	59.99 (±0.47)

Table 4: Energy consumption in Wh for the StarCoder2-3B and StarCoder2-7B models on 128 and 256 tokens. The energy is measured over the inference of the HumanEval coding evaluation framework with 164 individual tasks. The estimated CO_2-emissions come from CodeCarbon as described in Section 2.3.2. The values are the average of five experiments, in brackets the standard deviation.

Energy consumption of quantised StarCoder2-3B

	4-bit		8-bit	
	128 tokens	**256 tokens**	**128 tokens**	**256 tokens**
DRAM (Wh)	11.41 (0.03)	22.34 (0.20)	43.02 (0.50)	84.72 (0.70)
GPU (Wh)	22.72 (0.06)	44.11 (1.13)	73.08 (0.86)	141.71 (3.85)
CPU (Wh)	71.19 (0.87)	128.70 (8.94)	110.04 (20.20)	55.08 (26.56)
Total (Wh)	105.32 (0.96)	195.16 (8.01)	226.14 (19.32)	281.52 (24.25)
Est. CO_2in grams	37.32 (0.34)	69.15 (2.84)	80.12 (6.84)	99.74 (8.59)

Table 5: Energy consumption in Wh for the StarCoder2-3B model on 128 and 256 tokens, quantised using 4 or 8-bit. The energy is measured over the inference of the HumanEval coding evaluation framework with 164 individual tasks. The estimated CO_2-emissions come from CodeCarbon as described in Section 2.3.2.

Energy consumption of quantised StarCoder2-7B

	4-bit		8-bit	
	128 tokens	**256 tokens**	**128 tokens**	**256 tokens**
DRAM (Wh)	11.93 (0.10)	23.83 (0.15)	29.04 (0.27)	58.70 (0.25)
GPU (Wh)	29.47 (0.16)	58.17 (0.23)	59.01 (0.44)	113.69 (0.38)
CPU (Wh)	74.19 (7.32)	133.04 (0.87)	21.62 (3.62)	36.49 (1.38)
Total (Wh)	115.60 (7.56)	215.03 (1.26)	109.67 (4.19)	208.88 (2.00)
Est. CO_2in grams	40.96 (2.68)	76.19 (0.45)	38.86 (1.48)	74.01 (0.71)

Table 6: Energy consumption in Wh for the StarCoder2-7B model on 128 and 256 tokens, quantised using 4 or 8-bit. The energy is measured over the inference of the HumanEval coding evaluation framework with 164 individual tasks. The estimated CO_2-emissions come from CodeCarbon as described in Section 2.3.2.

Increase in cache misses of quantisation compared to full-sized model

	4-bit		8-bit	
	L1 misses	**L2 misses**	**L1 missess**	**L2 misses**
Run 1	77%	45%	322%	428%
Run 2	77%	50%	315%	439%
Run 3	73%	51%	302%	437%
Run 4	75%	47%	316%	430%
Run 5	75%	48%	318%	442%
Avg. increase	75%	48%	315%	435%

Table 7: The increased L1 and L2 cache misses for each of the five runs in experiment 2 for StarCoder2-7B on 128 tokens.

Appendix B

Energy consumption and pass@1 for Phi-2

	Phi-2, 128 tokens	**Phi-2, 256 tokens**
DRAM (Wh)	18.14	34.41
GPU (Wh)	33.54	63.52
CPU (Wh)	101.70	46.84
Total (Wh)	153.38	144.76
Est. CO_2in grams	36.46	51.29

Table 8: Energy consumption in Wh for Phi-2 on 128 and 256 tokens. The energy is measured over the inference of the HumanEval coding evaluation framework with 164 individual tasks. The estimated CO_2-emissions come from CodeCarbon as described in Section 2.3.2.

	HumanEval (sanitized)	**HumanEval+ (sanitized)**
Phi-2, 128 tokens	0.451	0.409
Phi-2, 256 tokens	**0.482**	**0.439**

Table 9: Pass@1 score for Phi-2 on the HumanEval+ set with predicting 128 and 256 tokens. Models were used to see whether our pipeline caused the deviation between the StarCoder2 models and the paper.

Appendix C

Parallelisation

The StarCoder2-15B model consists of 15 billion parameters, making up to 32 GB in memory. Unsurprisingly, this size is not only infeasible for personal computers but also for a single GPU. With billions of parameters, using a LLM on a personal computer or laptop becomes infeasible. And for models with over 10 billion parameters, even a GPU with 40 Gigabytes of memory is insufficient. We then need to divide the model across multiple GPUs. We then spread the work over multiple GPUs, called parallelization, to allow computations with such big models.

Two parallelisation techniques could help overcome this problem, which we introduce briefly.

- **Distributed data parallelisation** splits the training data across GPUs, maintaining a replicate of the model at each GPU.
- **Tensor parallelisation** splits the model across various GPUs and instead leaves the data intact across GPUs.

Distributed Data Parallelization (DDP) is mainly useful during training with large data sets. If the data set cannot be placed on a single GPU but the model can, DDP is a useful technique to parallelise the training. In our case, the model is too big for the GPU and we need to perform tasks during inference as we do not train the model ourselves. Therefore we apply tensor parallelisation to conduct experiments with StarCoder2-15B. Tensor parallelisation divides the layers over GPUs in such a way that the model can run in parallel without GPUs having to wait on the input of previous layers (Shoeybi et al., 2020)[24].

Implementation

As we use models available on Hugging Face, we consult the Hugging Face tutorial on model parallelism which provides four options. Megatron-LM is very model-specific, and the StarCoder2 authors advised us not to use this method as the implementation is hard. The next option is parallelformers, which requires a sharding policy based on the model architecture. However, StarCoder2 uses grouped-query attention which was first mentioned in 2023, and the parallelformers repository has been inactive for over 2 years. Though we attempted to construct a policy that would work for the model, the implementation did not succeed and similar issues were mentioned on GitHub. We then decided that we would not use parallelformers. The third option, SageMaker is a proprietary solution that can only be used on Amazon Web Services. Due to budget constraints, the use of SageMaker is not feasible in this project. Finally, OSLO has tensor parallelism implementation based on the Transformers but also has an outdated repository on GitHub.

[24]For a more detailed explanation, please see this explanation on GitHub that discusses horizontal parallelisation.

Consequently, the implementation of model parallelisation is time-intensive. The High Performance Computing team of our computation resource provider SURF advised us not to work with model parallelisation. Next to that, using multiple GPUs could introduce an unfair comparison between the smaller StarCoder2-3B and StarCoder2-7B models that run on a single GPU. Finally, there are indications that model parallelism could negatively impact the memory usage of quantisation (Frantar, 2023). We thus exclude StarCoder2-15B from our experiments.