Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Econometrics and Management Science
Business Analytics and Quantitative Marketing

Assessing Task-Specific Performance Gains from Parameter-Efficient Fine-Tuning of Autoregressive Large Language Models

Philip Verdonk (477745)

Supervisor:	dr. Paul Bouman
Second assessor:	Markus Mueller
Company Supervisor:	Gijs Gubbels
Date final version:	21st March 2024

Abstract

This study investigates the effectiveness and economic feasibility of fine-tuning Large Language Models (LLMs). With a focus on the open-source LLaMA-2 model of Touvron, Martin et al. (2023), Parameter-Efficient Fine-Tuning (PEFT) is used in addition to an innovative context extraction mechanism. Specifically, Quantized Low-Rank Adaptation (QLoRA) adds a trainable adapter to an LLM, after which the Flash Attention (FA) technique is used for efficient context extraction. It assesses the application of the fine-tuned models for two natural language processing tasks. The evaluation demonstrates that the fine-tuned LLaMA-2 models significantly outperform their base counterparts in speed, efficiency, and cost-effectiveness, offering advantages for many task-specific applications. It identifies the optimal fine-tuned model choice as the 13 billion parameter variant for applications with an emphasis on accuracy, and the 7 billion parameter model for applications focused on speed. This study provides a valuable framework for applying efficient fine-tuning methods to any open-source LLM, making advanced artificial intelligence techniques more accessible. Not only are computational costs reduced and is quality enhanced, but also speed is increased. By proving that smaller, efficiently trained LLMs can outperform their larger variants, this work contributes to lowering the threshold of using such models.

Contents

1 Introduction

Not a day goes by without significant advancements in Artificial Intelligence (AI), particularly within the realm of Natural Language Processing (NLP). This field, which enables machines to understand human language, has seen remarkable progress thanks to the development of Large Language Models (LLMs). These pre-trained models on a large amount of text, characterized by their vast number of parameters and the transformer neural network architecture of Vaswani et al. (2017), have set new standards for machine understanding, generation, and translation of human language. The capabilities of LLMs span a wide range of problems, from powering complex linguistic systems, such as question-answering systems, to writing and augmenting high-quality programming code and predicting protein structures, as shown in Z. Lin et al. (2023). This makes the potential of LLM applications auspicious.

Given the growing interest in generative AI in the global market, a McKinsey report from Michael Chui & Zemmel (2023) predicted that the novel technology would add $2.6 to $4.4 trillion to the economy, yearly. Along with this substantial economic potential, it has the power to solve many present complex tasks. However, the development and deployment of state-of-the-art LLMs has predominantly been the domain of major tech companies like OpenAI and Google, given the substantial resources required for training such models. In their modern digital space race, the best-performing LLMs are kept proprietary, thus there is no thorough understanding of the architecture and data the models are made of. Zheng et al. (2023) provided a rating mechanism that is used to assess LLM performance. They created an open, crowd-sourced platform where these ratings are presented. The leaderboard can be found on this webpage. Only two of the top 20 models are open-source. In particular, the highest-ranking open-source LLM as of March 8, 2024, holds the 16th place. Especially the undisclosed model information and training data, and the possibly irresponsible usage of private data, have stimulated the development of open-source models. Platforms such as HuggingFace enable researchers to develop and share their findings. Unfortunately, the high costs of the required training resources often remain prevalent. This poses significant barriers to entry for academic institutions and smaller organizations, limiting the scope of open innovation within the field.

In response to these challenges, Meta AI released its open-source generative LLM, named *Large Language Model of Meta AI*, or LLaMA by Touvron, Lavril et al. (2023). It marks a pivotal moment in the democratization of access to large language models. By offering a model where its training data is disclosed and fine-tuning is available to private, task-specific data, LLaMA paves the way for broader research and development activities. The model democratizes access to large language models and allows more researchers to study and innovate the techniques. Already

improved by Meta to LLaMA-2 in Touvron, Martin et al. (2023), the model comes in three sizes: 7 billion parameters, 13 billion parameters, and 70 billion parameters. Still, fine-tuning all the parameters in an LLM requires a vast amount of computing resources, making it inefficient for academic researchers and small businesses. Nonetheless, even when full fine-tuning is performed, the general abilities lagging behind the performance of proprietary LLMs, thus the economic trade-offs are not viable either.

Recently, novel methods were developed that significantly reduce the number of trainable parameters to specialize LLMs in certain tasks. Houlsby et al. (2019) found that by freezing the original model parameters and training an added adapter with learnable parameters, similar performance can be achieved compared to completely fine-tuning an LLM. The authors called the collection of related methods *Parameter-Efficient Fine-Tuning*, or PEFT. Hu et al. (2021) went a step further, creating an efficient method called *Low-Rank Adaptation* (LoRA), that performs similarly or better than normal fine-tuning while reducing the number of trainable parameters by 10,000 times. In this context, fine-tuning should not be confused with transfer learning, which is teaching a pre-trained model a specific application by training it on more data. Fine-tuning involves the adjustment of parameters, either new or existing.

This study is structured to assess the task-specific performance gains of autoregressive LLMs using PEFT techniques. The term autoregressive indicates that only past text is taken into account to predict successive text, referred to as text generation. Essentially, an analysis is conducted on the effect of parameter-efficient model enhancement for specific textual tasks. At the time of the beginning of this research, a promising model is LLaMA-2, an open-source autoregressive large language model. According to Touvron, Martin et al. (2023), it is trained on roughly 8 trillion characters of disclosed textual data, with a high focus on safety and unbiasedness. Although other open-source models, such as Falcon of Penedo et al. (2023) and Mistral of Jiang et al. (2023) sometimes outperform LLaMA-2 in certain tasks, the longer-existing and more reviewed steady base model of Meta will be used as the benchmark.

For the sake of this evaluation, two specific NLP tasks will be exploited. This research is conducted at Freeday.ai, a Rotterdam-based AI-enabled start-up. The application of LLaMA-2 models holds the potential to enhance their products, such as Customer Support and Know Your Customer, by providing high-quality AI agents that can handle repetitive tasks with unprecedented efficiency and accuracy. Freeday.ai helps enterprises to build a digital workforce that automates repetitive tasks. The company claims that this enhances the creativity and productivity of its clients, leading to improved revenue and consumer experience. Two common tasks Freeday can automate using LLMs are for instance search question generation (QG) to

look up information in a database using the semantics of an input, rather than the traditional keyword search. This could in theory reduce the cost and time for answering questions in a customer support conversation. The second is *named entity recognition* (NER), which is the ability of a model to extract entities from a text. The Know Your Customer product extracts entities, e.g. name, birth date, etc., from ID cards in order to verify identification. They now use expensive one-purpose models, which they hope to replace by less expensive and more transparent alternatives.

By focusing on semantic search query generation and NER, this research addresses two critical areas where improvements in LLM performance could lead to substantial benefits for research institutions and smaller AI-enabled enterprises with not many resources. Furthermore, the exploration of PEFT methods offers a pathway to leverage the capabilities of LLMs, without the exorbitant costs and computational power requirements associated with full model training. First, labeled data is gathered for the two tasks. Afterward, fine-tuning is performed by adding learnable adapters to the hidden model layers, vastly reducing the amount of parameters to be trained. The method of Dettmers et al. (2023), called Quantized Low-Rank Adaptation (QLoRA), uses this technique, while simultaneously reducing the memory requirements by an order of magnitude of 10. Additionally, an improved context extraction mechanism is imposed that uses computing units more efficiently, meaning that the best-performing and most expensive processing units are no longer compulsory in the fine-tuning process of large language models. This method, called Flash Attention by Dao (2023), makes the training of LLMs more accessible and less resource-intensive.

In this study, empirical evidence is provided for the positive effects of efficiently fine-tuning transformers in combination with an improved context attention mechanism. In addition, the ability of LLMs to perform NER and QG is explored and proven to be valuable. In particular, the research finds that fine-tuning LLaMA-2 models is economically viable and significantly rewarding in terms of performance. Concerning quality, speed, and cost-effectiveness, the fine-tuned variants of the LLaMA-2 model outclasses the base models in the two mentioned tasks. The fine-tuned models even outperform the greater models. However, it should be considered per task which model to use, because of the speed-quality tradeoff that arises. For the NER case, the 13 billion parameter model is the optimal choice as accuracy is essential. For the QG task, the 7 billion parameter model performs best due to its prediction speed.

As this efficient fine-tuning methodology can be applied to any open-source LLM, the research provides a clear and effective framework to make task-specific learning of large language models accessible to anyone, from individuals and academic researchers to small enterprises. The ultimate

goal of the research is to further democratize the use of LLMs, reducing computational costs, and enhancing the capabilities of AI agents deployed in the fields of customer service and other applications. Affirmatively stated in recent literature such as Hoffmann et al. (2022), the best performances are not achieved by the largest models, but by smaller models trained efficiently on more data.

The structure of this paper is as follows. After the introduction, the required knowledge and a foundational review of existing theories are provided in Section 2. Thereafter, the used data will be discussed in Section 3. Furthermore, the methods used for the research are presented and explained in Section 4. Following the methods, an overview of all the results from the analysis and implications of fine-tuning LLaMA-2 models are presented in Section 5. The findings of this research are summarized, declared, and discussed in the final section of this paper, finalized with a concise outline for further research.

2 Theoretical Background

In this section, a critical analysis of the important topics is presented in the form of a literature review. The most relevant academic literature on the well-performing transformer neural networks, parameter-efficient fine-tuning techniques, and the quantization methods are used to summarize and analyze the current state of knowledge.

2.1 Natural Language Processing

Fundamental for NLP is the transformation of words to numbers or vectors, explained in Jurafsky & Martin (2008). These high-dimensional vectors are called embeddings, representing the words but also encapsulating relationships to other words. As this embedding can store the context, this method is essential in NLP. More recently, the text is also split into standard pre-defined tokens, forming vocabularies for models. The LLaMA model vocabulary comprises 32k tokens (Touvron, Lavril et al. (2023)), GPT-1 has 40k tokens (Radford et al. (2018)) and its third generation model, GPT-3, an astonishing 50k tokens (Brown et al. (2020)). Following tokenization, embeddings are formed for the token indices. These embeddings are used to perform computations and predictions and are decoded afterward to return normal text. In the early days of NLP, the field consisted mostly of Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), as explained in Yin et al. (2017). These models could robustly interpret and predict sequential data such as text. For the sake of processing text, context is highly relevant. Take a long sentence, where relative pronouns are used. Context should be preserved carefully to perform tasks such as translation, summarization, and generation on a human level. However, in

these networks, problems occurred when sequences of textual input grew substantially. As RNNs had an advantage in preserving memory using internal states, extensions to this model arose. A flaw was discovered with respect to the length of the input data. As in a neural network, a gradient is computed during backward propagation to minimize some loss function and closely approximate the data. When the sequences extended, memory loss became inevitable. For text data, this meant that longer sentences were not correctly interpreted by the model.

A model designed to overcome this problem is called the *Long Short-Term Memory* network, or LSTM network. In Hochreiter & Schmidhuber (1997), the researchers tackled memory loss in traditional RNN by retaining existing short-term memory for thousands of time steps, hence the name *long short-term memory*. This characteristic made sure that there was a significantly smaller information loss for extensive sequential data input, by adding an extra state for long-term memory. Unfortunately, this model still had two major issues. First, the sophisticated architecture to retain the long short-term memory leads to high complexity and therefore high computational costs. Secondly, the network processes data sequentially, making it impracticable to parallelize the training. With long sequences and large datasets, the LSTM turned out to be a sub-optimal choice for NLP.

2.2 Transformer Neural Networks

In 2014, a group of scientists developed a method to improve sequence-to-sequence models for machine translation. Introduced in Bahdanau et al. (2014), the authors claim that the attention mechanism enables a model to rate the importance of parts of the input text, capturing all internal relations regardless of their positions. Queries, keys, and values are defined, which represent different transformations of the same input. Specifically, for each input ID of a word or token, a query is generated to search for other elements, keys are generated to be matched by queries, and values are the actual content we want to focus on. The mechanism computes a dot product from these queries and keys. Then, a SoftMax function (Luce (1959)) is applied that provides a set of weights. These weights are used to dynamically weigh the importance of different parts, or tokens, of the input data. This allows the model to focus more on relevant parts for better contextual understanding and processing of the input sequence. 'Paying attention' to all the parts of the input sequence, significantly improved the output generation and hence the translation performance. This was further improved in Luong et al. (2015). This multiplicative attention was effective for LSTMs and RNNs, but experienced problems during scaling.

In the search for a solution to these past breakthroughs, a couple of years later the *transformer neural network*, or *transformers* in short, was designed and introduced by Vaswani et al. (2017).

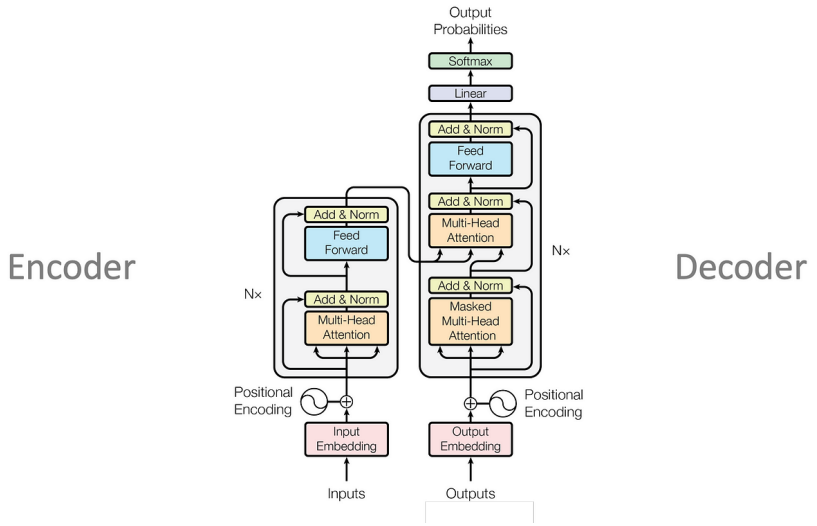


Figure 1: The Transformer Model architecture from Vaswani et al., comprising of an encoder-decoder structure.

This novel model architecture comprises an encoder and decoder, which can be implemented independently for different applications. The functioning of the transformer is visualized in detail in Figure 1.

To understand the functioning as described by the authors, a few operations need to be explained. After tokenizing the input as explained earlier, the information about the token positions should be incorporated. To convey token order, *Absolute Positional Encodings* (APE) are performed in advance. This method indexes the sequence, where the sine and cosine functions are applied for the even and odd indices, respectively. The formula for even indices is

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right). \tag{1}$$

For odd indices it is

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

where pos is the position in the sequence, i is the dimension index and d_{model} is the dimension of the embeddings, a defined architecture parameter. Subsequently, a positional embedding vector and the token embedding are summed to incorporate positional information. The choice of sinusoidal functions is due to their continuous and differentiable characteristic. As they are periodic and the range can continue infinitely, the sequence length can be longer than the transformer has seen during training because the functions can be extrapolated.

At the heart of the transformer lies the self-attention mechanism, which is a slight deviation from the prior mentioned attention mechanism. In this paper, the self-attention layer of a transformer was proposed. In this technique, the input embeddings are linearly transformed into

query, key, and value vectors. The queries are used to search for relevant information across all positions in the input sequence. The keys are matched against queries to compute attention scores, indicating the magnitude of focus on different parts of the input sequence. Finally, the values are weighted with the attention scores and summed. This produces self-attention information for each token, effectively allowing the model to aggregate information from the entire sequence based on its internal relevance. The calculations are simply $Q = XW^Q$, $K = XW^K$, and $V = XW^V$. Here, X is the input matrix, where each row corresponds to the embedding of a token. W^Q, W^K and W^V are the weight matrices for the queries, keys and values, which are learned model parameters. Additionally, Vaswani et al. designed the mechanism such that it could be scaled to many layers, introducing *Multi-Head Attention* or MHA. The scalability is due to the addition of a scaling factor $\frac{1}{\sqrt{d_k}}$ added to the dot product of the queries and keys such that the dot product would not grow large in magnitude. This would then lead to a vanishing gradient problem, causing significant problems in prediction. Thus, the attention equation for one layer simplifies to

$$\text{Attention(Q,K,V)} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V, \tag{3}$$

where the Q, K, and V are the query, key, and value matrices, respectively. d_k contains the keys. Generalizing to multiple attention heads, results in the following formula

$$\begin{aligned} \text{Multi-head}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_H)W^O \\ \text{head}_k &= \text{Attention}(QW_k^Q, KW_k^K, VW_k^V), \end{aligned} \tag{4}$$

with the weight matrices $W_k^Q \in \mathbb{R}^{d_k \times d_k}$, $W_k^K \in \mathbb{R}^{d_k \times d_k}$, $W_k^V \in \mathbb{R}^{d_v \times d_v}$and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ for the queries, keys and values, respectively. In the first line, W^O is the weight matrix for the entire projection of output. h is the number of heads in the MHA and d_k and d_v are the dimensions of the keys and values, respectively.

As can be deduced from the math and can be seen in Figure 2, the computational complexity scales quadratically. $\mathcal{O}(N^2)$, with N being the length of the input sequence. Every element added to the input means that attention must be computed again for all existing tokens. Many methods such as sparse approximation (Kitaev et al. (2020)) and low-rank approximation (Choromanski et al. (2020)) or even combinations of the two as in Beltagy et al. (2020) successfully reduced computational requirements to near-linear relations in the input length. This is arguably not enough, and Dao et al. (2022) proposed a different technique. It focuses on memory writing on GPUs, which ensures that memory will not overflow and significantly accelerates the attention algorithm. The technique called *Flash Attention* (FA) reaches an attention computation speed

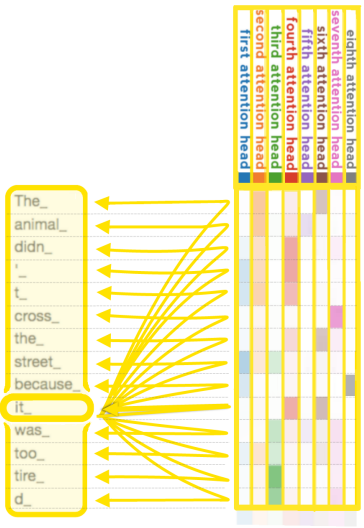
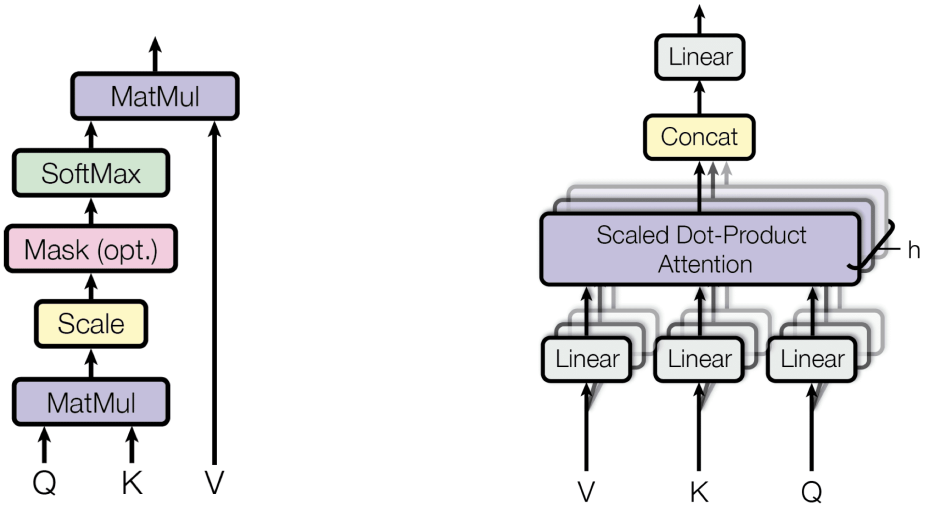


Figure 2: Self-attention mechanism on example sentence, where the relationship of all tokens to the token 'it_' is illustrated. Retrieved from Alammar (2018).



(a) Computation of the attention weights using queries, keys, and values in a self-attention layer.

(b) The Multi-Head Attention with h layers, all concatenated to one output.

Figure 3: A self-attention architecture illustration from Vaswani et al. (2017).

increase of $2-4\times$ for the GPT model of Radford et al. (2018). Extended in Dao (2023), *Flash Attention 2* (FA-2) reached a speed increase of $5-9\times$ compared to the regular self-attention mechanism. It improves on the original FA by optimizing how work is divided among GPUs, reducing unnecessary operations and enhancing parallel processing. FA-2 revolutionized deep learning network training, making it more accessible to lower-budget research groups.

Thus, both the encoder and the decoder consist of multiple layers of self-attention, where the small difference between the encoder and decoder MHA is the masked characteristic. This feature indicates that only takes past tokens are taken into account, as it does not know the future tokens which are to be predicted. Hence, the future tokens are masked.

Furthermore, the MHA is followed by the feedforward block, which helps the model learn more complex representations of the data. In general, it represents two fully connected feedforward neural networks, with a nonlinear activation function in between. In many models, the *Rectified Linear Unit*, or ReLU, is implemented as the activation function to allow non-linearity in the model for complex pattern extraction from the data. The first network expands the dimensionality of the input data to a higher dimension than the embedding size to capture complex features. The subsequent network, after the ReLU function, projects the output of the former back to the original embedding dimensionality.

Moreover, in the architecture shown in Figure 1, an *Add & Norm* operation is incorporated. The addition is simply summing the embedding vectors, ensuring equivalent dimensions. The normalization performed is the layer normalization of Ba et al. (2016), reducing the computational burden and minimizing training time. The operation is performed on the activations per layer, using the mean and standard deviation as follows

$$LayerNorm(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta. \tag{5}$$

Here, x is a vector of activations. μ and σ are the mean and standard deviation, respectively. γ is a learnable gain parameter and β is a learnable bias parameter, both of the same dimension as the activation vector x. The latter two can alter the normalization of single activations if dictated by the training data.

Lastly, output can be produced after projecting the resulting vectors on a higher-dimensional space, equal to the model's vocabulary size. This linear transformation is crucial to subsequently form a probability distribution over all possible tokens. That distribution is constructed using the SoftMax function. The next token produced is then the token with the highest probability.

The paper of Vaswani et al. showcases the transformer's superiority in problems like machine translation and language modeling, achieving state-of-the-art results on all NLP tasks. This scalable architecture has since become the foundation of many NLP models, reshaping deep learning methodology for natural language understanding and generation.

As progress persisted in the field of NLP in the last decade, a profound best practice emerged. Pre-training transformers in an unsupervised fashion on a large corpus of unlabeled text enabled the model to acquire significant language understanding. Proposed in Dai & Le (2015), pre-trained LSTMs proved to be more stable and generalize better. Pre-training was defined as masking a token and predicting it, iteratively scanning thoroughly through all the unlabeled text. The first transformer model that was pretrained and fine-tuned on NLP tasks was the OpenAI GPT model, proposed in Radford et al. (2018). This 12-layer stacked, decoder-only transformer

is unidirectional, meaning that it only takes previous context into account for predicting the next token. Trained with language from text, these models are then called autoregressive language models. This characteristic causes autoregressive transformers to predict more accurately, given the input or previous context, as explained in T. Wang et al. (2022). Due to this property, these models become more task-agnostic, meaning applicability to a variety of tasks. After pre-training, supervised fine-tuning for specific tasks can attain high performance with minimal adaptation to the core transformer.

The dominance of decoder-only models in generative tasks is also confirmed in T. Wang et al. (2022). The authors of the paper compared three model architectures: causal decoder-only, noncausal decoder-only, and encoder-decoder models. What causal means in this context, is that the model predicts the next token. They found that causal decoder-only trained autoregressively were superior, demonstrating tremendous adaptability in unsupervised scenarios. It proved to be the best at taking text into account and predicting the best answers. Concerning inference speed of text generation, another feature is added to speed up the decoder-only transformer model. *Multi-Query Attention* (MQA) indicates that the keys and values are shared between the different attention heads. N. M. Shazeer (2019) found that there is minimal change in the keys and values, thus when computed the first time, those numbers are a good approximation of the other keys and values and can thus be shared. While significantly reducing computations at a negligible cost of performance, it requires greater GPU memory, which is scarce and expensive.

Scientists at Google saw another potential in the transformer architecture. They found that pre-training an encoder-only model on a vast amount of text enabled the model to comprehend natural language more efficiently. Their model that incorporated this architecture in multiple language applications, was called BERT, or *Bi-directional Encoder Representations from Transformers*. Introduced in Devlin et al. (2018), after pre-training few extra data were required to align the attained knowledge to perform NLP tasks as never before. The authors describe the training process where they used a new technique called masked language modeling (MLM). MLM teaches transformer networks to understand the context of words and phrases by masking tokens from training data iteratively. A cross-entropy loss function is optimized using gradient-based algorithms, predicting the masked token depending on the tokens before and after the masked one, hence the bidirectional property. Due to this property, the model takes the previous context as well as future context into account. This enabled the model to extract many relationships in an enormous amount of texts fed into the model. With this contextual comprehension, BERT models are able to efficiently learn difficult tasks that require outstanding natural language understanding, e.g. summarization, topic extraction, and numerous others

(A. Wang, Singh et al. (2019), A. Wang, Pruksachatkun et al. (2019)).

The first model to implement the complete transformer architecture was the Text-to-Text Transfer Transformer, or T5, introduced by Raffel et al. (2019a). This was the first model able to understand language and also generate text. Attaining excellent translation performance in a variety of languages, this model is still a good choice as it resembles closely the human benchmark.

2.3 LLaMA model

Following the high performance of decoder-only models in generative tasks, scientists sought competitors to the GPT model. In Touvron, Lavril et al. (2023), the LLaMA model is introduced, and not long thereafter the LLaMA-2 model from Touvron, Martin et al. (2023) gained popularity. For further reference, LLaMA-2 will henceforth be denoted as LLaMA. The model architecture did not change, but training data was expanded and the context length doubled. In addition, the MQA layers are replaced for the greatest model of 70 billion parameters. The other two smaller versions are with 7 billion and 13 billion parameters. In fact, the exact number of parameters per model are 6 758 404 096, 13 047 157 760, and 68 993 032 192. The differences in parameters mainly result from the additional attention layers. Some important alterations are implemented to enhance the performance of LLaMA.

First, an enhanced positional embedding technique is exploited in the LLaMA model. Many scientists discussed that APE came short because it does not incorporate relative positioning in input sequences, suffers from a lack of generalization to unseen sequence length, and rapidly achieves high computational costs (Shaw et al. (2018), Liutkus et al. (2021), Kiyono et al. (2021)). As relative positional information could lead to more accurate encoding of the input, Shaw et al. (2018) introduced *Relative Positional Encoding* (RPE). This technique utilizes pairwise distances between positions in the input. In contrast to APE, which adds positional encodings to the input token embeddings element-wise, RPE adds the encodings element-wise to the attention matrix before it enters the SoftMax layer. RPE conflicts with the MQA feature for inference, as positional encodings cannot change when new words are generated. For text generation, RPE is not suitable as embeddings change for each newly generated token. Therefore, Su et al. (2021) introduced *Rotary Positional Encoding* (RoPE). Being a combination of both APE on the token embeddings using a transition matrix and RPE in the self-attention formulation, it provides an efficient method of incorporating positional information. As it decays with increasing relative distance, the encoding is scalable and compatible with linear self-attention. Due to these outstanding characteristics, the creators of the LLaMA model incorporated RoPE.

Furthermore, to achieve simplification of the layer normalization, a root mean square normalization is performed. B. Zhang & Sennrich (2019) argued that by not computing the mean, the computational complexity is reduced. Also, more stable gradients can be obtained by avoiding the subtraction of the mean, which could lead to numerical instability. The computation is as follows

$$\text{RMSNorm}(x) = \frac{x}{\sqrt{\frac{1}{H}\sum_{i=1}^{H} x_i^2}} \cdot \gamma \tag{6}$$

The variables are the same as in the layer normalization in Equation (5), with additionally H as the number of hidden units, x_i the individual activations in x, and γ as a scale parameter.

Moreover, in the Feed-Forward blocks of the LLaMA model, a more advanced activation function is used to increase the learning capabilities of the transformer. Introduced by N. Shazeer (2020), the *Swish Gated Linear Unit* (SwiGLU) function uses a gating mechanism to control the flow of information through the network for a more powerful representational capacity, while preserving nonlinearity. It is a variant of the *Gated Linear Unit* (GLU) function of Dauphin et al. (2016), which is defined as follows

$$GLU(a, b) = a \otimes \sigma(b), \tag{7}$$

where a and b are the output vectors of the linear transformations of the input. σ is the sigmoid activation function that gates the elements of a. The second component of SwiGLU is the Swish function, defined in Ramachandran et al. (2017) as

$$Swish(x) = x \cdot sigmoid(\beta x), \tag{8}$$

where β is a learned parameter. Ramachandran et al. found that their Swish function tends to perform better in deep learning networks than ReLU in a variety of challenging datasets. The former two equations are then combined by N. Shazeer in the function

$$SwiGLU(a, b) = a \otimes Swish(b). \tag{9}$$

Lastly, the *Grouped Query Attention* (GQA) is implemented in the 70 billion parameter model. Originally proposed in Ainslie et al. (2023), it replaces MQA, which uses only a single key-value head for speed increases during inference, but can lead to quality degradation. A generalization of MQA, GQA shares single key and value heads for each group of query heads. It is a combination of the straightforward multi-head implementation and the more recent multi-query implementation, which is visually displayed in Figure 4.

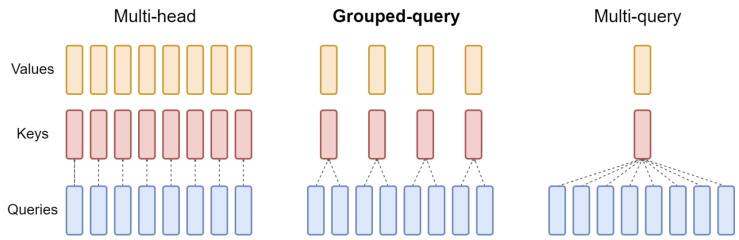


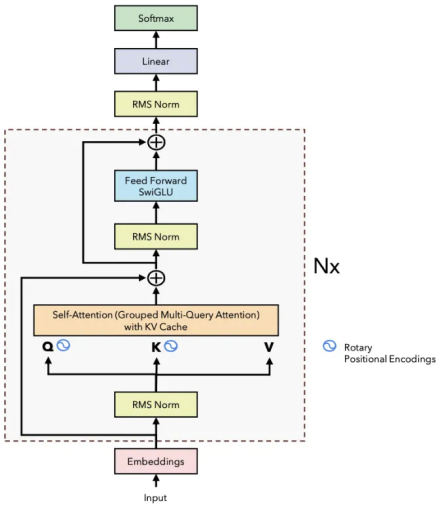
Figure 4: Grouped Query Attention, as shown in Ainslie et al. (2023).

The pre-training of LLaMA-2 is performed on *two trillion* tokens. These tokens are texts from various sources, shown in Table 5a. The sizes of the data are dated, since this table comes from the LLaMA-1 model. The authors expanded the existing datasets with more recent data. The texts are tokenized using *Byte-Pair Encoding* (BPE) algorithm from Sennrich et al. (2016), with the implementation of the Sentence-Piece of Kudo & Richardson (2018). The greatest is the Common Crawl dataset for the English language, from Wenzek et al. (2020). Furthermore, the C4 dataset of Raffel et al. (2019b), a public GitHub dataset of open-source license-approved projects, Wikipedia pages from June to August 2022, public books in the Gao et al. (2021) dataset, scientific articles of Lewkowycz et al. (2022) and Stack Exchange pages, a website with qualitative questions and answers on a wide range of knowledge domains.

Concerning the used optimizer, the AdamW optimizer of Loshchilov & Hutter (2017) is used. Regarding optimization during the backpropagation step, the Adam optimizer of Kingma & Ba (2017) proved to be superior in the field of deep learning. Adam adjusts the *learning rate* (LR) for each parameter individually, based on the first and second moments of the gradient. This means it scales the LR with an estimate of the variance of the gradients, which can be helpful when dealing with sparse gradients or noisy data, which is the case in deep learning. AdamW takes this a step further, decoupling the weight decay from the gradient updates. Weight decay is applied after the parameter is updated per step. In AdamW, this was applied to running averages of the gradients and squared gradients, which regularized parameters suboptimally. The training arguments used for this optimizer are $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay of 0.1, and gradient-clipping is implemented to prevent exploding gradients as described in Pascanu et al. (2012). Finally, a cosine LR schedule, meaning that the final LR is equal to 10% of the maximum LR, as proposed in Loshchilov & Hutter (2016). The loss function is the standard language modeling loss, the cross-entropy loss. It equals the negative log-likelihood of the correct tokens without label smoothing (Szegedy et al. (2015)), aiming to minimize the discrepancy between the predicted probability distribution and the actual distribution of tokens in training data. An

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

(a) The datasets used for pre-training LLaMA.



(b) The LLaMA model architecture, comparable to the decoder part of the transformer structure earlier provided.

Figure 5: The LLaMA model, introduced by Touvron, Lavril et al. (2023).

auxiliary loss is implemented to enhance training stability, which is $z_{loss} = 10^{-4} log_2 Z$, penalizing the SoftMax normalizer $log(Z)$.

For training, NVIDIA GPUs of type A100 were used. A total of $184,320$ GPU hours was performed for the 7 billion parameter model, $368,640$ hours for the 13 billion parameter model, and $1,720,320$ hours for the 70 billion parameter model (Touvron, Martin et al. (2023)). The price per hour for such a GPU is around \$2, thus it requires much capital.

The chat models of LLaMA-2 are extensions to the pre-trained model. Fine-tuned in a supervised fashion, with prompt-response pairs to optimize the safety and helpfulness of the model. In essence, this training is the same as the pre-training without the high computational burden. Furthermore, *Reinforcement Learning Human Feedback* (RLHF) is implemented to align the model behavior with human-preferred knowledge. By annotating a decision between two outputs of the model, a reward model is trained and human preference can be incorporated. Nevertheless, *Ghost Attention* (GAtt) was also introduced in Touvron, Martin et al. (2023), inspired by the Context Distillation of Bai et al. (2022). GAtt incorporated instructions during fine-tuning that should not be forgotten as prompts became longer. This way, the instruction was better preserved. In interactive cases, the fine-tuned model is preferred as it incorporates the prompt more efficiently.

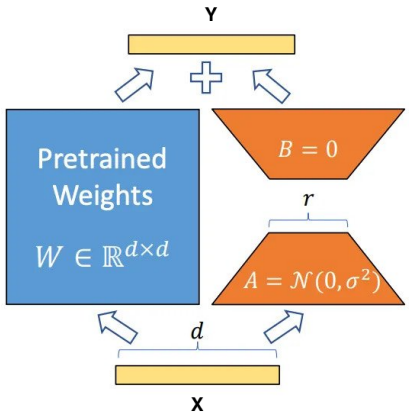
2.4 Fine-tuning methods

As these transformers grew in model size, the name *Large Language Models*, or LLMs emerged. For further reference, LLMs are defined as multi-layer stacked decoder-only transformer networks trained on vast amounts of textual data. In the context of text generation, state-of-the-art LLMs are the optimal choice as of the year 2024. As model sizes grew excessively, new solutions emerged as training all model parameters became infeasible for lower-budget teams or individuals.

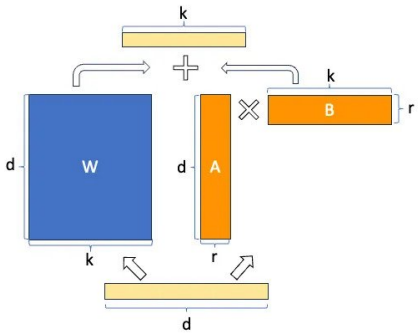
Some scientists have shown that these LLMs use only a small number of parameters for a specific input. According to Aghajanyan et al. (2020), optimizing only 200 of the hundreds of millions of parameters was just as effective as fine-tuning 90% of the parameters. This low intrinsic dimensionality, as the scientists name the phenomenon, laid the foundation for an efficient fine-tuning technique. Introduced in Hu et al. (2021), *Low-Rank Adaptation* (LoRA) is a method for fine-tuning deep networks that learns a low-rank update to the weights of the transformer, rather than fine-tuning all the weights in the model. Gradients are passed during an optimization algorithm through the fixed model weights to the adapter, which is updated according to a loss function. The linear projection of a layer is augmented with an additional factorized projection, obtained via *Singular Value Decomposition* (SVD). Then given a projection $\mathbf{XW} = \mathbf{Y}$ where $\mathbf{X} \in \mathbb{R}^{b \times h}$, $\mathbf{W} \in \mathbb{R}^{h \times o}$, LoRA resembles $\mathbf{Y} = \mathbf{XW} + s\mathbf{XAB}$. Here, $\mathbf{A} \in \mathbb{R}^{h \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times o}$, and s is a scalar. The letter b represents batch size, h the dimension of the input and o the dimension of the output after the transformation. Figure 6 illustrates the method. All bold letters represent tensors, which is a multidimensional data type for storing multiple matrices. These represent the different computational paths that arise from the different matrices such as the query, key, value, and output matrices W_q, W_k, W_v, W_o of the attention layers.

LoRA offers several advantages over traditional fine-tuning, including reduced memory usage, improved training speed, mitigating catastrophic forgetting of training data, and generalize more efficiently. Evaluated on several LLM benchmarks, LoRA outperforms traditional fine-tuning on accuracy, training speed, and memory usage. Moreover, LoRA has the potential to make LLMs more accessible to a wider range of users and applications. It is based on the principle that Aghajanyan et al. (2020) vast over-parametrized models have a low intrinsic dimension which can still learn efficiently, even when projected onto a smaller dimension.

Two expansions of LoRA are QLoRA, proposed by Dettmers et al. (2023), and AdaLoRA, introduced by Q. Zhang et al. (2023). First, the *Quantized LoRA*, or QLoRA, algorithm combines the benefits of quantization and LoRA. Quantization refers to the process of reducing the number of bits that represent information, compressing memory. Three improvements have been added to the LoRA algorithm. The first is a new data type named *4-bit Normalized Float Quantization*



(a) General structure of LoRA, where an adapter of A and B is added next to the pretrained model weights. The output of both computations is summed and returned as normal. Retrieved from Hu et al. (2021).



(b) Low-Rank solution, with an emphasis on the rank r to illustrate the decrease in trainable parameters.

Figure 6: Low-Rank Adaptation structure.

(NF4), which is based on the principle that LLM weights are normally distributed. Those weights are then normalized to have a mean of 0 and a standard deviation of 1. Those are then uniformly quantized from 16-bit into 4-bit precision by dividing them into 16 buckets, as 4-bit numbers have 16 possibilities. In Dettmers et al. (2023), it is additionally shown that the zero-shot accuracy performance of a 4-bit quantized LLaMA model was significantly improved over normal 4-bit quantization. Second, a double quantization is performed by quantizing the quantization constants as well, further reducing the average memory footprint. This is the process of quantizing the quantization constants to 8-bit precision. Thirdly is the usage of page-to-page optimizers to manage memory spikes, activated when the GPU runs out of memory. The paged memory of the computation is then allocated to the CPU RAM and paged back to GPU memory when the optimizer needs it again, as CPU RAM is larger and less expensive. The inference time is lower and due to less memory requirements, it can handle large datasets and enables people to fine-tune an LLM on a single GPU. The authors argued that, however, the quantization might introduce slight accuracy trade-offs. Moreover, the training time is likely to increase as each iteration parameters are quantized.

Secondly, Adaptive Low-Rank Adaptation, or AdaLoRA, also uses SVD to parameterize the incremental updates. This enables it to efficiently prune the singular values of unimportant updates, lowering their parameter budget without requiring exact SVD computations. A disadvantage is the complexity of determining which updates to prune, which could require additional model insights or heuristics.

Lastly, another adapter-based fine-tuning method called Infused Adapter by Inhibiting and

Amplifying Inner Activations, or $(IA)^3$. The algorithm operates by first calculating the inner activations of the LLM. These are the values that are passed between the different layers of the LLM. This exhibits which model parameters are most important. $(IA)^3$ then scales these inner activations using a set of learnable parameters. These parameters are trained during fine-tuning and allow $(IA)^3$ to learn to focus on the most important parts of the input and to suppress the less important parts. However, as argued by the authors of the method, a potential weakness is the complexity of effectively training the learnable algorithmic parameters to achieve the desired effects without overfitting or neglecting important features.

2.5 Quantization methods

For the sake of token prediction, the model parameters should be loaded onto your system. As a model with seven billion parameters, e.g. the smallest LLaMA-2, in the world of LLMs sounds small, it is still enormous. These weights can be computed in many decimals. To translate numbers into computer language, bits are used. A bit represents a 0 or a 1. Using these bits, numbers can be represented as a series of zeroes and ones. This series is commonly named a *data word*, or simply *word*. The memory necessary for one bit is more tangible when translated into bytes. One byte can store eight bits, or a data word with a length of eight. To load the model in full precision, meaning with all the decimals included, 32 bits per weight are stored in exactly 4 bytes per weight. When loading LLaMA 2 with seven billion parameters on your computer in full precision, around 28 gigabytes (GB) of VRAM is required. That number comes from multiplying 4 bytes by roughly seven billion parameters, which equals 28 billion bytes. Furthermore, as models with many parameters require more complex calculations, this could also affect inference speed. Simplifying the weights to smaller words can provide similar output in less inference time, but sacrifices precision and performance as a trade-off exists between precision and memory usage. Moreover, fine-tuning via optimization requires many additional parameters to be stored in the memory, such as gradient states. All these values are accounted for in the memory, thus reducing the overall memory footprint can have great benefits. Several quantization methods have been developed to make this procedure more viable. In essence, a quantization method reduces the word length that represents the parameter values to decrease the memory size of the large language models without significantly deteriorating their performance.

The first is the LLM.int8() method, introduced by Dettmers, Lewis, Belkada & Zettlemoyer (2022). The authors introduce the first multi-billion scale quantization procedure for transformers that prevents performance degradation. In essence, it is a two-part quantization procedure consisting of block-wise quantization and a mixed-precision decomposition. Block-wise k-bit

quantization is the procedure where weights are evenly split into chunks and transformed from a value representation that contains more information to a value representation able to hold less information. Parameter weights W are normalized to ensure that the entire range of the lower-bit data type is used. Normalization is performed on the scale of the absolute maximum of the weights. To illustrate, if data is in 32-bit precision and the quantized values are to be in the range of $[-128, 127]$ corresponding to 8-bit integers, the quantized values can be computed as

$$X_{\text{Int8}} = \text{round}\left(\frac{127}{\text{absmax}(X_{\text{FP32}})} \cdot X_{\text{FP32}}\right) = \text{round}(c_{\text{FP32}} \cdot X_{\text{FP32}}), \tag{10}$$

where the quantization constant is c_{FP32}. The issue with this method is that the presence of large magnitude values within the parameters can lead to an inefficient use of the quantization bin. This inefficiency arises because this bin, which represents a specific combination of bits, ends up being underutilized with few or no numbers quantized in those bins. This causes serious degradation in model performance due to these outlier values, thus a block-wise quantization approach is used. Each block is independently quantized as in Equation 10, all having a separate quantization constant. These transformations can also be performed simultaneously on a GPU. The mixed-precision feature comprises retaining blocks with outlier weights in 16-bit precision while quantizing the other blocks to 8-bit precision. In their research, Dettmers, Lewis, Belkada & Zettlemoyer are still able to quantize 99.9% of the parameters.

During weight optimization, gradient-based methods like stochastic gradient descent or Adam (Kingma & Ba (2017)) introduce additional parameters to the model for optimization acceleration. In Adam, two additional variables per model parameter are used. To tackle this, Dettmers, Lewis, Shleifer & Zettlemoyer (2022) introduce 8-bit optimizers, which maintain the performance of 32-bit optimizers while significantly saving memory. They achieve this through block-wise dynamic quantization, dividing input vectors into smaller blocks for independent quantization, ensuring faster optimization and precision. These 8-bit optimizers offer a practical and rapid solution for memory-constrained training of large models and come with open-source code addressing quantization, efficiency, and stability challenges in computational optimization.

More promising is the GPTQ algorithm, a concatenation of Generative Pre-trained Transformers and Post-Training Quantization. The method quantizes parameters layer-wise down to 4-bit precision. GPTQ, proposed in Frantar et al. (2023), is a more sophisticated quantization algorithm that was specifically designed for LLMs. The idea behind the algorithm is to find a compressed version of the weight, yielding a minimum mean squared error from an objective function. The calibration is evaluated on a high number of data samples to enhance the quality of quantization.

Lastly, an advanced method discovered recently is Activation-aware Weight Quantization (AWQ), introduced by J. Lin et al. (2023). In this article, the authors propose an algorithm for low-bit weight-only quantization, focusing on preserving the most important model weights while reducing quantization errors. AWQ operates by optimizing per-channel scaling based on activations rather than weights, ensuring it does not overfit to calibration sets and maintains LLMs' generalization capabilities across various domains and modalities. To elaborate on activation, it is defined as the function that processes the input into the output of a neuron in the transformer neural network. In other words, AWQ focuses on the transfer functions of the parameters rather than the weights. This way, the salient weights are distinguished from the insignificant ones. AWQ surpasses previous methods on language modeling and domain-specific benchmarks, demonstrating exceptional quantization performance for instruction-tuned and multi-modal LLMs.

3 Data

The autoregressive decoder-only transformer networks accept only textual data as input. The used datasets thus consist of a prompt, defined as a natural language text that describes the task for a model to perform. The prompt guides the output of the model, providing an instruction, request, context, and, if applicable, constraints for the output. Regarding the fine-tuning process of a model, the desired output is also added to the prompt. In this section, the datasets for the pre-specified applications are explained. These are in both cases shuffled and split into a train and test set, according to a relative 75/25 split, respectively. A detailed analysis of the data gathering process, data characteristics, and distributions of the data are presented.

3.1 Query Generation

Introduced in Section 1, the first application is the Query Generation (QG). This is applied to the Customer Support product, which powers the conversation between an AI agent and a user. In essence, the LLM is requested to return questions which can be used for a semantic search through a database. It must generate these questions, called queries, given a summarization of the conversation and a question asked by a user. The number of generated queries should not be too large, as this increases inference time and does not add any value to the output; concise and simple queries suffice.

The dataset consists of three parts. The first is the additional conversation history, covering relevant knowledge of unclear words such as pronouns. Then a question is stated, which should be either reformulated or split and reformulated, if a user question contains multiple subjects.

Table 1: Example data for the Query Generation application, with a History and Question column for input and the Queries which are to be predicted by the models.

History	Question	Queries
Conversation about the benefits of a plant-based diet	Is it good for weight loss?	Does a plant-based diet contribute to weight loss? - What are the effects of a plant-based diet on body weight?
A conversation about climate change, with a focus on rising sea levels.	How are small islands coping with climate change?	How are small islands affected by climate change? - What are the impacts of rising sea levels on island nations? - How are island communities adapting to climate change?
Discussion about sports	Who are best performing in the NBA?	Who are the highest-ranking basketball players in the NBA currently?

Lastly, the query or queries are included, resembling the desired model output. Some examples are provided in Table 1.

As real data cannot be gathered for this specific use case, data is generated using the previous three examples. The model used to generate this additional data is GPT-4 of OpenAI et al. (2024), implementing the version released on May 15, 2023. LLMs proved to be excellent data generators, especially for task-specific text data as their natural language understanding and generation is unparalleled (Meng et al. (2022)). Although this makes the data gathering process accessible, it nevertheless comes at a cost. Exhibited in Yu et al. (2023), the model can be prone to biases and less diversity. Bias is less hazardous in this application, but one should be cautious for less diversity as this could deteriorate the model performance after fine-tuning. To solve this, the attributed prompt for the generation should be carefully written. The used prompt is can be found in Appendix A. It is important to note that GPT-4 persists a maximum number of generated tokens, this prompt is placed in a loop until a sufficient amount of data is generated. According to Zhou et al. (2023), 1000 data points should suffice. The loop is repeated until 1050 data points are gathered. Subsequently, the data is cleaned as duplicates and empty rows must be removed and 990 data points remain, in the format of Table 1. The data is gathered on November 15, 2023. Following the generation and cleaning, the data is thoroughly explored to see if it suffices for the defined use case. This final touch ensures applicability to the use case. For instance, the hyphen separator is checked and the number of queries is reduced to not more than three. With regard to real conversations, questions ought to be not lengthy and too sophisticated. In Figure 7, the average question length is between 6 and 7 words with no

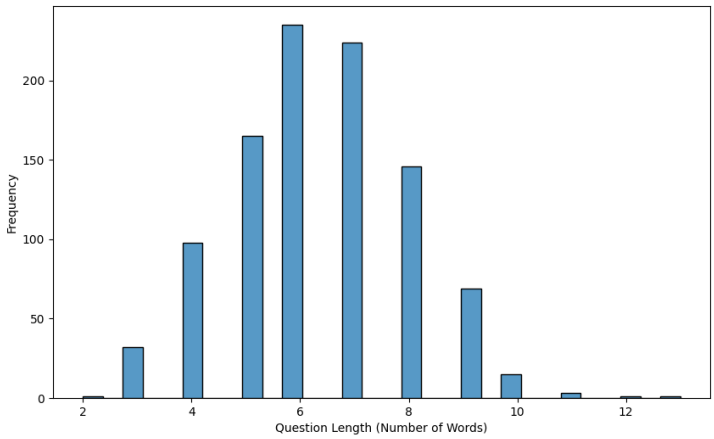


Figure 7: The distribution of the length of the questions, measured by the number of words.

question exceeding 13 words. This makes the questions suitable for the research.

Additionally, the number of generated queries should not be too high. As the queries are used to search through a database based on the semantic meaning of the query, more than three queries could be an overkill and significantly slow down the prediction time. Upon examining Figure 8, most of the data contain two queries and the maximum number of queries is limited to three.

Furthermore, to extract significant topics in the data, the *Latent Dirichlet Allocation* or LDA is implemented. The procedure of Blei et al. (2003) extracted the following ten topics that encapsulated roughly all the data points:

1. Discussing solar energy as a renewable source.

2. Talking about the history of mental health with famous figures.

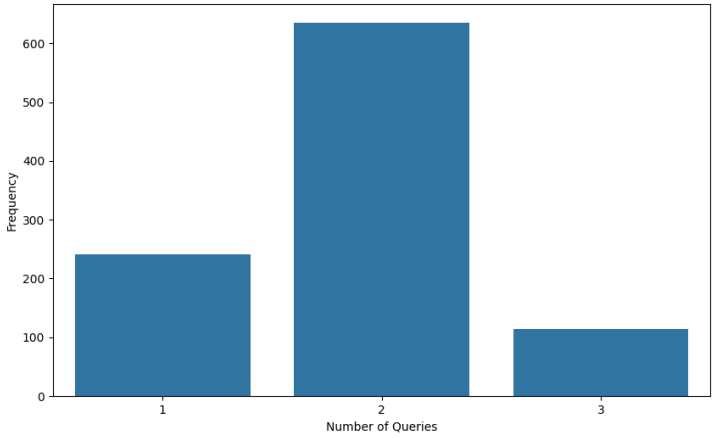


Figure 8: The distribution of the number of queries per question in the data.

3. Discussing the health benefits of a healthy diet.
4. Debating the issue of global warming and climate change.
5. Talking about Mars exploration and its implications for the internet.
6. Discussing the best approaches to weight management.
7. Talking about current trends in global fashion.
8. Discussing the best travel destinations and places to visit.
9. Discussing mental health and its importance.
10. Conversing about endangered music genres and their history worldwide.

As these topics uncover many subjects and examples, the data can be used to acquire diverse knowledge, which enhances generalizability. This can improve the model performance, increasing user satisfaction.

Thus, taking the previous three analyses into account, the dataset is suitable for the query generation application. The cleaned dataset can be found in the Github repository in Appendix **??**. At last, the dataset is shuffled and split into two sets, a training set of 75% of the data and a test set of the remaining 25%.

3.2 Named Entity Recognition

Furthermore, the second application introduced in Section 1 is the NER in the Know Your Customer product of Freeday. In summary, all the text from an ID card is extracted. The LLM should then name the values of the pre-specified entities from this text and return them in a key-value pair structure.

Regarding the dataset, Freeday holds a database of ID cards for Novum Bank, a European digital bank. Limited training data is saved, resulting in a dataset of 408 points gathered from July 2023 up to and including September 2023. The data is in the format of PDF files or images. If the file is not already in JPG format, the document is converted to JPG. Those images can be seen as a long array of zeroes and ones, or a byte array. When a document is processed, the text is extracted. This is done using Google Vision, Google's Optical Character Recognition model, or OCR model. The resulting text from this model is securely stored in the database of Freeday. The distribution of OCR text length measured in characters can be found in Figure 9. When the OCR text length is significantly longer than average, this could mean that text extraction went wrong and the text becomes counterproductive. Upon examining the figure, the number of data

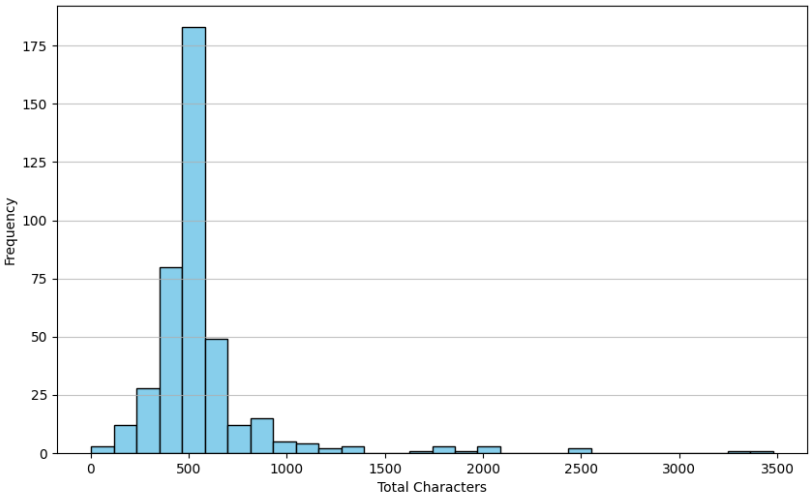


Figure 9: The distribution of the length of the OCR texts, measured in number of characters.

points that have a large deviation in characters is negligible. As disclosed in Touvron, Martin et al. (2023), the LLaMA-2 model has a context window of 4096 tokens, or roughly 16.3 thousand characters. The greatest number of characters in an OCR text does not exceed 3500 characters and the texts have a mean of 578.97 characters. This indicates that the data will be suitable for the research.

The entities to be extracted are *Given Name*, *Last Name*, *Nationality*, *Birth Date*, *Birth Place*, *Expiry Date*, *Document Type*, *Document Number* and *Issuing Country*, which are formatted as a Python dictionary format as

> {'GIVEN_NAME': XXX, 'LAST_NAME': XXX, 'NATIONALITY': XXX, 'BIRTH_DATE': XXX, 'BIRTH_PLACE': XXX, 'EXPIRY_DATE': XXX, 'DOCUMENT_TYPE': XXX, 'DOCUMENT_NUMBER': XXX, 'ISSUING_COUNTRY': XXX}

The labeling process is performed on the images, via a data labeling application of Kili Technology. The platform provides a clear and efficient interface for labeling images. The labeling process is performed by reviewers hired by Freeday. Each label is reviewed by one of the ML specialists at Freeday. As exhibited in Figure 10, not all documents contain all the entities. In some cases of missing entities, this may still be an error of the reviewers. More often, it occurs that a less common document is used as identification. Not all approved identification documents contain the nine previous entities, thus this must be taken into account during prediction.

The dataset is shuffled and split into two sets, a training set of 75% of the data and a test set of the remaining 25%.

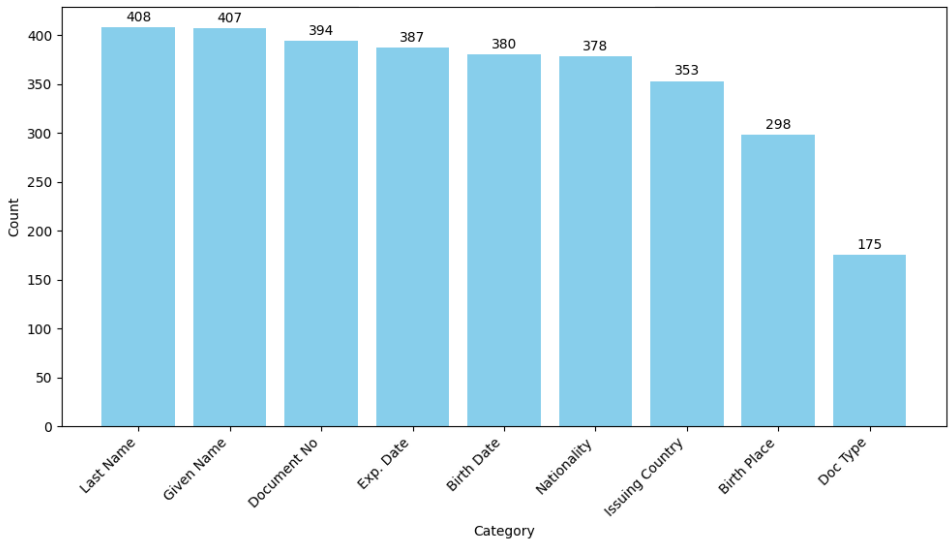


Figure 10: The frequency of the entities in the data.

4 Methodology

In this section, the methods used will be discussed. The two models that are to be fine-tuned are LLaMA-2 with 7 billion parameters and 13 billion parameters. The 70 billion parameter model is solely used for prediction, as fine-tuning this model requires such vast amount of resources it is not viable for the sake of this research.

First, an improved model attention implementation, called Flash Attention, is presented. Next, the fine-tuning method is explored in combination with the quantization to meet memory requirements to minimize training time, computational costs, and power. For these fine-tuned models and for the base models, predictions will be run according to the original computational flow inherited from the classic transformer structure to predict the desired output tokens, described in Section 2. This output is then transformed into the right format, needed for model performance evaluation. This is applied to the two test cases the LLMs will be evaluated on, both of which are explained once more thoroughly in this section along with the corresponding evaluation metrics.

4.1 Flash Attention

As earlier explained in Section 2, self-attention scales quadratically with respect to the length of the input sequence; therefore, memory-optimized attention mechanisms are required for sufficient output. Introduced in Section 1, *Flash Attention* (FA) is an advanced algorithm to optimize the efficiency and performance of the self-attention mechanism of a transformer.

For every attention head, FA adopts classical tiling techniques to minimize memory reads and

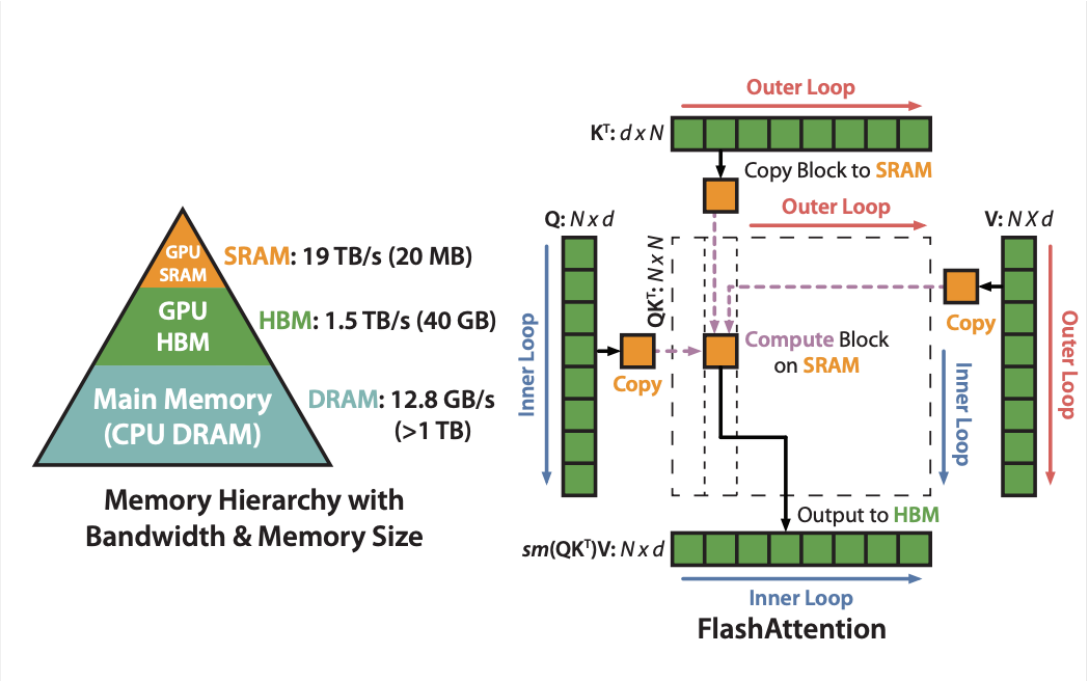


Figure 11: Flash Attention as explained in Dao et al. (2022), where the memory hierarchy of a GPU is shown on the left. On top is the smallest memory type (SRAM), with the fastest computational speed. The right shows how the input vectors are transferred to the SRAM for more rapid computations.

writes. It shuttles blocks of the Q, K, and V matrices from the *High-Based Memory* (HBM, main memory) to the rapid SRAM (static RAM, fast cache) inside a GPU. After performing attention computations of the current block, it writes back the output to the HBM. A visualization can be found in Figure 11. This memory read/write reduction yields a substantial speedup, often ranging from 2 to 4 times the original speed in most use cases. Yet this technique still only utilizes 25-40% of the theoretical maximum number of computations or operations on the required hardware. Therefore, *Flash Attention 2* (FA-2) introduces additional optimizations to tackle the memory and speed problems.

In FA-2, several optimizations are implemented in the strategy of its predecessor. First of all, the algorithms is altered to decrease the frequency of non-matrix multiplication floating point operations (non-matmul FLOPs), while ensuring the output remains unchanged. Although non-matmul FLOPs represent only a minor portion of the total FLOPs, they are significantly slower due to GPUs being optimized for matrix multiplication operations, where the throughput for matrix multiplication can be up to 16 times higher than non-matrix operations. Therefore, minimizing non-matmul FLOPs and maximizing the time spent on matrix multiplication operations is crucial for increasing both inference and fine-tuning speed. Secondly, the forward and backward pass through the model is paralellized over the GPUs. Especially during processing of long sequences, this achieves significant speed improvements. Thirdly, within each attention block, the

computational workload is distributed among different execution units (warps) of a thread block on the GPU. It now allocates Q to all warps, while ensuring that K and V remain accessible to all warps. Thereafter, each warp performs matrix multiplication to obtain a slice of QK^T. It is then simply multiplied by the shared slice of V to derive their respective output slice. All of this makes up the desired attention output. This arrangement eliminates the need for inter-warp communication. This approach is designed to minimize the need for inter-warp communication and the frequency of shared memory reads and writes, enhancing computational efficiency and speed.

Collectively, FA-2 applies tiling techniques and kernel optimizations to reduce the peak memory of attention computation and speed up attention computation by optimizing GPU usage. The increase in speed averages 5 to 9 times over the normal self-attention implementation and reaches 73% and 63% of the theoretical maximum GPU operations during forward and backward passes through the model, respectively. The only bottleneck is that to achieve these results, advanced GPUs with sufficient memory bandwidth are required. Hence, GPUs with at least the Ampere architecture can be exploited for this method.

4.2 QLoRA

For fine-tuning the LLaMA-2 model with a Flash Attention implementation, QLoRA is used due to its significant reduction of memory footprint while reaching similar fine-tuning performance increases compared to LoRA and full-finetuning in general. QLoRA involves adding a set of trainable parameters to each linear model layer while freezing the pre-trained model parameters. This reduces memory usage and computational requirements, making the model more feasible for resource-constrained devices or applications. QLoRA has one low-precision storage data type, *4-bit Floating Point* (FP4), and one computation data type, *16-bit Brain Float* (BFloat16). As stated in Kalamkar et al. (2019), computations in BFloat16 precision achieve the same results as in FP32 precision at half the memory cost. Whenever QLoRA uses weights for computation, the weights are dequantized to BFloat16, and then a matrix multiplication is performed in 16-bit. The method is visualized in Figure 12.

The initial step is to quantize the model to a lower precision to reduce the memory footprint of the model. The weights are transformed to the FP4 data type using the NF4 quantization described in Section 2. This algorithm is subject to the principles of normally distributed model weights and quantile normalization, a technique that seeks to estimate the quantile of the weights through the empirical distribution function. A fast quantile approximation algorithm from Dettmers, Lewis, Shleifer & Zettlemoyer (2022) called SRAM quantiles are used, as traditional

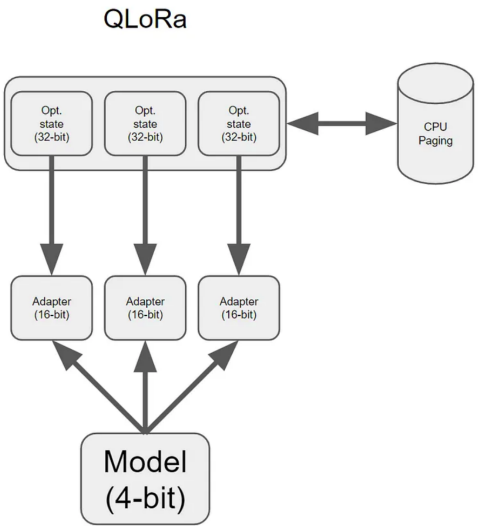


Figure 12: QLoRA visualization, where the differences in precision of the parameters are shown. The pretrained model parameters in 4-bit, the adapter weights in 16-bit and optimizers from AdamW in 32-bit, along with CPU paging. Illustration from Dettmers et al. (2023).

quantile estimation is computationally expensive. First, quantile estimation of a standard normal distribution is performed, which provides a reference for the distribution of the input weights. This is essential for adjusting the distribution of weights to fit within the 4-bit range. The $2^4 = 16$ quantiles for a theoretical $N(0, 1)$ distribution are calculated to establish a baseline for the quantization bins. Mathematically, this process is defined as

$$\begin{aligned} q_i &= \frac{1}{2}\left(Q_X\left(\frac{i}{2^k+1}\right) + Q_X\left(\frac{i+1}{2^k+1}\right)\right), \\ &= \frac{1}{2}\left(Q_X\left(\frac{i}{17}\right) + Q_X\left(\frac{i+1}{17}\right)\right), \end{aligned} \tag{11}$$

where $Q_X(\cdot)$ is the quantile function of the standard normal distribution and i is the quantile index. As this approach does not have an exact representation of the value 0 for padding tokens in the input sequences and other zero-valued elements without error, an asymmetric data type is initiated. To ensure a discrete 0 and to use all 16 bins, an asymmetric data type is created by estimating the quantiles q_i of two ranges: 8 for negative values and 9 for positive values. These sets of values for q_i are unified, where one of the two zeros that occur in both sets is to be removed. Subsequently, this data type is normalized to the range $[-1, 1]$ and blockwise quantization is performed through absolute maximum rescaling, shown in Equation 10. Hence, since quantization levels are evenly spaced to the normalized weights and the expected number of values in each quantization bin is equal, the quantized values efficiently represent the original weights.

Following the quantization of the model parameters, an additional quantization is performed.

Namely, the quantization constants are subsequently quantized. These values are stored in FP32 precision. With a block size of 64, indicating a memory requirement of $32/64 = 0.5$ bits per model parameter. Block-wise quantizing these constants to FP8 with a block size of 256 saves $8/64 + 32/(64 * 256) = 0.373$ bits per model parameter, or 0.05 bytes per parameter, as 1 bit equals 0.125 bytes in memory. For the LLaMA-2 model with 7B parameters, this saves 350 MB of memory, but 3.5 GB for the 70B parameter variant.

Subsequently, the pretrained model parameters of LLaMA-2 are frozen, and all updates are to be placed in two separate lower-dimension weight matrices, $\mathbf{L}_1$ and $\mathbf{L}_2$. The split is based on the principle that overparameterized models reside on a low intrinsic dimension, and thus that the change in model weights has a low intrinsic rank as well. By an SVD decomposition with a defined rank r, the dimensions of the two matrices $\mathbf{L}_i$ are found, both initialized. $\mathbf{L}_1$ is initialized using a random Gaussian distribution $N(0, \sigma^2)$. $\mathbf{L}_2$ is instantiated at 0. This method is applied to each linear self-attention block of the transformer network, where relationships in the input sequence are extracted. A visual representation of this is given in Figure 6. In the self-attention layers of the transformer, matrix multiplication is performed between the input sequence X and the pre-trained weight matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$, where $\mathbf{W}$ represent a tensor of all the weight matrices, d denotes the input dimension of a transformer layer, k denotes the output dimension, and r denotes the rank of the low-rank adapter of this method. QLoRA is defined in a single layer as

$$\begin{aligned} \mathbf{Y}_{\text{BF16}} &= \mathbf{X}^{\text{BF16}} \text{DDQ}\left(c_1^{\text{FP32}}, c_2^{k\text{-bit}}, \mathbf{W}^{NF4}\right) + \mathbf{X}^{\text{BF16}} \Delta \mathbf{W}^{\text{BF16}} \\ &= \mathbf{X}^{\text{BF16}} \text{DDQ}\left(c_1^{\text{FP32}}, c_2^{k\text{-bit}}, \mathbf{W}^{NF4}\right) + \mathbf{X}^{\text{BF16}} \mathbf{L}_1^{BF16} \mathbf{L}_2^{BF16}, \end{aligned} \tag{12}$$

where with $\mathbf{L}_1 \in \mathbb{R}^{d \times r}$ and $\mathbf{L}_2 \in \mathbb{R}^{r \times k}$, obtained using SVD with a predefined rank. They are both trainable parameters while $\mathbf{W}$ remains frozen. The double dequantization function $DDQ(\cdot)$ dequantizes the weights and quantization constants for accurate computations. It is formally defined as

$$\text{DDQ}\left(c_1^{\text{FP32}}, c_2^{\text{FP8}}, \mathbf{W}^{k\text{-bit}}\right) = \text{dequant}\left(\text{dequant}\left(c_1^{\text{FP32}}, c_2^{k\text{-bit}}\right), \mathbf{W}^{4\text{bit}}\right) = \mathbf{W}^{BF16}. \tag{13}$$

Here, for the weights $\mathbf{W}$, NF4 is used. c_2 indicates the quantization of the quantization constants, where a block size of 256 and the FP8 data type are used. c_1 indicates the precision quantization of the original data type, which is defined as FP32 and a block size of 64 for higher quantization performance. All computations are performed in BFloat16 precision.

Concerning the parameter updates in $\mathbf{L}_1$ and $\mathbf{L}_2$, first, the training data is tokenized as earlier described. Subsequently, the input sequences are all padded to the same length, namely,

the maximum length of LLaMA-2 of 4096 tokens. The attention mask, indicating which tokens should be taken into account during self-attention computation, is also appended to this length, but is set to false as these additional tokens need not affect attention computations. They merely serve to make matrix operations feasible on GPUs. After passing the padded input sequences through the layers, the gradients are calculated with respect to the error for the adapter weights $\frac{\delta E}{\delta \mathbf{L}_i}$. E is the error resulting from the cross-entropy loss function. For this optimization, $\frac{\delta \mathbf{X}}{\delta \mathbf{W}}$ must be computed, after dequantization from NF4 to BFloat16, since the calculations are performed in BFloat16.

The rank r can be set during the SVD method, which is commonly specified at 2, or multiples of 2. for LLMs. In Hu et al. (2021), it is however shown that increasing the trainable parameters by specifying a greater rank, the marginal fine-tuning performance diminishes. Hence, they find that a rank of 8 performs best, taking into account the computation/performance trade-off. Matrix $\mathbf{L}_1$ is set using a random Gaussian initialization and matrix $\mathbf{L}_2$ is initialized at 0. $\Delta\mathbf{W}$ in Equation 12 is scaled by $\frac{\alpha}{r}$, $0 < \alpha < r$, where α can be seen as the learning rate of the algorithm.

Lastly, the optimizer is defined. From recent literature, the most used algorithms in deep neural network training are stochastic gradient descent (SGD), Adaptive Moment estimation, either with or without weight decay regularization (Adam and AdamW), or the Evolved Sign Momentum (Lion) optimization algorithm of Chen et al. (2023). While SGD is known for its simplicity and sometimes superior generalization in certain tasks, AdamW's adaptive learning rates can lead to faster convergence in optimizing complex transformer networks. Lion is a promising method as Chen et al. achieved similar performance as Adam on autoregressive MLM optimizations and fine-tuning tasks, but has not yet achieved widespread validation in diverse scenarios. AdamW nevertheless performs sublimely with sparse gradients, which are evident in transformers as well.

In particular, when processing longer input sequences, self-attention computations scale quadratically with length. Two significant problems arise with this occurrence. The first is that the gradients for the attention become sparse. To minimize the effect of sparse attention, the AdamW optimizer is used from Loshchilov & Hutter (2017). This method decouples weight decay from the optimization steps, applying it directly to the parameters. This approach prevents the decay rate from being adapted by the optimizer, which can lead to more predictable regularization effects. Regularization is a crucial factor in handling sparse gradients and preventing overfitting of the LLM. As stated earlier, the standard deterministic cross-entropy loss function is used. The second problem that arises during the fine-tuning method is the high memory cost. This originates from two occurrences. First, there is a quadratic increase in self-attention parameters,

causing memory spikes during computation. This is resolved by Dettmers et al. (2023), where the NVIDIA Unified Memory (Li et al. (2015)) feature is utilized to page memory from GPU VRAM to CPU RAM. This prevents out-of-memory (OOM) errors. Secondly, the additional parameters of the AdamW optimization algorithm require significant extra memory. This is due to AdamW's requirement for additional memory to store both the gradients and the running averages of both the gradients and the squared gradients, along with the weight decay term adjustments. The solution to be implemented is that these optimizer state parameters are also quantized, as proposed by Dettmers, Lewis, Shleifer & Zettlemoyer (2022).

Combining the above, resembles a *paged AdamW optimizer in 8-bit precision.* Utilizing the unified memory system, an AdamW optimizer can dynamically allocate memory for these states in a 'paged' fashion. That is, when the memory capacity of the GPU is reached, typically during the processing of long input sequences, the memory pages containing the optimizer state are seamlessly offloaded to the RAM of the CPU. As the optimizer progresses to the update step and requires access to these states, the current values are directly paged back into GPU memory from the CPU. This process ensures continuous, uninterrupted computation, avoiding OOM errors that would otherwise halt the training process.

4.3 Query Generation

Considering the first task for the LLM, generated queries are evaluated. In the dataset, validation queries can be found corresponding to a user question and a summary of the conversation. The models generate these queries according to a specified prompt. This prompt should be optimized by some best practices, such as from Reynolds & McDonell (2021), as well as trial and error for a few data points to meet requirements of the output, declared in Section 3. Using QLoRA, the adapter weights are adjusted to the prompt, including the data. When evaluating the model, the queries are removed from the formed text. The model should then predict the corresponding queries with next-token prediction. The returned text is processed into an answer that can be compared to the validation queries to evaluate the model performance.

Essentially, the query must match the relevant texts in the database. In the Freeday Customer Service application, FAQs and other explanatory documents are searched in the database. These documents consist of questions and answers as title/description pairs. Hence, optimally, the queries to use for the search are short and concisely formulated questions in a general way. Words should not be complex and sentences should not be lengthy. For fine-tuning, the complete prompt including the variables *History*, *Question*, and the *Queries* are passed to the model on which the weights are to be estimated autoregressively. The prompt can be found in Appendix A.

For inference, the same instruction and the two textual variables *History* and *Question* from the test set are passed to the model. It will then predict the *Queries*. Evaluating the questions returned by the LLM requires a probabilistic approach. As LLMs are statistical predictors, output can differ in separate text generations for the same input. This does not mean that the output is wrong. If the output has the same semantic meaning as the validation, the LLM performed well. Therefore, measures are created to obtain a standardized evaluation of this output. They use the underlying context of output and compare that with the context of the test data.

One such method is the BERT-score, proposed by T. Zhang et al. (2019). *BERT-score* is, in essence, an LLM-based prediction metric that creates contextual embeddings that encompass both words and the relationships between words. The score measures semantic similarity, assessing the underlying context rather than text measures focusing only on solely the words, which is the approach in conventional metrics like BLEU (Papineni et al. (2002)) and ROUGE (C.-Y. Lin (2004)). It extracts embeddings from both candidate and reference texts using a variant of the BERT transformer of Devlin et al. (2018), called DistilBERT, proposed in Sanh et al. (2019). The choice is due to the fact that the model is smaller, faster, cheaper, and lighter. This comes at a cost in performance, but in the comparison application here that does not matter significantly to the metrics. Then, these embeddings are used to compute the *Precision*, *Recall*, and *F1 score*, which are translated to the relevancy of the candidate text context to the reference text context. According to T. Zhang et al., it has the highest correlation with human judgment of all automatic textual output evaluation metrics.

To compute the score, the texts are tokenized with the BERT tokenizer. This subword-based tokenizer returns a vector of subword IDs, $x = (x_1, ..., x_k)$ for the reference text and $\hat{x} = (\hat{x}_1, ..., \hat{x}_k)$ for the LLM generated candidates, which are transformed to a sequence of high-dimensional embeddings. Let the two vectors of embeddings be $\mathbf{x}$ and $\hat{\mathbf{x}}$, the reference vector and the candidate vector, respectively. Then, the vectors are $\mathbf{x} = (\mathbf{x_1}, ..., \mathbf{x_k})$ and $\hat{\mathbf{x}} = (\hat{\mathbf{x_1}}, ..., \hat{\mathbf{x_k}})$. These embeddings represent the meaning of the text generated for the generated text as well as the reference text. After normalizing the vectors, the cosine similarity measure reduces from $\frac{\mathbf{x}^T\hat{\mathbf{x}}}{\|\mathbf{x}^T\|\|\hat{\mathbf{x}}\|}$ to just $\mathbf{x}^T\hat{\mathbf{x}}$. The final score then matches each token of the candidate to the reference using a greedy matching algorithm, resembling the matching of each token to the most similar token in the other sentence. Subsequently, these matchings are used to compute *Precision*, *Recall*

and the *F1* as follows:

$$R_{\text{BERT}} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x_i}^T \hat{\mathbf{x}}_j, \tag{14}$$

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x_i}^T \hat{\mathbf{x}}_j, \tag{15}$$

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} - \rho_{F1}. \tag{16}$$

Here, ρ_{F1} is the penalty term. An additional penalty is passed on the F1 score to penalize the overgeneration of questions. Superfluous questions slow the model down and will likely be less relevant to the original question. Hence, for the positive difference between generated queries and validation queries, a penalty of 0.05 per generated but obsolete query is subtracted from the F1-score. The reason that only the F1 score is penalized is due to simplicity and interpretation, as *Recall* and *Precision* have distinctive interpretations.

The interpretation of these BERT scores is for precision how much each subword token in the candidate text is represented in the reference text, the recall as how much each token in the reference text is represented in the candidate text, and the F1 score as the harmonic mean of precision and recall. All three metrics range from 0 to 1, for which it is true that the closer it is to 1, the better the generated text aligns with the reference text in terms of words and relevance.

4.4 Named Entity Recognition

In the NER use case, a carefully optimized instruction and OCR text with the desired dictionary is provided to the model to adjust the adapter weights using QLoRA. The resulting model is the fine-tuned variant. For running inference, the same instruction is then passed to the model with an OCR text from the test set. Now, the dictionary is excluded and the model should predict the entities from the ID card using next-token prediction, iteratively forming the output. The prompt can be found in Appendix A.

Measurement of the performance of this NER model is generally done using regular metrics. Those entail the *Accuracy*, *Precision*, *Recall*, *F1 score* and the *Entity-Level Accuracy*. Thoroughly explained in Dalianis (2018), these give an adequate and applicable number to rate the entity extraction performance. First and most common, the *Accuracy* indicates the number of correct predictions, meaning both correctly extracted entities as correctly not-extracted entities. Thabtah et al. (2020) showed how an imbalanced dataset can severely deteriorate the power of this measure, hence this must be further evaluated. *Precision* shows how many entities identified by the model as relevant are relevant. The *Recall* indicates how many of the actual relevant items have been

successfully identified. *F1-score* combines the former two into a harmonic mean. Lastly, the *Entity-Level Accuracy* is the percentage of completely correct extracted documents of the total number of documents.

In advance, some variables are counted. *True Positives* (TP) is the number of correctly predicted entities, while *False Positives* (FP) is the number of incorrectly extracted entities. *True Negatives* (TN) are the entities not predicted that are not present in the text as well. Finally, *False Negatives* (FN) is the number of correct entities that are not extracted.

The first metric is the *Accuracy* score, measuring the correct predictions on all data. The formula is

$$Accuracy = \frac{TN + TP}{TN + FN + TP + FP} \tag{17}$$

The precision score is the percentage of predicted entities that are correct, computed as

$$Precision = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{18}$$

The recall score is the percentage of correct entities that are predicted, evaluated with the following formula

$$Recall = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{19}$$

Afterward, to combine the previous two metrics in a harmonic mean, the F1-score is computed by

$$F1 - Score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{20}$$

Lastly, an entity-level accuracy is computed. This is the percentage of documents in which all entities are extracted correctly. This is computed by dividing the number of documents with all entities extracted correctly by the total number of documents.

$$\textit{Entity-level accuracy} = \frac{\#(\text{Docs with all entities extracted correctly})}{\#(\text{Total Docs})} \tag{21}$$

5 Experimental Details

5.1 Training Setup

During fine-tuning, a set of carefully selected hyperparameters must be specified to optimize performance while efficiently managing computational resources. As there are limited resources to train, the selection is based on heuristics and best practices from previous research such as Hu et al. (2021), Dettmers et al. (2023), Phang et al. (2018), Devlin et al. (2018) and Touvron, Martin et al. (2023). The most important hyperparameter during optimization is the *learning rate*, set at

0.0002 as that is the base learning rate used by Touvron, Martin et al. (2023), striking a balance between convergence speed and stability in the learning process. Furthermore, the optimal decay rates for the first and second moments in the AdamW estimator in QLoRA are $\beta_1 = 0.9$ and $\beta_2 = 0.999$, verified by Dettmers et al.. For small high-quality datasets with this learning rate, three *epochs* over the dataset are commonly stated as best practice. Regarding QLoRA-specific hyperparameters, the *rank* r is fixed at 8. As Hu et al. (2021) demonstrates, a higher rank with additional learnable weights has minimal increases in performance while requiring significantly more computations. By the same authors, the α value is set at two times the rank, thus 16. To recall, this value represents the scaling factor of the adapter weights matrices. This is reasonable if data is of high quality, as it increases the influence of the adapter. Lastly, the *dropout* parameter is stated at 0.05 as Dettmers et al. (2023) showed the best performance during fine-tuning with this value. This indicates that, for regularization, 5% of the adapter parameters are turned off during training to avoid overfitting. The calculations are performed in *BFloat16*, further optimizing the computational efficiency. A *cross-entropy loss* function is used, standard for classification tasks due to its effectiveness in handling probabilistic output by measuring the difference between two probability distributions for a variable. This configuration underscores a sufficient balance between model performance and computational efficiency in the deep learning networks.

The NVIDIA A10G GPU, equipped with 24GB of VRAM, is a formidable processor tailored for a wide range of academic research applications, from AI and machine learning to data analytics and high-performance computing. It leverages NVIDIA's Ampere architecture, which contains Tensor Cores that are specifically designed for large matrix operations. Due to different data precision, these cores significantly increase matrix operations, almost up to twice the computational throughput compared to previous GPUs. In addition, high memory bandwidth is attained, allowing for rapid data transfer. These two characteristics allow the processor to work with FA-2, and thus only Ampere-structured or more advanced units can be used. To minimize costs, the A10G is the most viable option for this research.

5.2 Training Details

Given the setup defined in Section 4, the trainable parameter weights can be initiated for both the 7 billion and 13 billion models. By definition, the 7 billion parameter model has $6,758,404,096$ parameters, and the 13 billion parameter model has $13,047,157,760$ parameters. Corresponding to a QLoRA parameter rank of 8, the models have $19,988,480$ and $31,293,440$ trainable adapter parameters, respectively. This resembles a decrease in trainable parameters of 99.70% and

99.76%, respectively, compared to full fine-tuning.

The batch size varies with the model size, set at 5 for the 7 billion parameter models and 4 for the 13 billion parameter models, reflecting the computational demands and memory constraints associated with the complexity of the model. Approximately 75% of the VRAM was used for 7B models at this batch size, and 94% of VRAM for 13B models. This could be further optimized by testing for higher batch sizes, but that is not done in this research due to budget constraints.

Further details of the training requirements and specifications can be found in Appendix B.

5.3 Prediction

After the algorithm has converged and the QLoRA adapters are estimated, the adapters can be used in the main forward pass through the network. First, the input is multiplied with the frozen pre-trained weights and separately multiply the input with the adapter weights. Subsequently, these two outcomes are merged, indicating a simple addition of the two resulting tensors. This is done in all the hidden layers, where the hidden states are afterwards transformed into tokens following the conventional prediction method of the transformer. This procedure is described in Section 2.

Concerning the 70 billion parameter model, a cluster of 4 A10G's is used to perform predictions, as 24GB of VRAM is not enough to load the model for inference. This cluster has a shared memory of 96 GB of VRAM, allowing the model to be divided over the four units. This is performed using the *Fully Sharded Data Parallel* (FSDP) algorithm of Paszke et al. (2019). This complex technique requires careful implementation and maintenance, but otherwise, predicting with the greatest models would not be feasible.

6 Results

In this section, the results are exhibited and analyzed. Prediction inference is performed for five models. First, the LLaMA-2 7B and 13B models fine-tuned on the specific use case are used to predict the output given the prompt including the test data without the desired output. This is repeated for the three base models of LLaMA-2: 7B, 13B, and 70B. The model parameters are loaded in the same precision as the fine-tuned models, namely 8-bit. The output of the models is stripped and converted to the right format and used to compute the metrics defined in Section 4. The prompt can be found in Appendix A. The tables in this section show the performance metrics measured on the two test sets defined earlier in Section 3. The interpretations of the results are subsequently explained. Lastly, the computational requirements for this analysis are discussed.

6.1 Query Generation Evaluation

The prediction of queries is run over 248 test cases. All models are used to predict output for the test data. Table 2 presents the performance of the models on the QG task, employing BERT-score metrics for precision, recall, and F1 score, along with the average inference time in seconds. The fine-tuned LLaMA-2 with 7 billion parameters has the highest precision and recall, indicating that it has the highest efficacy in predicting the correct semantics to be used for a search. The fine-tuned LLaMA-2-13B model demonstrates superior performance in the F1 score, indicating that the 7 billion parameter variant on average has a greater penalty imposed due to the overgeneration of questions.

Interestingly, the fine-tuned models show remarkable efficiency in reducing prediction time compared to their base models. For 7B and 13B, the average inference times are 2.47 and 3.26 seconds, respectively. Compared to their base models, both realize a speedup of approximately 40%. The difference in F1 score is 0.0144 and 0.0324, thus the most gains from fine-tuning are realized in terms of speed. This is particularly valuable for applications that require rapid response times without a significant compromise in output quality, as is the case in the QG application. The user experience enhances significantly due to lower latency.

In this context, recall is how much semantic meaning of the desired output is included in the generated queries. Precision is how much of the predicted semantic meaning actually matches with the desired output. As these scores are high, all five models seem to perform well on these tasks. The fine-tuned models tend to be better at predicting the correct context. Noteworthy, is that the F1 score is penalized significantly in each model. This is due to the overgeneration of the questions, meaning that all models are inclined to generate too many queries. The fine-tuned models both obtain a penalty term of 2 percentage points more on their F1 scores than their base models. An explanation could be that more relatively more multi-query output was included in the training data after the random shuffle and split. The largest penalty term is found in the 70 billion parameter model, suggesting that an increase in model size is associated with too complex output generation. In the context of QG, simplicity and conciseness is essential, thus larger models can be redundant.

It should be noted that BERT score is a probabilistic measure, thus scores can differ slightly per evaluation. Nevertheless , the BERT score metrics are well applicable. This is due to their power of measuring and comparing semantic meaning between two texts. Even though the metric remains stochastic, the semantic meaning is exactly what needs to be compared in this context as it will be used for the semantic search.

Table 2: Performance of LLaMA-2 Models on QG, with the BERT-score values of Precision, Recall and F1 score. Average inference time is measured in seconds.

Model	Precision	Recall	F1 Score	Avg. Inf. Time (s)
FT-LLaMA-2-7B	**0.9201**	**0.8911**	0.8040	**2.47**
FT-LLaMA-2-13B	0.9190	0.8910	**0.8056**	3.26
LLaMA-2-7B	0.8677	0.8772	0.7896	4.12
LLaMA-2-13B	0.8513	0.8612	0.7732	5.46
LLaMA-2-70B	0.8837	0.8790	0.7590	11.61

6.2 Named Entity Recognition Evaluation

All the models are again used to predict output for the test set. The corresponding test set contains the OCR texts of 102 documents and is formatted as a text prompt, which can be found in Appendix A. The results of the NER application are revealed in Table 3. It is clear that the fine-tuned model with 13 billion parameters outperforms all other models. On every task-specific benchmark, from accuracy to F1 score, it scores highest, indicating its predominance in the NER application. Yet, the fine-tuned 7B model takes 3 seconds less to complete the output prediction compared to the fine-tuned 13B model, as expected when the model contains 6 billion parameters less.

To illustrate, when looking at the recall score, the model performs well in finding the existing entities. The fine-tuned models prove to be reliable to ensure that no information is missing. However, the precision scores differ roughly 25 percentage points from the recall scores, meaning it tends to pick up non-existing entities, indicating overgeneration. This trade-off is not unusual in complex tasks like entity extraction. The accuracy and an F1 score show that the fine-tuned models emphasize extracting entities on which they are trained to detect. Even though minor mistakes occur, the answering structure enables the wrongly extracted entities to be filtered out effortlessly, such that the correctly extracted entities remain.

Interestingly, the performance increase of the fine-tuned models, measured in F1 score, compared to their base models is greater than 52% for the 7 billion model and almost 38% for the 13 billion model. This indicates that the fine-tuning methods in this research significantly enhances the predictive performance of the models. Even the greatest base model variant, LLaMA-2-70B, is outperformed by the fine-tuned LLaMA-2-7B.

A significant result for the fine-tuned models is the marginal increase in inference time when comparing the two. Specifically, the FT-LLaMA-2-7B model exhibits an average inference time of 9.80 seconds, whereas the FT-LLaMA-2-13B model demonstrates a higher inference time of 12.91 seconds. This marginal difference of 3.02 seconds, or a relative difference of 31.7%, is associated with a sheer increase in performance of 3.3%.

Table 3: Performance of LLaMA-2 Models on NER, with average inference time in seconds.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**EL-Accuracy**	**Avg. Inf. Time (s)**
FT-LLaMA-2-7B	0.6330	0.6507	0.9089	0.7584	0.0098 (1 doc)	**9.80**
FT-LLaMA-2-13B	**0.6547**	**0.6753**	**0.9333**	**0.7836**	**0.0196** (2 docs)	12.91
LLaMA-2-7B	0.3486	0.3708	0.7596	0.4983	0.0000	13.87
LLaMA-2-13B	0.4107	0.4291	0.8440	0.5689	0.0000	15.04
LLaMA-2-70B	0.4728	0.4877	0.8963	0.6317	0.0000	28.90

What is not evident from the metrics, but is noteworthy is that the base models extract entities that were not requested. Examples are 'date_of_issue' and 'sex' or 'gender'. In some cases, German entity names were returned due to the German language in the document. Additionally, sometimes an incorrect answer structure was returned, making the answer unusable in an automated setting. The fine-tuned models prove to be significantly better at sticking to the right format and entities. This could also explain the shorter inference time, as the overgeneration of unrequested entities takes longer for the model to predict.

The entities predicted correctly per model are shown in Figure 13. The most accuracy gains are realized for 'Document Number', 'Document Type', 'Issuing Country', and 'Nationality'. This could have something to do with unseen structures, abbreviations, or even German words, although Touvron, Martin et al. (2023) claim that the LLaMA-2 models understand the German language as well.

Moreover, taking the OCR text into account, some entities are announced. This makes it easier for the model to recognize the corresponding text, such as given and last name. Other entities with a standard format are also well learned and recognized by the models, e.g. birth date and expiry date. Concerning the document type, the models perform the worst on this entity. When examining the test data, the entity only exists in 39 documents, of the 102 in the test set. A cause could be that the data was not as reliable as it seemed, or the dataset not large enough. Nevertheless, the entity occurs under different names, as each document contains a description in English, French, German, and sometimes Romanian. The combination of the mentioned difficulties could be the reason why the models struggle to learn and finally predict all the entities correctly, as can be deduced from the poor EL-accuracy for the models. This indicates that the application of LLMs on the NER task requires further improvements, in terms of data or model quality.

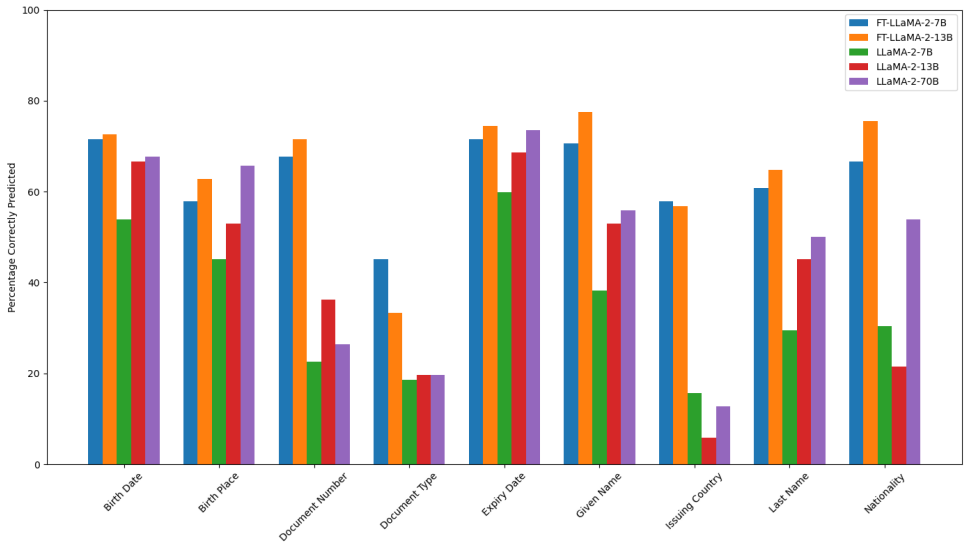


Figure 13: The prediction accuracy per entity for all five models.

6.3 Computational Requirements

Table 4 provides an insightful overview of the training time and associated costs for fine-tuning LLaMA-2 models on the two NLP tasks. Training times and costs are critical factors in the fine-tuning process, as they directly impact the feasibility and scalability of LLMs on NLP projects. The prices in this research are based on the used instances, defined in Appendix B.

Analyzing the data, a clear trend is observed. Training time increases with the complexity of the model as expected, as well as with the size of the training data as the QG dataset has 2.5 times more observations. In both applications, it holds that a parameter increase of approximately 85% more parameters comes with an increase in training time, thus also in costs, of 75%. The fact that training time increases diminishes with respect to the increase in parameters could be explained by the implemented methods. QLoRA forms additional adapters on top of the linear layers in the model. These are mostly made up of the attention layers. When model size increases, extra attention layers are added, leading to more relative efficiency from QLoRA. Moreover, the Flash Attention mechanism is also applied on more parameters as more attention layers are added to the model. The characteristics of these two methods can lead to more efficient training when scaling models in parameter size.

The results highlight the trade-offs involved in model selection. Although larger models like LLaMA-2-13B on average offer superior performance, they also entail higher training costs and longer training times. Yet, when the costs of fine-tuning do not exceed $10, they can be deemed as negligible when processing a high volume of transactions. Even though the relative costs of the model differ significantly, fine-tuning will never require such vast datasets that either price becomes exceedingly high. In general, a fine-tuning approach is realized at a low cost, making

Table 4: Training Time and Costs per model. Price is measured against the costs of the computing instance described in Section 5.

Training Job	**Training Time (s)**	**Cost ($)**
LLaMA-2-7B for QG	7041	4.97
LLaMA-2-13B for QG	12261	8.65
LLaMA-2-7B for NER	3307	2.33
LLaMA-2-13B for NER	5801	4.09

investment decisions economically viable. Selecting which model to use should depend mostly on the task-specific performance evaluation.

7 Conclusion

The extensive analysis of the fine-tuned and base LLaMA-2 models in this research using two pivotal NLP tasks, Named Entity Recognition (NER) and Query Generation (QG), has provided compelling insights into the performance, efficiency, and the economic considerations of fine-tuning. The goal of the research was to evaluate to what extent efficiency can improve for a LLaMA-2 large language model (LLM) by efficiently fine-tuning the model parameters on specific textual tasks.

The fine-tuning methodology in this research applied on the LLaMA-2 models has demonstrated the enhanced capabilities in handling specific use cases, where both fine-tuned models seem to increase equally in performance evaluation compared to their base models. Moreover, when taking the specific application into account, the 7 billion parameter model performs best on the QG task, whereas the 13-billion parameter variant shows superior performance in terms of the NER task-specific metrics. This is due to the fact that speed plays a more vital role in the QG case, similarly to the essential role of output quality in the NER case. An important result is that a fine-tuned model outperforms models 10 times larger in parameter size on both use cases.

Interestingly, the inference time in both applications was reduced. Mainly due to the more efficient attention mechanism, the prediction speed of both models has increased significantly as attention computations are performed more adequately. In an application where speed is vital, a smaller fine-tuned model should be considered as performance in task-specific measures does not decrease as much as generation times do. Still, when quality is preferred over speed, the larger fine-tuned model suits the application better. This trade-off must be considered for individual cases.

Economically considering the efficient fine-tuning of an LLM, additional training time proves to diminish for larger transformer networks. Because the additional parameters are mostly in

the attention layers, the proportion of parameters influenced by QLoRA and Flash Attention increases relatively. Hence, larger LLMs profit more from the methods in terms of training time. Nevertheless, for the two LLMs fine-tuned in this research, training costs can be negligible when handling higher volumes. Even when considering an LLM nearly twice as large, namely from 7 billion parameters to 13 billion parameters, the additional costs are insignificant.

In conclusion, the efficient fine-tuning framework used in this research offers empirical evidence of the advantage of fine-tuned LLMs in real-world tasks. Still, the choice of which fine-tuned LLaMA-2 model to use depends on the NLP task. The decision should be determined by a careful decision-making process, where the enhanced performance must be weighed against the speed decrease and thus cost increase of a larger model. The used techniques nevertheless prove to be generalizable to other open-source LLMs, as they are applicable to any transformer-based model. Yet the model selection, according to a comprehensive understanding of their performance and operational requirements, remains vital to unlock the potential abilities of the models.

For further research, it could be investigated how more data would influence the performance in both applications. As Zhou et al. (2023) found, 1000 high-quality data points are required to optimally fine-tune language models. Especially for the NER application, this could improve the model performance. Moreover, performance evaluation should not be limited to these two applications. Theoretically, generalization is possible but empirical evidence for the performance in other applications remain prevalent in claiming the generalization. In addition, simpler tasks can be solved efficiently by smaller models, but it can be investigated how complex an application can become. This could test whether the smaller models still outperform the larger variants. Also, the choice for the LLaMA-2 model can be challenged, as newer models are released of which the authors claim to be superior to the performance of LLaMA-2, such as Mistral of Jiang et al. (2023). It can be analyzed in which application which model performs better. Nevertheless, the fine-tuning methodology used in this research can be further optimized for better training and prediction results.

References

Aghajanyan, A., Zettlemoyer, L. & Gupta, S. (2020). Intrinsic dimensionality explains the effectiveness of language model fine-tuning.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F. & Sanghai, S. (2023). Gqa: Training generalized multi-query transformer models from multi-head checkpoints.

Alammar, J. (2018). *The illustrated transformer.* https://jalammar.github.io/illustrated-transformer/. (Retrieved February 6, 2024)

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016). Layer normalization.

Bahdanau, D., Cho, K. & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, *abs/1409.0473*. Retrieved from https://api.semanticscholar.org/CorpusID:11212020

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., ... Kaplan, J. (2022). Constitutional ai: Harmlessness from ai feedback.

Beltagy, I., Peters, M. E. & Cohan, A. (2020). Longformer: The long-document transformer. *CoRR*, *abs/2004.05150*. Retrieved from https://arxiv.org/abs/2004.05150

Blei, D. M., Ng, A. Y. & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993–1022.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... Amodei, D. (2020). Language models are few-shot learners.

Chen, X., Liang, C., Huang, D., Real, E., Wang, K., Liu, Y., ... Le, Q. V. (2023). Symbolic discovery of optimization algorithms.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., ... Weller, A. (2020). Rethinking attention with performers. *CoRR*, *abs/2009.14794*. Retrieved from https://arxiv.org/abs/2009.14794

Dai, A. M. & Le, Q. V. (2015). Semi-supervised sequence learning. *CoRR*, *abs/1511.01432*. Retrieved from http://arxiv.org/abs/1511.01432

Dalianis, H. (2018). *Evaluation metrics and evaluation.* Cham: Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-319-78503-5_6 doi: 10.1007/978-3-319-78503-5_6

Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A. & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness.

Dauphin, Y. N., Fan, A., Auli, M. & Grangier, D. (2016). Language modeling with gated convolutional networks. *CoRR*, *abs/1612.08083*. Retrieved from http://arxiv.org/abs/1612.08083

Dettmers, T., Lewis, M., Belkada, Y. & Zettlemoyer, L. (2022). Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Dettmers, T., Lewis, M., Shleifer, S. & Zettlemoyer, L. (2022). 8-bit optimizers via block-wise quantization.

Dettmers, T., Pagnoni, A., Holtzman, A. & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. N. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. Retrieved from `https://arxiv.org/abs/1810.04805`

Frantar, E., Ashkboos, S., Hoefler, T. & Alistarh, D. (2023). Gptq: Accurate post-training quantization for generative pre-trained transformers.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., ... Leahy, C. (2021). The pile: An 800gb dataset of diverse text for language modeling. *CoRR*, *abs/2101.00027*. Retrieved from `https://arxiv.org/abs/2101.00027`

Hochreiter, S. & Schmidhuber, J. (1997, 12). Long short-term memory. *Neural computation*, *9*, 1735-80. doi: 10.1162/neco.1997.9.8.1735

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... Sifre, L. (2022). Training compute-optimal large language models.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., ... Gelly, S. (2019, 09–15 Jun). Parameter-efficient transfer learning for nlp. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 2790–2799). PMLR. Retrieved from `https://proceedings.mlr.press/v97/houlsby19a.html`

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... Chen, W. (2021). Lora: Low-rank adaptation of large language models.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., ... Sayed, W. E. (2023). Mistral 7b.

Jurafsky, D. & Martin, J. (2008). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition* (Vol. 2).

Kalamkar, D. D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., ... Dubey, P. (2019). A study of BFLOAT16 for deep learning training. *CoRR*, *abs/1905.12322*. Retrieved from http://arxiv.org/abs/1905.12322

Kingma, D. P. & Ba, J. (2017). Adam: A method for stochastic optimization.

Kitaev, N., Kaiser, L. & Levskaya, A. (2020). Reformer: The efficient transformer. *CoRR*, *abs/2001.04451*. Retrieved from https://arxiv.org/abs/2001.04451

Kiyono, S., Kobayashi, S., Suzuki, J. & Inui, K. (2021). SHAPE: shifted absolute position embedding for transformers. *CoRR*, *abs/2109.05644*. Retrieved from https://arxiv.org/abs/2109.05644

Kudo, T. & Richardson, J. (2018, November). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. , 66–71. Retrieved from https://aclanthology.org/D18-2012 doi: 10.18653/v1/D18-2012

Lewkowycz, A., Andreassen, A. J., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V. V., ... Misra, V. (2022). Solving quantitative reasoning problems with language models. Retrieved from https://openreview.net/forum?id=IFXTZERXdM7

Li, W., Jin, G., Cui, X. & See, S. (2015). An evaluation of unified memory technology on nvidia gpus. , 1092-1098. doi: 10.1109/CCGrid.2015.105

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. , 74–81. Retrieved from https://aclanthology.org/W04-1013

Lin, J., Tang, J., Tang, H., Yang, S., Dang, X., Gan, C. & Han, S. (2023). Awq: Activation-aware weight quantization for llm compression and acceleration.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., ... Rives, A. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, *379*(6637), 1123-1130. Retrieved from https://www.science.org/doi/abs/10.1126/science.ade2574 doi: 10.1126/science.ade2574

Liutkus, A., Cífka, O., Wu, S.-L., Simsekli, U., Yang, Y.-H. & Richard, G. (2021, 18–24 Jul). Relative positional encoding for transformers with linear complexity. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 7067–7079). PMLR. Retrieved from https://proceedings.mlr.press/v139/liutkus21a.html

Loshchilov, I. & Hutter, F. (2016). SGDR: stochastic gradient descent with restarts. *CoRR*, *abs/1608.03983*. Retrieved from http://arxiv.org/abs/1608.03983

Loshchilov, I. & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, *abs/1711.05101*. Retrieved from http://arxiv.org/abs/1711.05101

Luce, R. (1959). *Individual choice behavior: A theoretical analysis*. Wiley. Retrieved from https://books.google.nl/books?id=a80DAQAAIAAJ

Luong, M., Pham, H. & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *CoRR*, *abs/1508.04025*. Retrieved from http://arxiv.org/abs/1508.04025

Meng, Y., Huang, J., Zhang, Y. & Han, J. (2022). Generating training data with language models: Towards zero-shot language understanding.

Michael Chui, R. R. A. S. K. S. A. S. L. Y., Eric Hazan & Zemmel, R. (2023). *The economic potential of generative ai: The next productivity frontier* (Tech. Rep.). McKinsey Company.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., ... Zoph, B. (2024). *Gpt-4 technical report.*

Papineni, K., Roukos, S., Ward, T. & Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. , 311–318. Retrieved from https://aclanthology.org/P02-1040 doi: 10.3115/1073083.1073135

Pascanu, R., Mikolov, T. & Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, *abs/1211.5063*. Retrieved from http://arxiv.org/abs/1211.5063

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *CoRR*, *abs/1912.01703*. Retrieved from http://arxiv.org/abs/1912.01703

Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., ... Launay, J. (2023). *The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only.*

Phang, J., Févry, T. & Bowman, S. R. (2018). Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *CoRR*, *abs/1811.01088*. Retrieved from http://arxiv.org/abs/1811.01088

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2019a). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, *abs/1910.10683*. Retrieved from http://arxiv.org/abs/1910.10683

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2019b). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, *abs/1910.10683*. Retrieved from http://arxiv.org/abs/1910.10683

Ramachandran, P., Zoph, B. & Le, Q. V. (2017). Searching for activation functions. *CoRR*, *abs/1710.05941*. Retrieved from http://arxiv.org/abs/1710.05941

Reynolds, L. & McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. Retrieved from https://doi.org/10.1145/3411763.3451760 doi: 10.1145/3411763.3451760

Sanh, V., Debut, L., Chaumond, J. & Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, *abs/1910.01108*. Retrieved from http://arxiv.org/abs/1910.01108

Sennrich, R., Haddow, B. & Birch, A. (2016, August). Neural machine translation of rare words with subword units. , 1715–1725. Retrieved from https://aclanthology.org/P16-1162 doi: 10.18653/v1/P16-1162

Shaw, P., Uszkoreit, J. & Vaswani, A. (2018, June). Self-attention with relative position representations. , 464–468. Retrieved from https://aclanthology.org/N18-2074 doi: 10.18653/v1/N18-2074

Shazeer, N. (2020). GLU variants improve transformer. *CoRR*, *abs/2002.05202*. Retrieved from https://arxiv.org/abs/2002.05202

Shazeer, N. M. (2019). Fast transformer decoding: One write-head is all you need. *ArXiv*, *abs/1911.02150*. Retrieved from https://api.semanticscholar.org/CorpusID:207880429

Su, J., Lu, Y., Pan, S., Wen, B. & Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *CoRR*, *abs/2104.09864*. Retrieved from https://arxiv.org/abs/2104.09864

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, *abs/1512.00567*. Retrieved from http://arxiv.org/abs/1512.00567

Thabtah, F., Hammoud, S., Kamalov, F. & Gonsalves, A. (2020). Data imbalance in classification: Experimental evaluation. *Information Sciences*, *513*, 429-441. Retrieved from https://www.sciencedirect.com/science/article/pii/S0020025519310497 doi: https://doi.org/10.1016/j.ins.2019.11.004

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., ... Lample, G. (2023). Llama: Open and efficient foundation language models.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. , *30*. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wang, A., Pruksachatkun, Y., Nangia, N., Singh, A., Michael, J., Hill, F., ... Bowman, S. R. (2019). Superglue: A stickier benchmark for general-purpose language understanding systems. *CoRR*, *abs/1905.00537*. Retrieved from http://arxiv.org/abs/1905.00537

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O. & Bowman, S. R. (2019). GLUE: A multi-task benchmark and analysis platform for natural language understanding. Retrieved from https://openreview.net/forum?id=rJ4km2R5t7

Wang, T., Roberts, A., Hesslow, D., Scao, T. L., Chung, H. W., Beltagy, I., ... Raffel, C. (2022). *What language model architecture and pretraining objective work best for zero-shot generalization?*

Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A. & Grave, E. (2020, May). CCNet: Extracting high quality monolingual datasets from web crawl data. , 4003–4012. Retrieved from https://aclanthology.org/2020.lrec-1.494

Yin, W., Kann, K., Yu, M. & Schütze, H. (2017). Comparative study of CNN and RNN for natural language processing. *CoRR*, *abs/1702.01923*. Retrieved from http://arxiv.org/abs/1702.01923

Yu, Y., Zhuang, Y., Zhang, J., Meng, Y., Ratner, A., Krishna, R., ... Zhang, C. (2023). Large language model as attributed training data generator: A tale of diversity and bias.

Zhang, B. & Sennrich, R. (2019). Root mean square layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2019/file/1e8a19426224ca89e83cef47f1e7f53b-Paper.pdf

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W. & Zhao, T. (2023). Adaptive budget allocation for parameter-efficient fine-tuning.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q. & Artzi, Y. (2019). Bertscore: Evaluating text generation with BERT. *CoRR*, *abs/1904.09675*. Retrieved from http://arxiv.org/abs/1904.09675

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., ... Stoica, I. (2023). *Judging llm-as-a-judge with mt-bench and chatbot arena.* (https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard)

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., ... Levy, O. (2023). Lima: Less is more for alignment.

A Used Prompts

Below is the data generation prompt for GPT-4, with the variables *max_tokens* = 7500 and *temperature* = 0.9.

> As an expert in search query generation for semantic search, your objective is to generate example data points containing concise but detailed conversation history and user queries that require concise reformulation. Reformulate these queries to enhance their suitability for semantic search. Address vague or contextually deficient queries to make them comprehensible for search purposes. Additionally, if a question requires multiple queries, split and reformulate them into distinct, focused search queries. It can occur that the user queries are missing context. In this case, the conversational history can be used to improve comprehensibility. Generate fifty data points in the format [History;Question;Queries]. When the column 'Queries' contains multiple queries in one row, separate them with a hyphen (-). The queries must be questions, ending with a question mark. Be creative and original. Examples of rows are: [A conversation about climate change and changing, with a focus on

rising sea levels and higher frequency of wildfires; How are small islands coping with climate change?; [How are small islands affected by climate change? - What are the impacts of rising sea levels on island nations? - How are island communities adapting to climate change?]], [Talk about recent advancements in artificial intelligence and machine learning using large language models; What are the ethical concerns with AI?; [What are the ethical concerns related to artificial intelligence? - What ethical issues arise from AI and machine learning? - What are the challenges in maintaining ethics in AI development?]]

Query Generation Prompt, where during inference the 'Queries' are omitted:

Instruction:
You are trained in question reformulation. You only respond with a concise and clear question. Your task is to reformulate and optimize questions based on the provided conversation history and question. The question you generate must be concise, specific, and contain the important nouns from the history and question. If the question is already clear enough, then that must be your output. If the question consists of multiple different subjects, separate the question with dashes into multiple questions. Return one to three questions. Each question may not be longer than ten words.
Input:
Conversation History: {"History"}
Question: {"Question"}
Response:
Queries: {"Queries"}

EE Prompt, where during inference the 'Labels' are omitted:

Instruction:
You are an Entity Extractor specialized in ID cards. You respond only in Python dictionary format as "entity": "value".The input is extracted Optical Character Recognition (OCR) text. The texts are mostly German. The only entities you may extract are ('GIVEN_NAME', 'LAST_NAME', 'ISSUING_COUNTRY', 'NATIONALITY', 'BIRTH_DATE', 'BIRTH_PLACE', 'EXPIRY_DATE', 'DOCUMENT_NUMBER', 'DOCUMENT_TYPE') . Answer solely in English language and only use the given input. Only return the entities that you are 100% sure about. Do not make up entity values that are not present in the input.

Input:

{"OCR text"}

Response:

{"Labels"}

B Training Environment and Details

The training of the models and the prediction will be performed in the AWS Sagemaker environment. This is an all-in-one integrated platform for deep network building, training, and deployment. It provides the best accelerated-computing instances and comes with an advanced *software development kit* (SDK) that is simple to implement. An AWS Sagemaker account is necessary for this platform and the required computing instances. In this application, additional access has to be requested to a 'G5' instance, which exploit the NVIDIA A10 GPU accelerator required to make training and inference time feasible. The instances 'ml.g5.4xlarge' and 'ml.g5.12xlarge' are used for the purpose of this investigation. The former contains one A10 GPU, and the latter implements a cluster of 4 of these GPUs. Prices are $2.539 / hour and $8.866 / hour, respectively, and are charged per minute.

Furthermore, a free HuggingFace account is required to perform the fine-tuning, which is a model hub where a variety of language models are downloadable via an API call in your Python code. Lastly, access to Meta's LLaMA-2 model is needed, free to request at Meta.

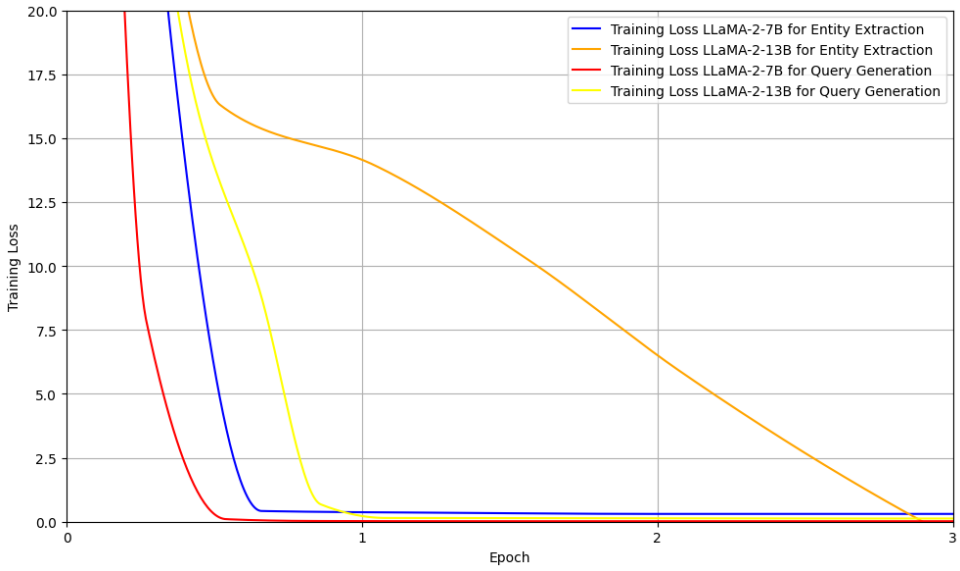


Figure 14: Training Loss during Fine-tuning over Epochs