Universidad Autónoma de Madrid

Escuela Politécnica Superior

23|24

Bachelor thesis

Language models for Question Answering problems

Enrique Saiz Oubiña

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor in Computer Science Engineering

BACHELOR THESIS

Language models for Question Answering problems

Author: Enrique Saiz Oubiña
Advisor: Manuel Antonio Sánchez-Montañés Isla

June 2024

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Enrique Saiz Oubiña
Language models for Question Answering problems

Enrique Saiz Oubiña
28701, San Sebastián de los Reyes, Madrid

PRINTED IN SPAIN

Para Sergio, q. e. p. d.

Another brick in the wall.

Pink Floyd

Acknowledgements

Thanks to my family, for they are also my friends. Thanks to my friends, for they are my family, too. Thanks to my colleagues: you have inspired and helped me, through the good and the bad, more than you will ever know.

Manuel, thank you for letting me work with the freedom that I have always loved to. AJ, I greatly appreciate your indications with LaTeX. Iván, I appreciate your Makefile as well. Google, thanks for allowing me to open 16 accounts for computations — no AI was harmed during this work.

And I am grateful. For everything given, even if it is often undeserved. For the health and strength, even if often wasted and lost. For the opportunities presented, even if they are often missed. I had the opportunity to take this work, and I did it the only way I know: my way.

Abstract

Recently, the field of Natural Language Processing (NLP) has experienced a rapid growth with the breakthrough of Large Language Models (LLMs). This evolution has been possible thanks to the advancements in computation resources, the massive availability of data, the development of open libraries and frameworks, and the apparition of a new language model architecture: the transformer. From the original encoder-decoder transformer architecture, new encoder-only and decoder-only approaches have emerged.

In this work, we explore the capabilities of state-of-the-art transformer models applied to Question Answering (QA) tasks, one of the main areas of interest of NLP. Three model families, one of each type, have been chosen: RoBERTa (encoder-only), T5 (encoder-decoder), and Phi (decoder-only). To experiment with them, we have successfully set up an open and accessible environment in Google Colab based in the HuggingFace Transformers framework. We further finetune the base models on the Machine Reading for Question Answering (MRQA) dataset in order to enhance their performance on QA.

After testing the base and finetuned models, we analyse the results to extract a series of principles and recommendations for other projects. We have found that RoBERTa, as an encoder-only model, performs the best in our extractive QA task after finetuning. Finetuning has proven to be beneficial and relatively inexpensive, compared to full pretrainings. We also tested that Finetuned Language Net (FLAN) models and community checkpoints are a good starting point for developing new model checkpoints. Besides, finetuning in decreased precision, although difficult and somewhat unstable, can yield good results. Furthermore, with decoder models, we check and find the optimal text generation strategy, which is task dependant and crucial to boost performance. Ultimately, we remark the importance of pre-processing, formatting and tokenization in the NLP pipeline.

We publicly release the code, datasets, models and training logs produced during the realization of this work.

Keywords

Artificial Intelligence, Natural Language Processing, Question Answering, Transformers, Language Models, Transfer Learning, Finetuning

Resumen

Recientemente, el campo del Procesamiento de Lenguaje Natural (NLP) ha experimentado un rápido crecimiento con el avance de los Modelos de Lenguaje Grande (LLMs). Esta evolución ha sido posible gracias al incremento de recursos computacionales, la disponibilidad masiva de datos, el desarrollo de bibliotecas y entornos abiertos, y la aparición de una nueva arquitectura de modelos de lenguaje: el transformador. A partir de la arquitectura original de transformador codificador-decodificador, han surgido nuevos enfoques de solo codificador y de solo decodificador.

En este trabajo, exploramos las capacidades de los modelos transformadores de última generación aplicados a tareas de Búsqueda de Respuestas (QA), una de las principales áreas de interés del NLP. Se han elegido tres familias de modelos, una de cada tipo: RoBERTa (solo codificador), T5 (codificador-decodificador) y Phi (solo decodificador). Para experimentar con ellos, hemos configurado con éxito un entorno abierto y accesible en Google Colab basado en la biblioteca HuggingFace Transformers. Además, afinamos los modelos base en el conjunto de datos de MRQA para mejorar su rendimiento en QA.

Después de probar tanto los modelos base como los afinados, analizamos los resultados para extraer una serie de principios y recomendaciones para otros proyectos. Hemos encontrado que RoBERTa, como modelo solo codificador, tiene el mejor rendimiento en nuestra tarea de QA extractiva después de ser afinado. La afinación ha demostrado ser beneficiosa y relativamente económica en comparación con el preentrenamiento completo de un modelo. También comprobamos que los modelos FLAN y los modelos ajustados por la comunidad son un buen punto de partida para desarrollar nuevos modelos ajustados. Además, el ajuste fino en precisión reducida, aunque difícil y algo inestable, puede arrojar buenos resultados. Por otro lado, con los modelos de solo decodificador, comprobamos y encontramos la estrategia óptima de generación de texto, que depende de la tarea y es crucial para mejorar el rendimiento. Por último, destacamos la importancia del preprocesamiento, el formateo y la tokenización en la cadena del NLP.

Publicamos de manera abierta el código, los conjuntos de datos, los modelos y los registros de entrenamiento producidos durante la realización de este trabajo.

Palabras clave

Inteligencia Artificial, Procesamiento del Lenguaje Natural, Búsqueda de Respuestas, Transformadores, Modelos de Lenguaje, Aprendizaje por Transferencia, Ajuste Fino

Table of Contents

Lists

List of equations

List of figures

List of tables

1 Introduction

1.1 Motivation

In recent times, Artificial Intelligence (AI) has become one of the main topics of discussion by the general public. What was once an area of interest solely to computer scientists and mathematicians, has now broken through as an interdisciplinary field which can be applied and merged with any productive and even creative process, further than computer science. This crescent interest has a lot to do with the latest advancements in the field, especially those related to generative AI and Natural Language Processing, which have benefited from the emergence of transformer models. Their success has been very notable in the booming of popularity that Large Language Models such as OpenAI's GPT [1] or Meta's LLaMA [2] have enjoyed.

Companies have not remained indifferent to this progress; they are being active and participant, gathering quality data to feed and train their own private, domain-specific models that satisfy and adjust to their needs [3] and improve efficiency and productivity, while also offering the public powerful models and solutions to make business and further grow their models on increasing feedback. These advancements have not gone unnoticed by governments either, which have begun to promote fair regulations and more open project developments [4].

All in all, the perfect environment for AI has been set: on the one hand, more and more people and corporations are embracing its benefits each day; on the other hand, the increasing use and feedback has become the main drive to continue improving machine learning models and solutions. Consequently, AI has the responsibility of developing and growing in a sustainable way to keep contributing to the society.

For these reasons, it made sense to focus the degree work on AI and NLP: to study and learn about the latest technologies available, the different models, their performance and their differences, as well as their biases and limitations, the opportunities they offer and their possible areas of application. Of course, NLP is very wide and encompasses a great amount of tools and tasks that is impossible to cover in a single work. So, which NLP task is more suitable to be tackled as a starter? For me, the answer to that question is Question Answering.

1.2 Scope

The project's main focus is to explore the field of Natural Language Processing from an engineering point of view, exploring the different tools and options that the current environment offers to the general public, and the possibilities that arise from them. To experiment with these tools, several state-of-the-art language models will be chosen for QA tasks; they will be fine-tuned on a selected dataset and later tested following standard evaluation techniques. The results will be then compared to draw conclusions on their performance and analyse the current state of the field. Throughout the whole project, special emphasis will be put on source-availability, efficiency and cost, as only tools available to the general public will be used. This guarantees that all the work is, not only reproducible, but also exploitable by everyone, for the better development of the NLP field.

1.3 Goals

Overall, the realization of this work focuses on the following objectives:

- Analyse the current state-of-the-art of NLP and LLMs, to choose the most suited environment and technologies for this work.
- Finetune the selected language models on QA tasks in the embraced environment. Set up a training and logging pipeline to keep track of the training as well as the computational costs associated.
- Compare the finetuned models in a common dataset and testbed. Set up a an evaluation pipeline, with rigorous metrics to measure quality performance.
- Gather results and extract principles and recommendations for other QA or NLP projects. Publish the results of this work for further use and development.

The code used during this work is available at Github [1]. Moreover, all the datasets, models and training logs developed in this work are published at the HuggingFace Hub [2].

[1] https://github.com/enriquesaou/tfg-lm-qa

[2] https://huggingface.co/collections/enriquesaou/tfg-66670a768e3ed59181581e65

2

State of the art

2.1 Natural Language Processing

Natural Language Processing is an interdisciplinary field of linguistics and computer science, whose purpose is to provide programs and computers the ability to understand and manipulate human language. NLP draws inspiration from very diverse fields, challenging today's professionals to expand their knowledge base further than computer science. Some of the most common NLP tasks are word classification, sentence classification, tokenization, Question Answering, mask filling, translation, summarization, speech recognition, text-to-speech generation, and text-to-image generation, among many others [5]. Basically, any task that encompasses understanding human language.

Natural Language Processing was born in the 1950s as a common ground for linguistics and artificial intelligence. The computer completed language tasks by applying a set of defined rules to a certain input sequence. These sets of rules could be of varying size and nature depending on the scope of the task; context-free grammars [6] have been commonly employed to represent the syntax of programming languages, while regular expression syntax, defined in 1956 [7], was often used to search and identify text patterns. Soon, new language tools based on these syntaxes appeared: lexers, which tokenize a given text, and parsers, which are in charge of validating token sequences. Formal grammars of symbolic NLP proved to be (and still are) useful for restrictive languages such as programming languages, but failed to assess the size and ambiguity inherent to natural human languages efficiently.

In the late 1980s, the sets of rules of symbolic processing began to be overperformed by natural language machine learning solutions. This probabilistic approach, which had been doubted in the past [8], was possible thanks to the increase of computational power, reflected in Moore's law [9], and to the larger availability of language data for training the machine learning models. The growth in data was particularly influenced by the birth and evolution of the web. The N-gram word model [10] became the best statistical algorithm; N-grams are sequences of N number of words which are more likely to appear together in the language.

Finally, statistical NLP was able to capture meaning and semantics in an efficient way. However, in 2003, the N-gram model was superseeded by a neural probabilistic language model based on a

Feedforward Neural Network (FNN) [11]. Neural NLP had taken over the field, and for the first time in 2010 a Recurrent Neural Network (RNN) was used for a language model [12]. Deep learning techniques started to be applied to process language, as well as word embeddings to capture word semantics. In 2017, NLP experimented a major breakthrough with the introduction of transformers in Google's paper *Attention Is All You Need* [13]. Since then, NLP has evolved by leaps and bounds, with several transformer model architectures revolutionizing the field, such as OpenAI's GPT-2, whose capabilities were deemed both powerful and dangerous at the time [14].

2.1.1 Question Answering

Question Answering is a NLP task that consists of retrieving or generating an answer to a question. The main QA variants are:

- **Extractive QA**: the answer is extracted from a context. Note that QA can be either textual or visual QA, depending on the nature of the context (if it is a video or image rather than text). However, the former is the most studied and relevant nowadays.
- **Generative QA**: the answer is generated freely based on the context. In reality, it is a task of text generation, but requires understanding and reasoning.
- **Closed generative QA (closed-book)**: the answer is generated freely, without given context. It is a task of text generation as well.

Moreover, one can classify QA depending on their scope:

- **Open-domain**: unrestricted questions that may cover any topic, relying only on general knowledge.
- **Closed-domain**: questions restricted to specific a domain (medicine, legal, corporate, etc.), which exploit the domain knowledge.

The applications of QA are plentiful; for example, technical support, shopping advice, fact checking, customer service, among many others.

2.2 Transformers

Transformers are deep learning architectures, introduced in 2017 in a research focused on translation models [13]. They soon outperformed RNNs and have completely revolutionized the fields of, not only Natural Language Processing, but also audio, computer vision and multimodal processing.

The success of the transformer architecture is largely thanks to the introduction of self-attention layers merged with the multi-head mechanism. This approach allows the model to weigh the importance of each input token, enabling the transformer to consider efficiently the context of each one of them in relation to other tokens which may appear far away in the input sequence.

2.2.1 Architecture

Every transformer shares three basic components: a tokenizer, to convert text into tokens; an embedding layer, to convert tokens and positional encodings of the tokens into vector representations (tensors); and transformer layers, to apply transformations on the tensors. Let us explore in detail these components with the architecture of the original transformer, seen in figure 2.1.

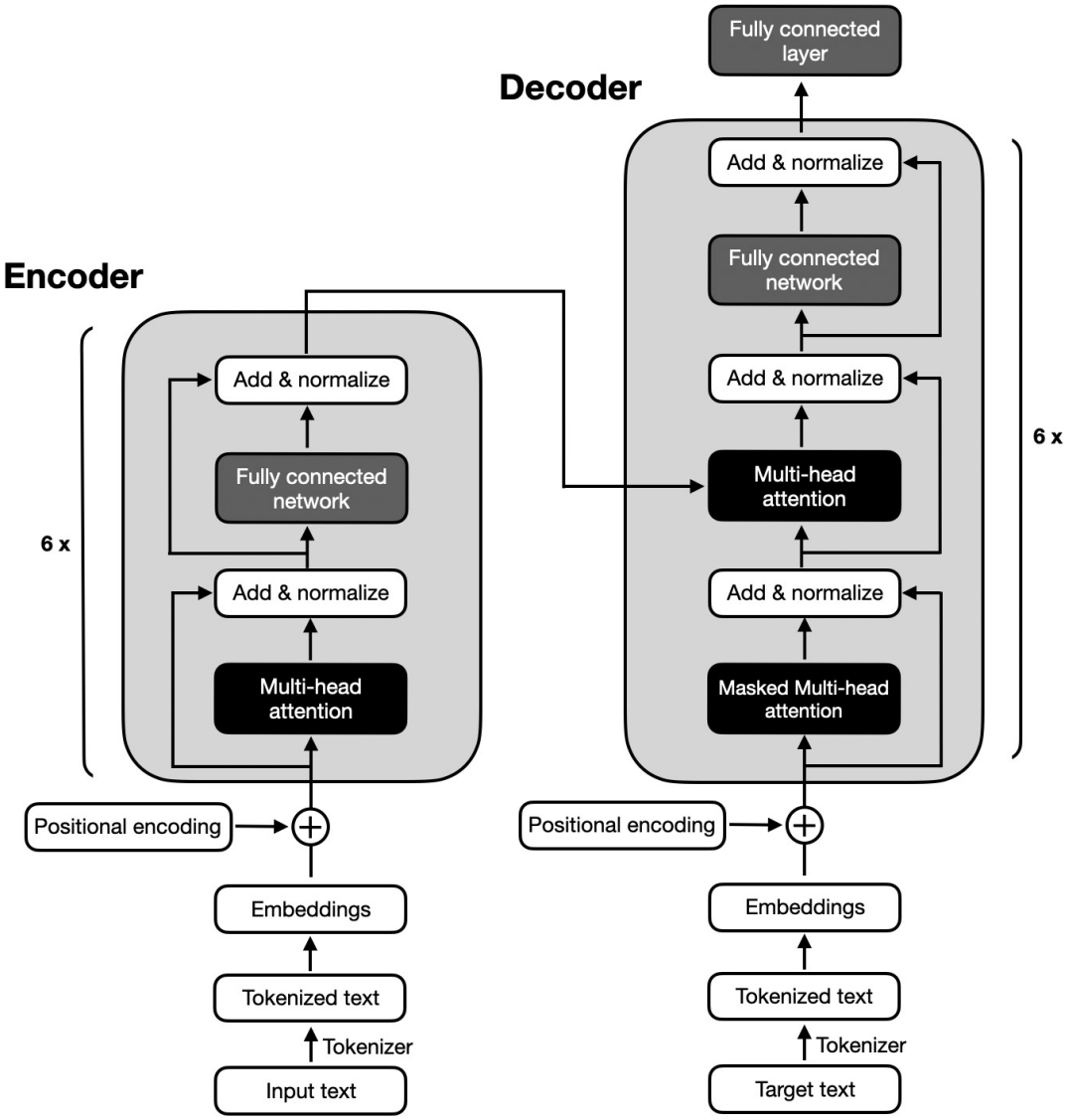


Figure 2.1: The original transformer architecture. The encoder part is to the left and the decoder to the right, both with $N = 6$ layers. *Extracted from [15].*

The original transformer architecture was based on an encoder-decoder structure, where the encoder processes the input sequence and the decoder generates the output sequence. The encoder maps an input sequence of symbol representations $x = (x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. With the z, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one at a time, using previously generated symbols as part of the input for the next. Each

encoder and decoder layer contains multiple self-attention heads along with a Feedforward Neural Network. In this case, there are $N = 6$ layers of each, and the dimension of the input sequence x is $d_{model} = 512$. The model does not inherently consider the positions in the sequence, as other previous architectures had achieved with recurrence, for example; to provide the information about the position of the tokens in the sequence, positional encodings of dimension d_{model} are added to the input embeddings of both encoder and decoder. Furthermore, to generate the embeddings, the input and the output tokens are converted to vector representations of dimension d_{model}. As a final step to translate decoder outputs to predicted next-token probabilities, the model applies a linear transformation and the softmax function.

What is attention? The attention mechanism relates different tokens and positions of an input sequence to compute a representation of the sequence. It takes as an input three vectors: a query Q and a set of key K value V pairs. Q and K are of dimension d_k while V has dimension d_v — the weight vectors W have dimension $d_{model} \times d_k$ and $d_{model} \times d_v$ respectively, as depicted in table 2.1.

Object	Shape	Original values
q_i, k_i	d_k	$(64,)$
v_i	d_v	$(64,)$
x_i, z_i	d_{model}	$(512,)$
W^Q, W^K	$d_{model} \times d_k$	$(512, 64)$
W^V	$d_{model} \times d_v$	$(512, 64)$

Table 2.1: Values of the shapes in the original transformer architecture. q_i, k_i and v_i are the query, key and value vectors of each head; W^Q, W^K and W^V the weight vectors; x_i the input; and z_i the output.

The output is computed as the weighted sum of the values; the weight of each value can be obtained as the scaled dot-product of Q and K (normalized with the softmax function), following the formula 2.1. Graphically, it can be visualized in 2.2.

$$Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V \tag{2.1}$$

Moreover, the encoder has self-attention layers; in these layers, the keys, values and queries have the same origin, that is, the output of the previous encoder layer, so each position in the encoder can attend to all positions in the previous layer of the encoder. This means that every position has an attention weight with every other position: for N positions, N^2 computations are required to calculate the respective attention weights. In the self-attention layers of the decoder, to prevent a certain position accessing the following positions (to only allow accessing previous and present positions), the scaled dot-product attention incorporates a mask to hide them in the softmax's input, depicted in figure 2.3(a).

Why attention? Since Q, K and V are calculated independently, the algorithm is parallelizable; this

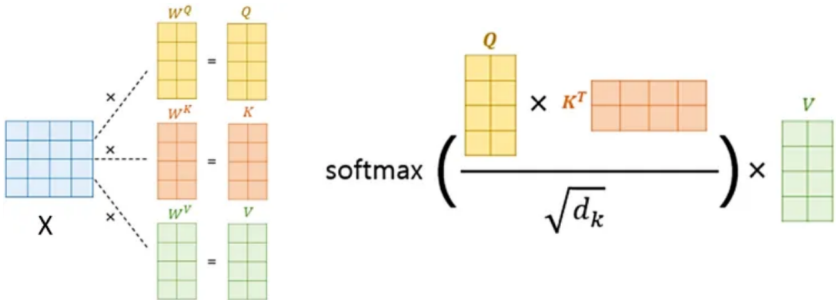


Figure 2.2: Graphical interpretation of scaled dot-product attention. Q, K and V are obtained by multiplying input X by their respective weight vectors W. *Extracted from [16].*

was a major drawback in RNN architectures, where computations had to be sequential by nature (they rely on hidden states where the state h_t corresponding to position t depends on the state h_{t-1}). Besides, in such previous architectures, when relating far away positions, the number of operations grew with the distance between positions in a linear or logarithmic fashion, depending on the implementation, which difficulted learning long-range dependencies. In transformers, however, this complexity is constant. The ability to learn these dependencies is given by the length of the paths that information has to traverse: the shorter, the easier it is to learn them. As depicted in table 2.2, self-attention layers require a constant number of sequential operations, while the RNN appraoch requires $O(n)$ sequential operations.

Layer type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$

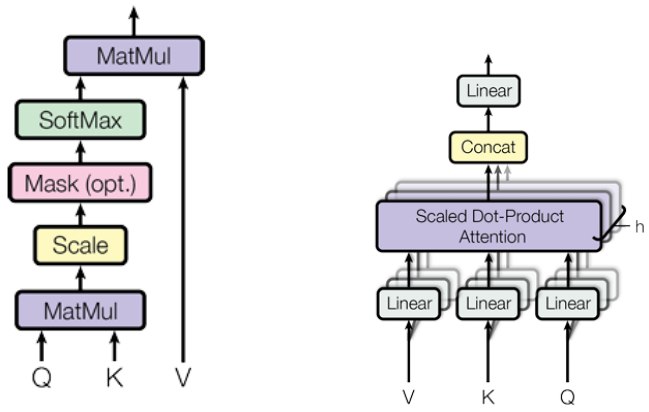
Table 2.2: Comparison in complexity among different layer types. n is the sequence length and d is the representation dimension.

Other benefit of attention is interpretability, as it is possible to analyse the model's attention layers, and the different behaviors and tasks that each head from the multi-head mechanism performs.

What is multi-head? As mentioned, the transformer does not work with a single head, but uses a multi-head attention mechanism. The embedding vectors for the input and output sequences get logically split across different heads. As a result, instead of performing a single attention function with d_{model} dimensional Q, K and V, they are projected h times with learned linear projections to d_k, d_k and d_v dimensions. The attention is then computed in parallel, the outputs are concatenated and finally projected to get the final values, as seen in figure 2.3(b).

Consequently, the multi-head attention equation is shown in 2.2.

$$\begin{aligned} MultiHead(Q,K,V) &= Concat(head_1,...,head_h) \cdot W^O \\ where\ head_i &= Attention(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \end{aligned} \tag{2.2}$$



(a) Scaled dot-product attention with optional masking.

(b) Multi-head attention mechanism with h heads.

Figure 2.3: Graphical interpretation of the multi-head attention mechanism. Figure 2.3(a) shows the attention computation procedure, and figure presents 2.3(b) the multi-head mechanism. Each head performs its own attention calculations, which are later concatenated. *Extracted from [13].*

Why multi-head? Multi-head allows separate sections of the embedding to learn different meanings, capturing richer interpretations of the embeddings. Each head could develop a different representation of the information relating it to different positions, and these understandings are then combined. With a single head, this behavior would be lost after averaging the weights.

2.2.2 General architectures

From the original encoder-decoder transformer, three main different approaches to the transformer model have been taken: encoder-decoder, encoder-only, and decoder-only. With time, several families of models have emerged, as seen in figure 2.4.

In encoder-decoder architectures (also known as sequence-to-sequence) like the original transformer, the encoder processes the input sequence and passes the embeddings to the decoder, which generates the output sequence. To achieve this, the encoder accesses all the tokens in the input sequence, while the decoder can only access the previous tokens in the sequence. Encoder-decoder models are best at generating an output sequence which requires understanding a given input; this is the case in text translation, summarization and generative QA. Examples of architectures based on this approach are BART [17] and T5 [18].

The encoder-only architectures, known as auto-encoding, use an encoder to acquire an understanding from the input, accessing all the tokens in the input sequence (thus called bidirectional transformers) and outputting its representation (also called embedding). Therefore, they are best at word classification, sentence classification, and extractive QA. Examples of encoder architectures are BERT [19] and RoBERTa [20].

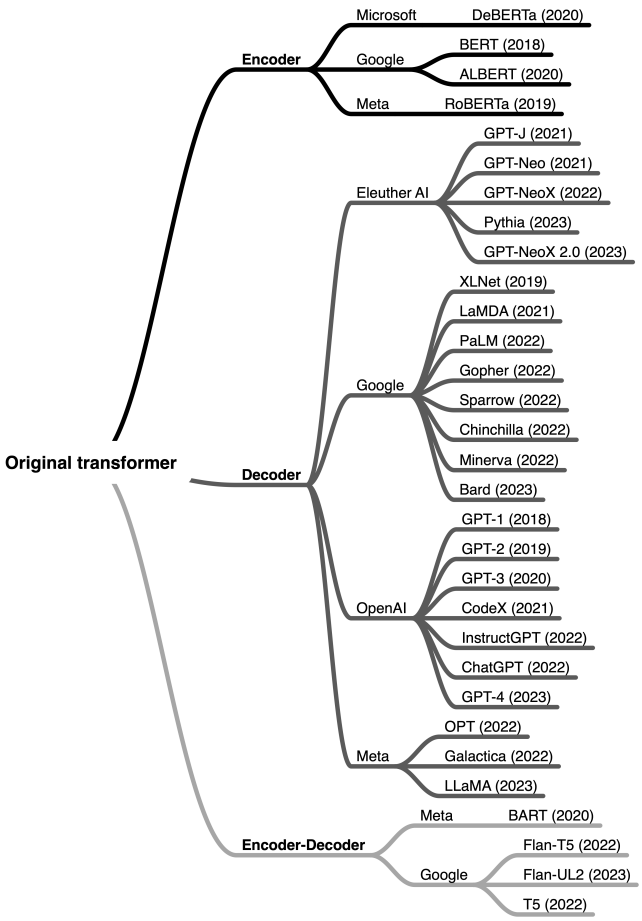


Figure 2.4: Popular transformer families classified by architecture. Main architectures are encoder-only, decoder-only, and encoder-decoder. *Extracted from [15].*

In recent times, decoder-only models (auto-regressive) have experienced a high degree of success thanks to the breakthroughs in text generation. As explained earlier, the decoder, using the attention mechanism to mask future positions, only accesses the previous tokens in the sequence, and takes them as an input to predict the next output token, one at a time. Popular architectures are LLMs such as OpenAI's GPT [1] and Meta's LLaMA [2], though some smaller and more accessible models like Microsoft's Phi [21] are gaining attention too.

Transformers have opened the doors to other domains beyond NLP, and currently the scope is set on multimodal learning transformers, capable of analysing text, audio and images to develop a robust representation of the real world.

2.2.3 Transfer Learning

Transfer learning is a machine learning technique that involves leveraging the knowledge learned from a task and adapt it to a similar task, in order to boost performance and learning efficiency. This approach can be easily applied to source-available transformer models, reducing the amount of training data and computational power required by intense, ground up trainings, and therefore speeding up the learning process [22].

Transfer learning offers the possibility to take a pretrained, general-purpose language model (such as foundation models) as a starting point, which has undergone expensive trainings on large amounts of data, to further develop and finetune it to a specific dataset and task. This will generate a model checkpoint: a specific set of weights loaded in the architecture that define the model behaviour and performance. The process of finetuning is less resource intensive and much faster, since the pretrained model has already undergone lengthy trainings.

This technique has greatly contributed to the development of the NLP field, and encourages researchers and engineers to further finetune model checkpoints using diverse approaches on specific tasks [18]. Besides, transfer learning has helped to reduce, not only training time and costs, but also the environmental impact of the process.

2.3 Large Language Models

Large Language Models are transformer-based models highly capable of understanding and generating language. Their sizes range from millions to billions of parameters, and they are often trained in massive corpus of data. Because they are versatile and have been pretrained on a wide range of tasks, LLMs can be further adapted to a certain domain or task. As a result, the usual training process, depicted in figure 2.5, involves a pretraining phase, then a domain adaptation through techniques such as in-context learning, finetuning and Retrieval Augmented Generation (RAG); and finally Reinforcement

Learning from Human Feedback (RLHF).

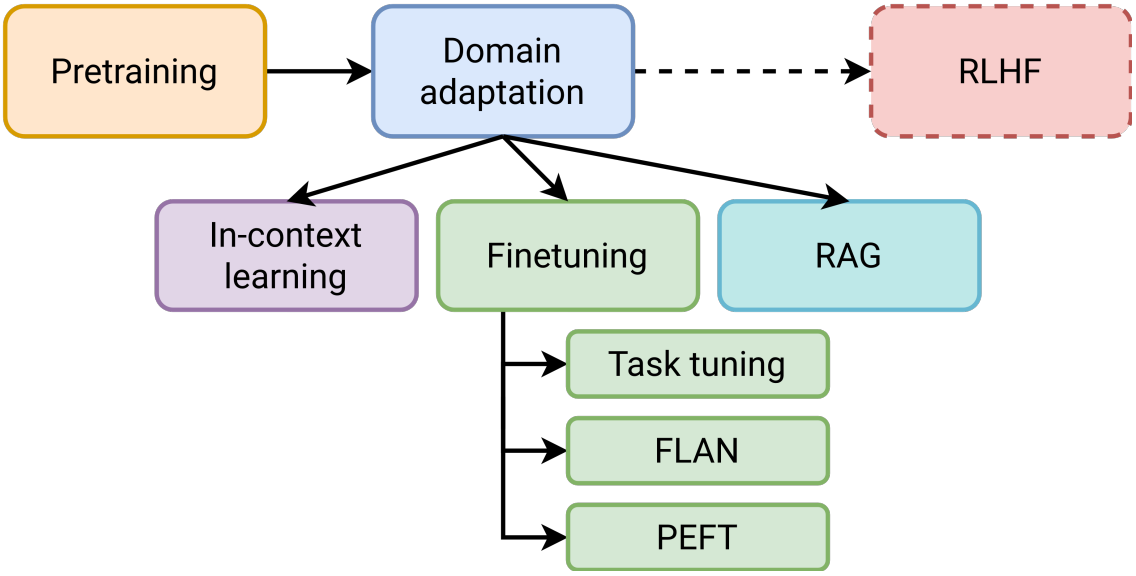


Figure 2.5: LLMs training methods. Main pipeline includes pretraining, domain adaptation through in-context learning or finetuning, and usually RLHF.

2.3.1 Training LLMs

Pretraining

Pretraining is the first and most necessary step that a base model undertakes before becoming fully functional. It provides basic understanding about language, without teaching specific knowledge. The pretrained model can be then tuned by taking advantage of these acquired linguistics, or alternatively it may be adjusted to perform general instruction tasks [23]. The process is shown in figure 2.6. It is important to note that the quality and quantity of the data makes a big difference when training models.

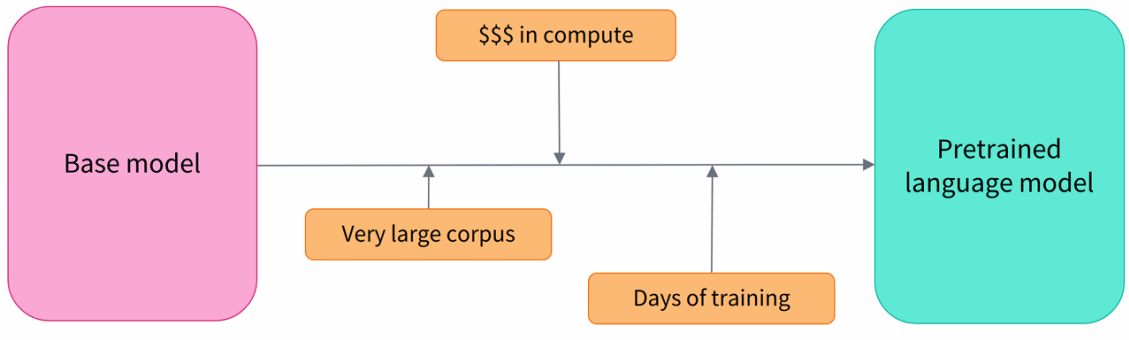


Figure 2.6: Pretraining process of base LLMs. It is an expensive and intensive process. *Extracted from [23].*

Finetuning

The second step in LLMs training is finetuning: using few, domain-specific data, the model is adjusted to perform certain tasks or areas of knowledge. This process, depicted in figure 2.7, is less costly and resource intensive than a pretraining, thus being more accessible and reproducible. Of course, finetuning can only be applied to source-available models, whose layer weights can be tuned.

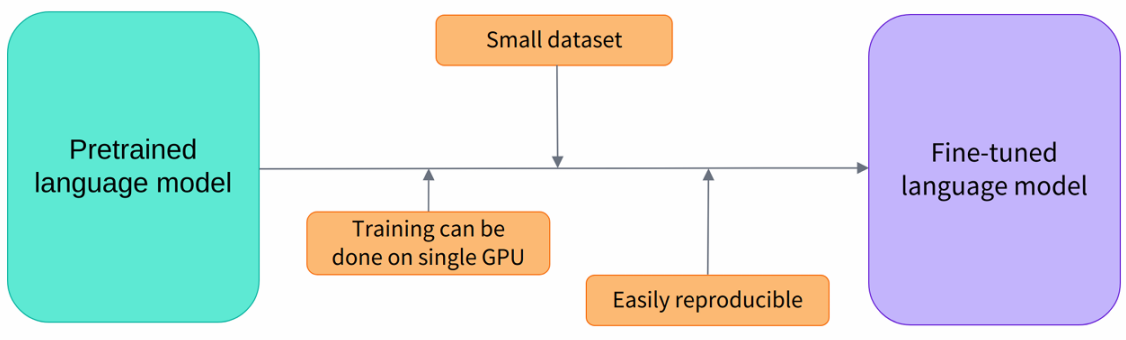


Figure 2.7: Finetuning process of pretrained LLMs. Finetuning is more accesible and manageable than a full pretraining. *Extracted from [23]*

In-context learning

In-context learning, also known as prompt engineering, is a straightforward training method usually recommended for situations where direct access to the model is limited. It manages to spare the hassle and costs inherent to finetuning by harnessing the powerful capabilities of transformer-based LLMs to weight tokens within their context. Thanks to the attention mechanism, it is possible to teach models solely with prompts to follow instructions in order to solve a task in a certain way, without modifying layer weights. This approach is of course limited to the context window, that is, the amount of tokens the transformer can take as input. As investigation and technology advances, LLMs specifications continue to improve, and latest models such as GPT-4 Turbo support up to 128000 input tokens [24].

However, large contexts may degrade the model's performance in certain situations, as the computations are more demanding and short range token relations lose importance, whereas small contexts may fail to understand long range relations. As a result, special attention is being paid in this area of research to understand the trade-offs of different context lengths in prompt engineering techniques.

RAG

Retrieval Augmented Generation consists of merging a language model together with an external indexed knowledge base, which can be frequently updated, releasing the need of finetuning the model too often. Relevant information and documents are retrieved from the database, and then this evidence is given as an input to the model. As a result, RAG usually reduces hallucinations in the model, though

this access implies higher latency. RAG is most suitable for knowledge-intensive tasks, where frequent finetuning is not viable or just would not be complete.

RLHF

Reinforcement Learning from Human Feedback involves the model making predictions or decisions which humans then evaluate, providing the appropriate feedback. The model adjusts its behavior based on the feedback, aligning itself to human judgement and values (policy). A reward model can be trained on human preferences and used to calculate the reward for the outputs. This process is expensive, as the preference data must be specially high-quality.

2.3.2 Notable LLMs

In recent years, thousands of different LLMs have been developed; some of the most relevant models, whose specifications are shown in table 2.3, will be briefly discussed and analysed.

Model	Date	Parameter size	Training compute (FLOP)	Accesibility
GPT	2018	117M	1.8e+19	Open source
BERT	2018	340M	2.9e+20	Open source
RoBERTa	2019	355M	4.2e+21	Open source
T5-Small	2019	60.5M	1.9e+19	Open source
T5-Base	2019	223M	6.9e+19	Open source
T5-Large	2019	738M	2.3e+20	Open source
GPT-2	2019	1.5B	4.3e+21	Open source
GPT-3	2020	175B	3.1e+23	API access
GPT-4	2023	Undisclosed	2.1e+25	API access
Mistral	2023	7B	Undisclosed	Open source
Mixtral	2023	46.7B	Undisclosed	Open source
Phi-2	2023	2.7B	2.3e+22	Open source
Phi-3	2024	3.8B	7.5e+22	Open source
Gemma-2B	2024	2.51B	2.4e+22	Restricted use
Gemma-7B	2024	7.75B	2.5e+23	Restricted use

Table 2.3: Comparison of notable LLMs. Training compute is given in total FLoating Point Operations (FLOP). *Data extracted from [25].*

BERT and RoBERTa

BERT, which stands for Bidirectional Encoder Representations from Transformers, is an encoder-only model introduced in 2018 by Google researchers. Due to the nature of its architecture, BERT is ex-

tremely efficient at language understanding, though it is not suitable for text generation. By 2020, BERT was used in almost every Google Search English query [26].

In 2019, Facebook researchers developed RoBERTa (from Robustly optimized BERT approach) based on BERT. It maintains the same architecture, but with several training and optimization improvements, thus creating a more performant model.

T5

T5 is a series of encoder-decoder models developed by Google, introduced in 2019 [18]. At the time, T5 achieved state-of-the-art results on most benchmarks covering summarization, QA and text classification, thanks to its ability to treat every NLP task as a unified text-to-text problem. This means that for every task, the model approach is the same: generating a piece of text from the given input text.

Currently, there are several T5 models available, including Small, Base, Large, 3B and 7B parameters variants, and FLAN trained variants, which are specially finetuned on instructions [27].

GPT

GPT is a family of decoder based models created by OpenAI. It has become the most popular generative model to date thanks to the robust API access provided through ChatGPT. GPT family, which stands for Generative Pretrained Transformers, is specially designed to generate human-alike content. Most notable models are GPT (2018), GPT-2 (2019), GPT-3 (2020) and GPT-4 (2023).

Mistral and Mixtral

Mistral language decoder-only model was introduced by the French company Mistral AI in 2023 [28], making great emphasis on the importance of open source models. While the tendency in the field is to evolve towards larger number of model parameters, Mistral aimed to become a relatively small and accessible model, with high performance yet practical and cheap to deploy in production environments. As a result, with barely 7B parameters, Mistral beats GPT-3's 175B parameters in most common sense and reasoning benchmarks [29].

In late 2023, Mixtral was introduced: an open source and one of the most powerful and versatile language models to date [30]. Mixtral relies on the Mixture of Experts (MoE) layer, depicted in figure 2.8. Each token input to the model is routed to 2 of the 8 model's experts, and the output is weighted summed together. Each expert is a Mistral 7B model, specialised in a group of tokens or a particular concept.

This technique scales the number of parameters while managing cost and efficiency, as the model only uses a fraction (12.9B out of 56B) of the total set of parameters per token. Therefore, MoE solves the scalability issues of LLMs, thus it is expected to be adopted by eminent NLP companies soon;

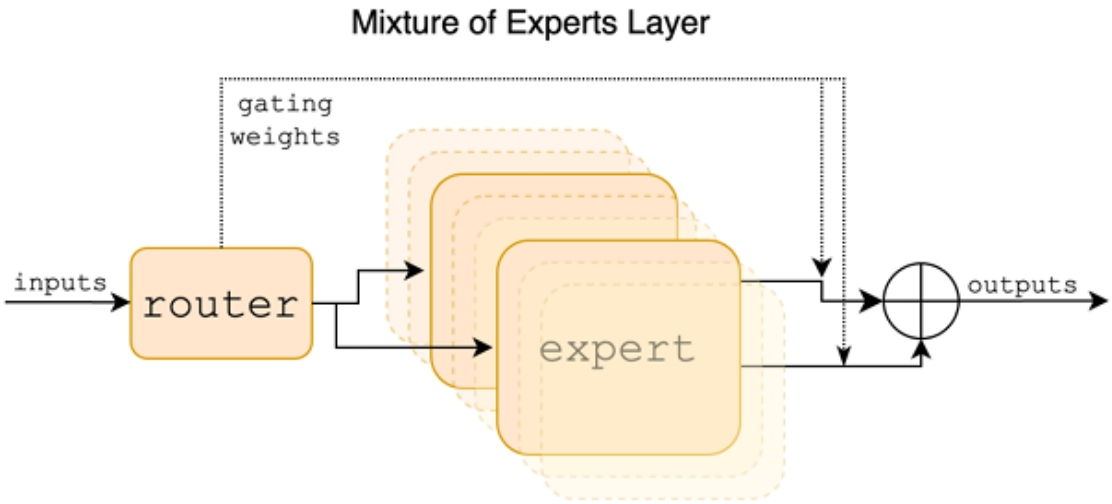


Figure 2.8: Mixture of Experts layer. Router sends inputs to 2 out of 8 experts, and the outputs are then merged. *Extracted from [30].*

OpenAI's GPT-4 is rumoured to be based on MoE design, too [31].

Phi and Gemma

In an attempt to fight the scalability trend issue in LLMs, companies have begun to design smaller, more efficient and lightweight models, which can be even run in consumer hardware. Two examples are Microsoft's Phi [21] and Google's Gemma [32] decoder based language models. Phi is open source, while Gemma is offered as an open access model with restricted use. Both are relatively small size, with Phi-2 and Phi-3 having around 3B parameters, and Gemma offering both 2B and 7B parameters versions, thus being more manageable than larger models while still giving decent performance. The key to this performance is primarily the quality of the training data: in the case of Phi, textbook-alike synthetic data was employed [21].

One of the main advantages of these small models is that they can be ran locally on mobile phones; for example, Phi-3 can be quantized to occupy less that 2GB of memory and process 12 tokens per second on an iPhone 14 [33].

2.3.3 Impact and cost

Perhaps the greatest drawbacks of large transformer models are the monumental costs and the environmental impact that they represent. State of the art models, such as GPT-4, have undergone trainings that reach the $100 million mark [34]. For reference, the original transformer model cost was around $1000 [13], as depicted in figure 2.9. The cost is increasing every year, as new, larger models are being trained; in figure 2.10, the tendency to devote greater computing power to model training is definitely

notable. Universities, which had always spear-headed AI and NLP research, are beginning to fall behind in model development as they simply cannot match the power and resources of big corporations.

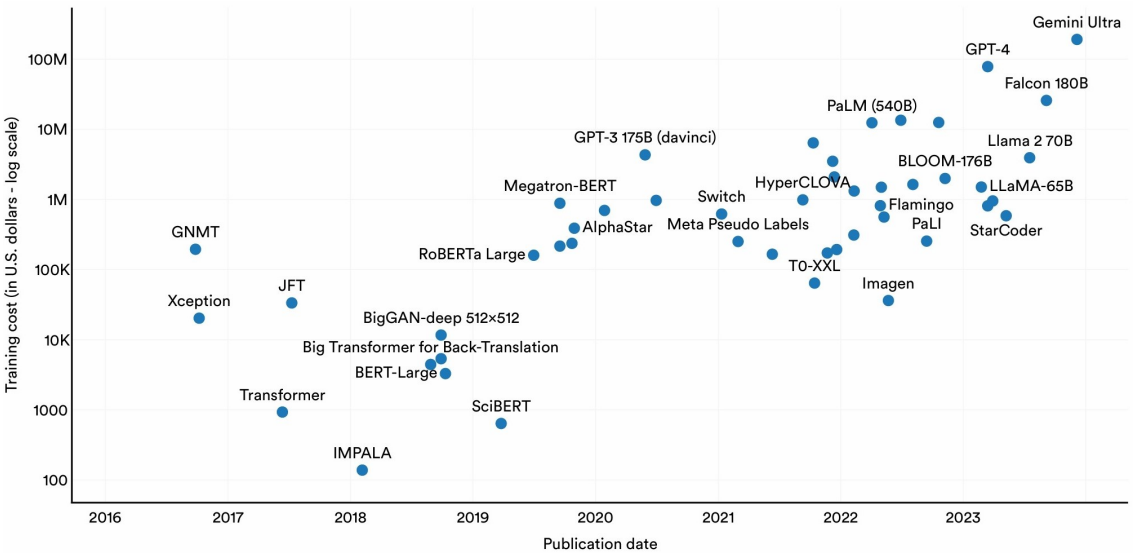


Figure 2.9: Estimated training cost of select models. *Extracted from [35].*

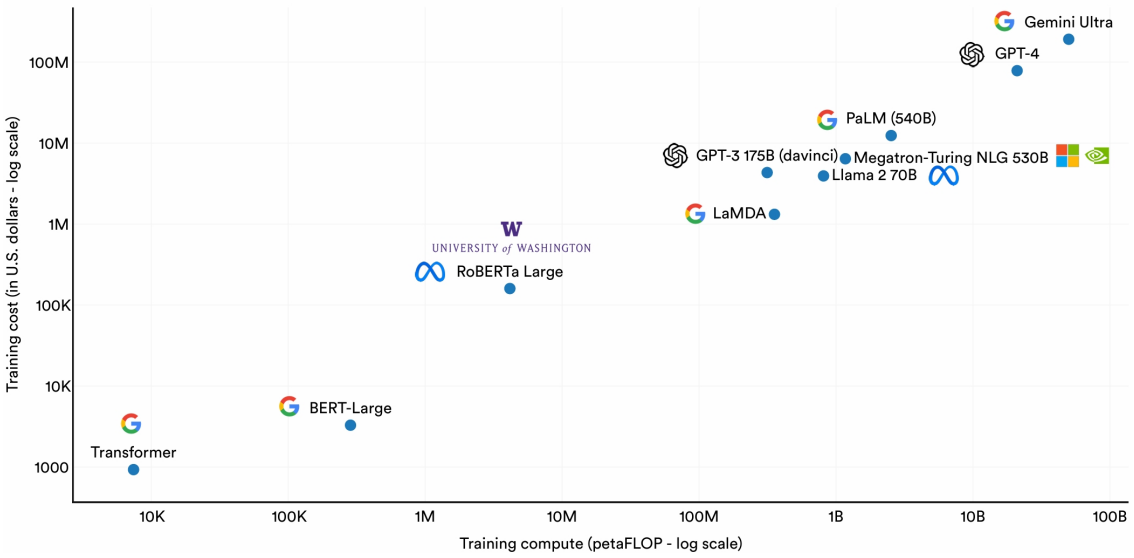


Figure 2.10: Estimated training compute of select models. *Extracted from [35].*

Of course, larger computing power means that more energy and resources are being invested into these models. As a result, the estimated CO2 emissions are already being counted in hundreds of tonnes, as shown in 2.11.

For these reasons, Artificial Intelligence in general has the responsibility of growing into a sustainable field, which does not represent a harm to society and the environment, as a way to achieve its ultimate goal: improving people's lives. Some techniques, such as transfer learning, directly help to reduce the costs and carbon footprint of the training process; this would not be possible if most models

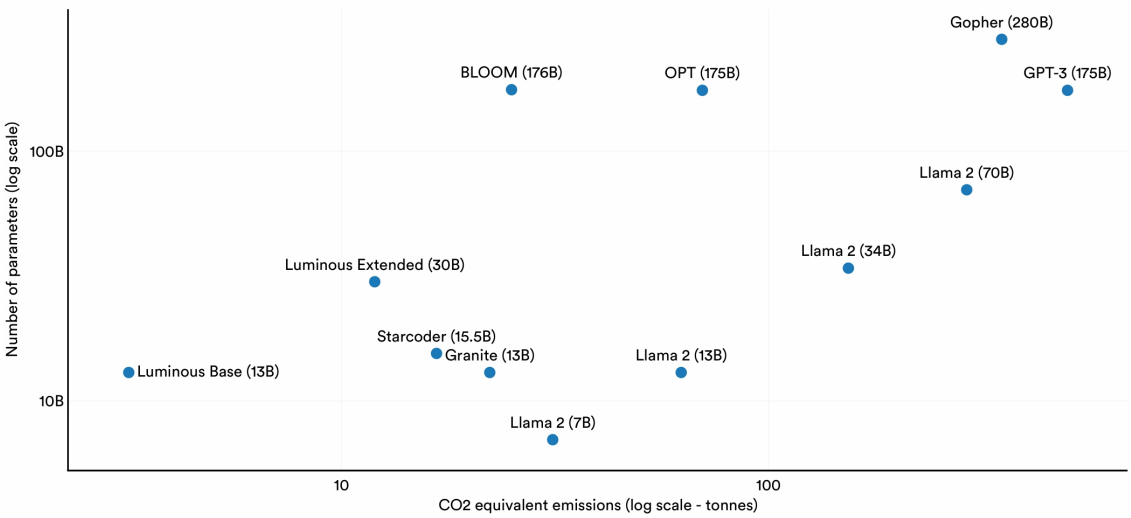


Figure 2.11: CO2 equivalent emissions by number of parameters of select models. *Extracted from [35].*

were not open and shared.

Not every impact is negative, though: AI-based systems are employed in environmental friendly solutions such as waste management [36], energy usage optimization and forecast of energy demand [37].

2.3.4 Limitations and biases

Even though LLMs are powerful and capable, they are not free from biases. Since they have been trained on large amounts of data, more often than not this data is not filtered enough from negative and slanted language and, as in any other machine learning process, the language model may inherit inappropriate behaviour. For this reason, most models are not suitable for production, as they can generate toxic and biased language. Moreover, LLMs are subject to hallucination: generation of incorrect, fabricated, or unbased language.

As a result, there is a special interest in developing neutral language models and safeguards to avoid generation of harmful content. With Phi model, Microsoft's researchers [21] used synthetic textbook-alike datasets for training in an attempt to reduce biases. In figure 2.12, safety scores of several LLMs are shown; these scores are computed using the dataset ToxiGen, designed for adversarial and implicit hate speech detection [38]. Among those tested, model Phi-1.5 proved to be the most neutral, whereas same model's version trained on filtered web data, Phi-1.5-web, was more likely to generate inappropriate content. However, Phi-1.5-web was also more performant in reasoning benchmarks, so the trade-off is there: web data can be rich and useful, but it comes with biases.

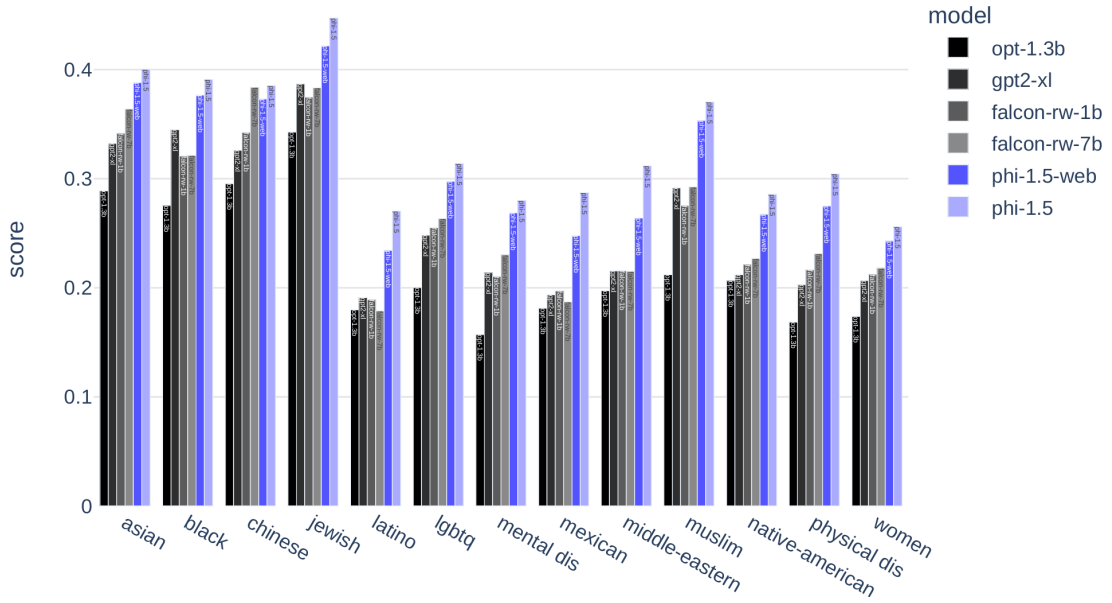


Figure 2.12: Safety scores of selected models. Scores are computed using ToxiGen dataset, and they range from 0 to 1: higher score indicates less likeliness to produce toxic content. *Extracted from [21].*

3 Technology and environment

Before delving into the experimental work, we shall present and analyse the environment to be used.

3.1 Python

When it comes to AI and machine learning, Python is the king. For this work, we will be using Python programming language, as it offers a key library: PyTorch [1], a free deep learning library based on Torch, commonly used in NLP and computer vision. It supports operations with tensors, which can be loaded into GPUs with Nvidia's CUDA, as well as the definition and training of neural networks. PyTorch is the most popular machine learning library alongside TensorFlow.

Moreover, other tools to be harnessed are based on Python, too: HuggingFace and Google Colab.

3.2 HuggingFace

HuggingFace [2] offers a platform and framework for building machine learning solutions. It provides the Transformers library [39], compatible with PyTorch and with access to hundreds of thousands of open source implementations of models for NLP, computer vision and audio. Within this library one can find Models, Tokenizers and Trainers, which will be necessary for the realization of this work.

From HuggingFace we shall also employ the libraries Datasets and Evaluate for dataset processing and model evaluation, and Accelerate [40], Bitsandbytes [41] and Parameter-Efficient Fine-Tuning (PEFT) [42] libraries for quantization and adaptation of large models. Moreover, we will make use of the HuggingFace Hub repositories, as they are a convenient way of storing and sharing datasets and model checkpoints.

All in all, HuggingFace provides more than enough tools to analyse, train and evaluate language models in a proper way.

[1] https://github.com/pytorch/pytorch
[2] https://huggingface.co

3.2.1 Tokenizers

Tokenizers are tools responsible for dividing text into minimal units of semantic meaning, that is, tokens. These are later converted into numbers or IDs, which are the inputs of language models. Since models work with numbers and tensors, tokenization is an elementary process in any NLP pipeline.

HuggingFace Tokenizers library [43] provides fast state-of-the-art tokenizers such as Byte-Pair Encoding, WordPiece, Unigram or SentencePiece; and all the necessary tools to pre-process and post-process data that goes in and out of the model. It is even possible to train a tokenizer to learn new language or domain tokens, for example, though we will not use this functionality during the work; we will focus on the default tokenizers of the base models employed, and we will explore special tokens (such as the pad and the separator tokens) to properly train and prompt the language models. In figure 3.1, the tokenizer pipeline for the language model T5 [18] is shown; from the input sentence, the tokenizer normalises (spaces, accents), pre-tokenizes (splits into words), proceeds with the tokenization (WordPiece, Unigram, etc. depeding on the implementation), and finally post-processes the result (for example, adding the special tokens, attention mask, etc.). The final result contains the separator token at the end, which was added at this last step.

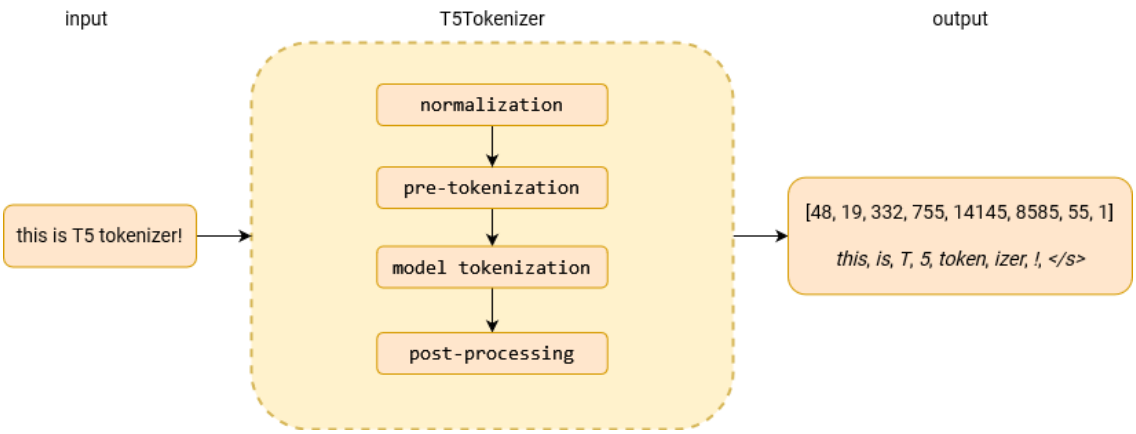


Figure 3.1: HuggingFace's T5 Tokenizer applied to a simple sentence.

3.2.2 Trainer

With a base model loaded, HuggingFace's Trainer and TrainingArguments classes from the Transformers library [39]provide an interface to further train or finetune it to generate a new checkpoint. To keep track of the trainings, we shall use the platform and library WandB [3].

Some relevant hyperparameters to be defined during transformers training are:

- **batch size**, which preferably should be a multiple of 8 as recommended by Nvidia [44]. Should be as high as the

[3] https://wandb.ai/site

processing unit can fit to maximize hardware resources.

- **evaluation strategy**, either steps or epoch. If the strategy is steps, it is necessary to configure warmup ratio or steps, max steps, and eval steps; with epoch, configure the max number of epoch instead.
- **learning rate**, establishes the convergence speed.
- **weight decay**, combined with learning rate in the case regularization is needed.

There are more hyperparameters to be configured in the Trainer; HuggingFace defines 111 arguments, but we will just focus on those aforementioned.

3.2.3 PEFT and QLoRA

For large models such as Phi, it is not feasible to train the full model; for this reason, we will employ PEFT [42] through the Quantized Low-Rank Adaptation (QLoRA) technique [45]. With quantization, we reduce memory requirements while maintaining good accuracy. The finetuning process is depicted in figure 3.2.

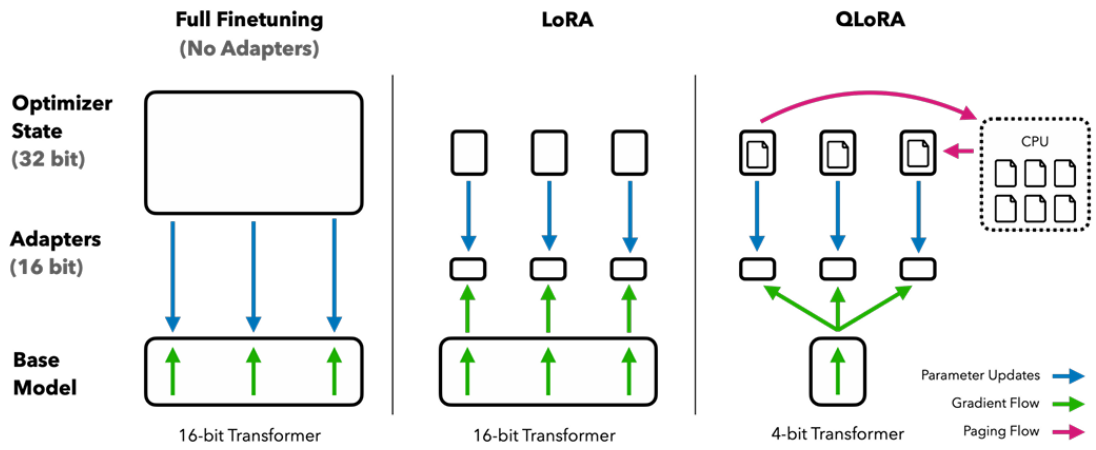


Figure 3.2: Model finetuning methods. QLoRA reduces memory requirements by quantizing the precision of the model and using paged optimizers. *Extracted from [45].*

A QLoRA adapter is mainly defined by the following parameters:

- **r**, the rank of the low-rank decomposition used in the adapters. Higher rank means more expressiveness and heavier computations.
- **alpha**, scaling factor for the low-rank adaptation (alpha/r), which controls the impact of the learned weights on the model. The greater, the more weight to the low-rank adapter activations.
- **target modules**, modules of the neural network to be adapted.
- **dropout**, the dropout rate applied to the low-rank matrices as a regularization technique to prevent overfitting.

3.2.4 Sharing is caring

The HuggingFace Hub has repositories for thousands of hundreds of models and datasets. This work is based on some of these models and datasets, and the results (finetuned model checkpoints, training logs, processed datasets) will be uploaded and published there as well, aggregated in a HuggingFace collection [4]. The goal is that anyone should be able to reproduce the results with the material provided.

3.3 Google Colab

Google Colab [5] is a cloud service that provides a hosted environment with the necessary resources and tools to code and share Jupyter Notebooks seamlessly, in Python and R languages. Colab was specially conceived with data science and machine learning in mind: users are offered several computing options, including access to Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) powered instances.

The main advantage of Google Colab is the possibility to use a free tier; even though there are several paid tiers, most services are available to everyone through the free tier, albeit with some constraints (for example, execution time, and most powerful processors are behind paywalls). Finding a stable and mostly open environment is part of the goals of this work.

We will analyse the different computing options currently available in order to choose the most suitable for our task. Note that the environment changes continuously, and it has evolved since this work started.

3.3.1 Hardware: GPU vs. TPU

A Tensor Processing Unit is an Application-Specific Integrated Circuit (ASIC) carefully designed by Google to perform operations with tensors using the TensorFlow library. TPUs are equipped with several Matrix Multiplier Unit (MXU), depicted in figure 3.3, which enhances the vector processing capabilities, necessary for training neural networks in machine learning tasks.

The crescent interest in AI solutions has led hardware companies to develop specialized GPU accelerators for tensor operations; in 2018, with Turing microarchitecture, Nvidia introduced for the first time tensor cores in general purpose GPUs [47]. These cutting-edge GPUs are suited for machine learning as well, and the chosen processing unit will depend on the task and the specifications of the model that is being trained. In table 3.1, a general overview and comparison of Google Colab's processor options is presented.

[4] https://huggingface.co/collections/enriquesaou/tfg-66670a768e3ed59181581e65
[5] https://colab.google

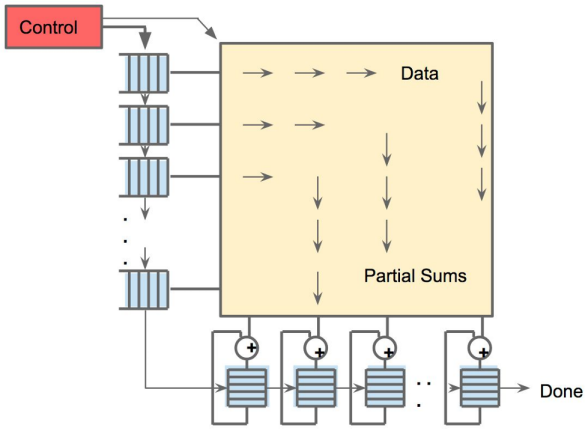


Figure 3.3: A Matrix Multiplier Unit, the heart of the TPU. *Extracted from [46].*

Processor model	Date	Litography	Clock (MHz)	Memory	TDP (W)	TOPS	Colab tier
TPUv1	2015	28nm	700	8GB GDDR3	Undisclosed	92	Free
TPUv2	2017	16nm	700	16GB HBM	280	180	Free
GPU Nvidia Turing T4	2018	12nm	1590	16GB GDDR6	70	130	Free
GPU Nvidia Ampere A100	2020/21	7nm	1410	40GB/80GB HBM2	250/300	624	Paid
GPU Nvidia Ada Lovelace L4	2023	5nm	2040	24GB GDDR6	72	485	Paid

Table 3.1: Comparison of Google Colab GPUs and TPUs. Tera Operations Per Second (TOPS) given are INT8 operations. Table is ordered by processor release date.

Latest TPU models, such as TPUv3 (2018), TPUv4 (2021) and TPUv5 (2023), are still not available for use in Google Colab (though they can be rented in Google Cloud services).

For this work, the chosen processor has been the GPU T4. The main drive to this decision has been its memory size (16GB vs. the 8GB of the first generation TPU). Another influential factor has been the average length of the execution sessions: in the free Colab tier, with the T4, Google offers from 2 to 3 hours of computing time, whereas with the TPU the available session time ranges from 1 to 2 hours. Besides, using a TPU, it was common to suffer disconnections from the service; with the T4 GPU, on the other hand, the environment has always been stable.

It is worth noting that when the training tasks started, TPUv2 was still not available for general use in Google Colab; for this reason, if the work was to be reproduced, it is encouraged to carry out a new analysis of the available hardware to choose the most suitable solution. At the moment, TPUv2 is still not recommended to use, as its availability is very low and, more often than not, connections to new execution sessions are rejected.

Ultimately, one has to keep in mind that a sometimes, for low workloads with small tensors, a TPU or GPU is not necessary, and a Central Processing Unit (CPU) might be better option. And, on the contrary, if the task is too demanding or the model is large, several processing units are going to be needed; libraries such as Microsoft's DeepSet [48] offer tools for distributed training and inference of large models, making them scalable, efficient and more accessible. However in this work, we will not

harness these technologies, since a multi-GPU environment is simply not feasible; a single GPU will be used.

3.3.2 Constraints

All in all, the environment presents the following constraints:

- **Finite computing time**, ranging from 2 to 3 hours per session, which can be enhanced by applying transfer learning techniques to finetune the model in several sessions.
- **Limited hardware**, as T4 GPUs are equipped with barely 16GB of vRAM. Training issues can be overcome by using gradient accumulation steps, effectively increasing the actual batch size, as seen in the equation 3.1. While large models do not fit in this hardware, relatively small models can be loaded and finetuned by using quantization techniques.

$$effective\_batch\_size = N_{devices} \times batch\_size_{device} \times gradient\_accumulation\_steps \tag{3.1}$$

- **No persistent storage**. However, HuggingFace provides utilities to upload datasets and model checkpoints to their repositories for later download and use. Moreover, repositories may also be shared for exploitation by other parties.

4

Experimental setup

With the environment and technologies chosen, it is time to focus on the experimental setup and procedures to train and evaluate the language models on QA tasks.

4.1 Models

For the work, three models families have been chosen in order to cover the three different transformer architectures: RoBERTa as an encoder-only, T5 as an encoder-decoder, and Phi as a decoder-only. They differ not only in architecture but also in size, as seen in table 4.1. It is worth noting that, to know the actual memory usage, one has to add the size of the data loaded into vRAM to the model size, and the size devoted to the computations (such as attention).

Architecture	Model	Precision	Size (MB)	Peak vRAM (MB)
Encoder-only	roberta-base	FP32	475	1860
Encoder-decoder	t5-small	FP32	231	923
	t5-base	FP32	850	3320
		FP16	425	1660
Decoder-only	phi-2	FP32	9880	39530
		FP16	4940	19760
		INT8	2470	9880
	phi-3	FP32	12844	51389
		FP16	6422	25688
		INT8	3211	12844

Table 4.1: Size comparison of base models. The peak vRAM corresponds to training with the Adam optimizer.

By selecting models of different architectures, we will be able to analyse the performance and possible trade-offs among them. Each architecture tackles QA tasks following its own approach: RoBERTa, as an encoder-only model, predicts the start and end positions of the tokens of the answer; T5 is

encoder-decoder, thus it generates the most probable answer tokens; and Phi, as a decoder-only model, excels at text generation, so we will teach the model how to generate an answer.

Furthermore, inside each model family, we have the chance of benchmarking several base and finetuned models, with alternative configurations. For example, for T5 there are two main base models: T5-base and T5-base FLAN. The latter has been finetuned in a mixture of instruction tasks, so it should perform better by scratch in QA — we can only be sure after evaluating, though. In addition, Google offers a small version of each T5 base model, so we can discuss whether the size and efficiency difference between them is noticeable enough. One may also compare if a base T5 trained in half precision (FP16/BF16), which occupies a similar size as a small T5 trained in full precision (FP32) according to table 4.1, has the same performance or not. Only after gathering the results, we will be able to obtain the main takeaways of the analysis.

The case of Phi is specially interesting; since it is a relatively large model, we need to use PEFT and QLoRA to properly load and train the model. In this way, we will train the model in 8 bit quantization with an adapter, which helps to reduce the memory requirements. After training the adapter, we can extract it and plug it into the full model for inference and evaluation. So, for Phi-2, we train in 8 bit and evaluate in full precision (FP32), whereas for Phi-3 we train in 8 bit and evaluate in half precision (FP16), as otherwise it would not fit into our hardware (GPU T4 has a memory of 16GB).

4.2 Dataset

After analysing the current QA datasets available, the chosen one has been the Machine Reading for Question Answering 2019 shared task dataset [49]. MRQA was a challenge that took part in 2019 in an attempt to become a testbed for evaluating generalization capabilities of QA models. The dataset used in the shared task carefully merges several QA subsets from different sources, presented in table 4.2, to the same format. Six of the subsets are in-domain, which are thought for training and validation, whereas the other six are out-of-domain and therefore are targeted for testing. As mentioned earlier, the goal is generalization, so an effective QA model should be able to predict out-of-domain examples too.

4.3 Pre-processing

Dataset transformation

Although MRQA states that the dataset follows the Stanford Question Answering Dataset (SQuAD) format, it is not true. SQuAD examples just have answers, context, id, question and title fields, which is not the case for MRQA examples. Therefore, the first pre-processing step involves converting to the

Split	Subset	Examples	Source
Training and validation (in-domain)	SQuAD	97095	Wikipedia [50].
	NewsQA	78372	CNN news articles [51].
	TriviaQA	69473	Trivia and quiz-league websites [52].
	SearchQA	134364	Jeopardy! TV show [53].
	HotpotQA	78829	Wikipedia (multi-hop reasoning) [54].
	Natural Questions	116907	Wikipedia [55].
Test (out-of-domain)	BioASQ	1504	PubMed articles [56].
	DROP	1503	Wikipedia (quantitative reasoning) [57].
	DuoRC	1501	Wikipedia and IMDb [58].
	RACE	674	English reading comprehension exams for middle and high school [59].
	Relation Extraction	2948	WikiReading [60].
	TextbookQA	1503	Lessons from middle school Life Science, Earth Science, and Physical Science textbooks [61].

Table 4.2: Subset number of examples and sources per split. Training and validation splits are considered in-domain, while test split contains out-of-domain examples.

SQuAD format, which is the standard in extractive QA. The resulting dataset has been published in the HuggingFace collection [1].

In order to ease the computations in our constrained environment, it is recommended to take a sample from a dataset instead of working with all the examples — after taking a random sample, we obtain the final dataset to operate, which can also be accessed at the HuggingFace Hub [2]. In table 4.3, the number of examples of each dataset is shown. To ensure the random sample distribution is similar to the original one, we have analysed the subset frequencies as seen in figure 4.1.

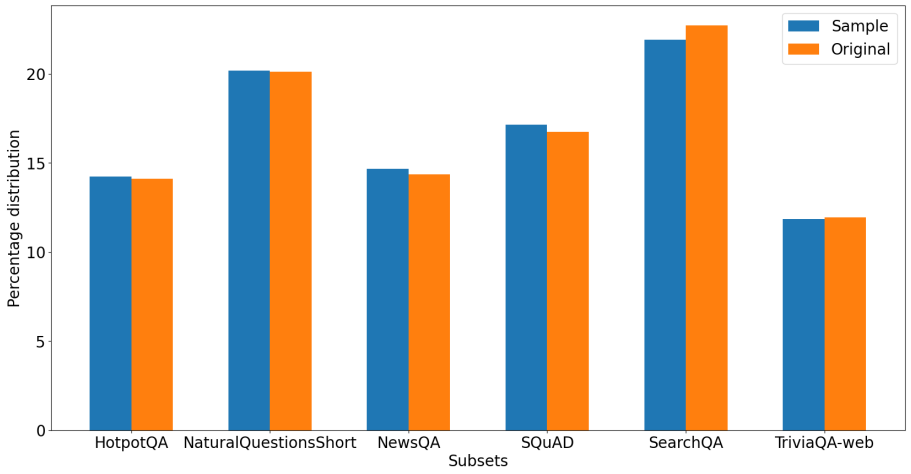
Dataset	Training examples	Validation examples	Test examples
MRQA	516819	58221	9633
Sampled MRQA	15000	1500	500

Table 4.3: Number of examples per dataset and split. The sampled dataset is more suitable for memory-constrained environments.

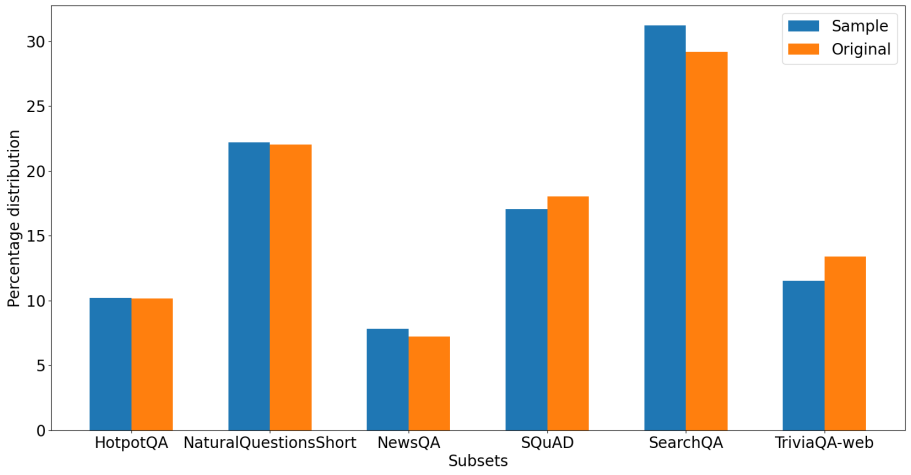
Formatting examples

After formatting the dataset for easier manipulation, it is necessary to format the training examples that will serve as inputs to the model. This process is dependent on the model architecture. We pad or

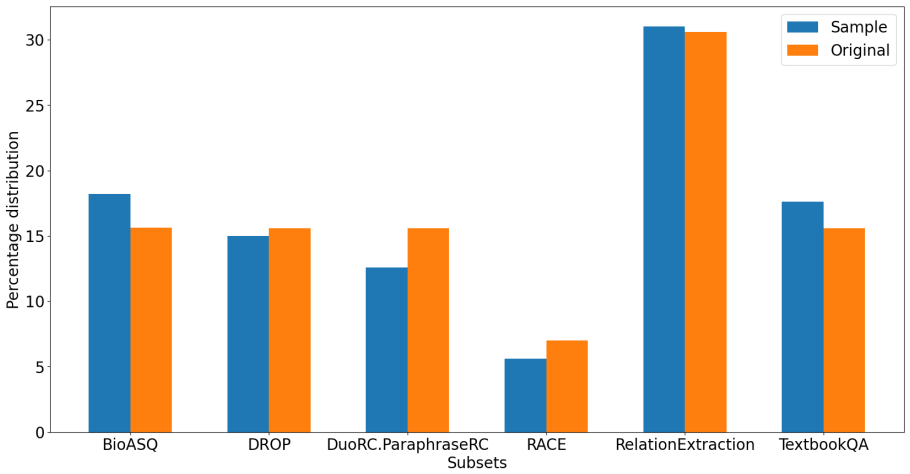
[1] https://huggingface.co/collections/enriquesaou/tfg-66670a768e3ed59181581e65
[2] https://huggingface.co/datasets/enriquesaou/mrqa-squadded-sample



(a) Distribution comparison in training data.



(b) Distribution comparison in validation data.



(c) Distribution comparison in test data.

Figure 4.1: Comparison of the subset distribution in the dataset splits. Figure 4.1(a) compares training data distribution, figure 4.1(b) compares validation data, and figure 4.1(c) compares test data. The distribution is very similar in all three cases.

truncate the examples to the maximum length estipulated for each model, in order to ease the tensor calculations.

For RoBERTa, the model takes as input the tokenized question and context, and the start and end positions of the answer within the tokenized context. Since it is an extractive model, it has to observe the entire context; as a result, if the example is larger than the maximum input length, the input is segmented with an stride. The maximum input length is 512 tokens and the stride is 128 tokens; they are both multiple of 8, and RoBERTa's maximum input length is 512, so setting it higher is not going to be advantageous.

In the case of T5, the model works sequence to sequence, so we feed input sequences with the format *[question: Q <sep> context: C]*; it has been proven that using the separator token enhances the performance of the model [62].

Ultimately, for Phi, we use the format *[Answer the question extracting from the context below. Context: C Question: Q]*. Since its a decoder-only model, the output of the model continues the model input, so we set left padding to prevent pad tokens to populate in between the sequences, breaking semantics. We set the maximum input length to 1200, which covers most of the examples in the dataset, as seen in figure 4.2.

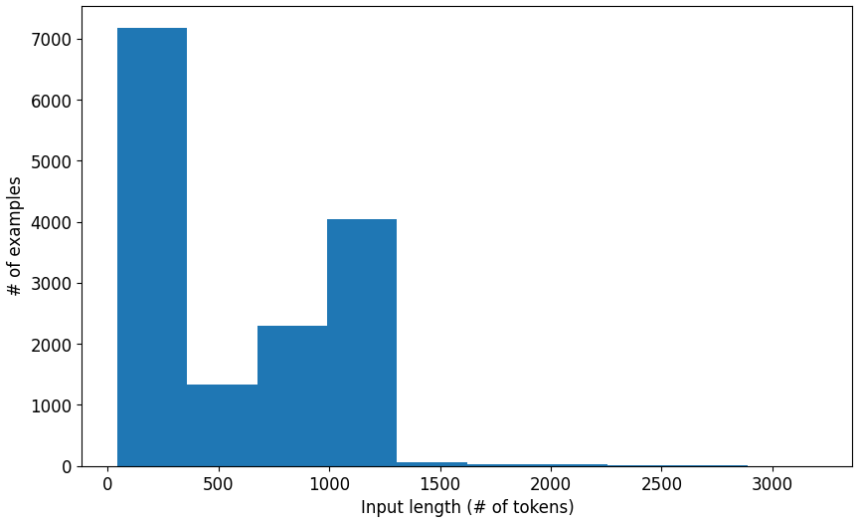


Figure 4.2: Token length of the training inputs. Examples have been tokenized with Phi-2 tokenizer.

4.4 Training

With the data properly processed and formatted, the training process is set to start. We finetune 2x RoBERTa checkpoints, 4x T5 checkpoints, and 2x Phi checkpoints. We compare these checkpoints with their respective base models and, in the case of RoBERTa, with a reference model trained on the MRQA dataset by VMware. All the recorded training logs and models are available at the HuggingFace

Hub [3].

Training details are presented in appendix A.1 for RoBERTa models, in A.2 for T5, and in A.3 for Phi. In all cases, the training optimizer is Adam (paged Adam when using QLoRA) with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. To fight overfitting, we have relied on weight decay and best-fitted checkpoints when necessary. In the case of Phi, we train the model following a PEFT approach using QLoRA adapters, quantized to 8 bits with computations performed in half-precision. The adapters used are the following:

- **Phi-2**: $r = 32, alpha = 16, target\ modules = [\text{"}Wqkv\text{"}, \text{"}fc1\text{"}, \text{"}fc2\text{"}], dropout = 0.05$
- **Phi-3**: $r = 16, alpha = 16, target\ modules = \text{"}all\_linear\text{"}, dropout = 0.05$

4.5 Metrics and evaluation

For evaluation, we shall use the MRQA official evaluation procedure, which relies on Exact Match (EM) and F1 score metrics, shown in equations 4.4 and 4.3 respectively. F1 score is the harmonic mean of the precision and recall; precision measures accuracy according to equation 4.1, whereas recall measures completeness as seen in equation 4.2. Note that MRQA metrics are computed at word-level.

$$Precision = \frac{|Ground\ truths \cap Predicted|}{|Predicted|} \in [0, 1] \tag{4.1}$$

$$Recall = \frac{|Ground\ truths \cap Predicted|}{|Ground\ truths|} \in [0, 1] \tag{4.2}$$

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 \times Precision \times Recall}{Precision + Recall} \in [0, 1] \tag{4.3}$$

$$EM = \begin{cases} 1 & if\ prediction = ground\ truth \\ 0 & otherwise \end{cases} \tag{4.4}$$

To calculate the scores, first the predictions are normalized; this procedure includes removing articles, punctuation, and converting the text to lowercase. With the normalized predictions, we obtain the maximum EM and F1 scores for a series of normalized ground truths (possible answers) for each example. To get the total score for each metric, we compute the average and obtain the percentage, as shown in equations 4.5 and 4.6.

$$EM_{total} = 100 \times \frac{\sum_{i=1}^{n} max(EM_i(ground\ truth_i, prediction_i))}{n} \in [0, 100] \tag{4.5}$$

$$F_{1\ total} = 100 \times \frac{\sum_{i=1}^{n} max(F_{1\ i}(ground\ truth_i, prediction_i))}{n} \in [0, 100] \tag{4.6}$$

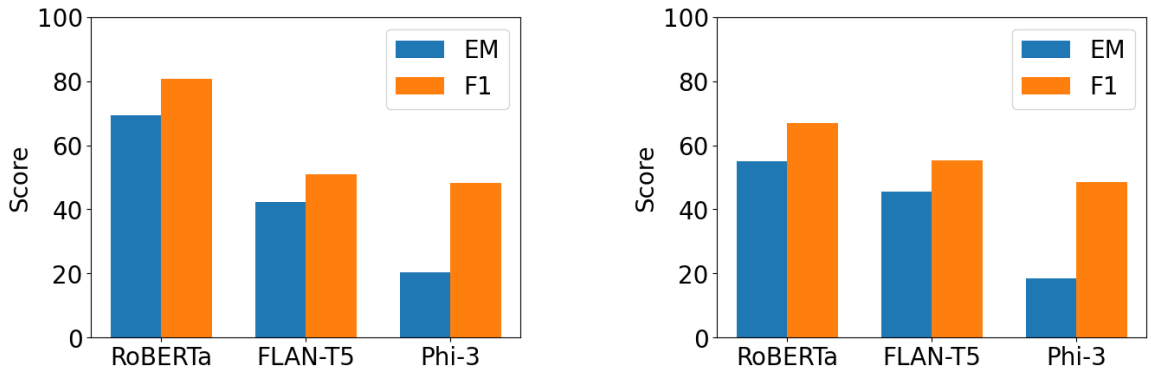
[3] https://huggingface.co/collections/enriquesaou/tfg-66670a768e3ed59181581e65

5 Results

In table 5.1, the relative performance increase for each finetuned model from their base version is shown. From this analysis, we find that RoBERTa is the best learner, and it is also the most performant as depicted in figure 5.1. Let us break down the results for each model in the following sections.

Base model	Val EM (%)	Val F1 (%)	Test EM (%)	Test F1 (%)
FacebookAI/roberta-base	+30533.35	+992.41	+NaN	+1131.38
VMware/roberta-base-mrqa	+0.29	+0.14	+0.36	+0.56
google-t5/t5-base	+9.74	+8.07	+0.58	+3.57
google-t5/t5-small	+44.67	+28.19	+5.88	+3.72
google/flan-t5-base	+35.32	+27.33	-7.32	-2.80
google/flan-t5-small	+65.12	+47.52	+29.73	+23.71
microsoft/phi-2	+240.70	+242.26	+181.40	+218.51
microsoft/phi-3-mini-4k-instruct	-6.69	+25.76	+15.00	+54.14

Table 5.1: Model relative performance increase. Relative percentage increase is given for EM and F1 metrics in the validation and test splits, from the base model to our finetuned version.



(a) Best scores per family in the validation split.

(b) Best scores per family in the test split.

Figure 5.1: Best scores per model family. Figure 5.1(a) shows the best models in the validation split; figure 5.1(b) in the test split. T5's best is tuned FLAN-T5-base and RoBERTa's is tuned VMware's.

5.1 RoBERTa: from zero to hero

RoBERTa is the model with most improvement among those tested. The performance is shown in table 5.2. The base model has the worst performance seen in this work, and the finetuned checkpoint over VMware's model has the best performance among all.

Base model	Checkpoint	Val EM (%)	Val F1 (%)	Test EM (%)	Test F1 (%)
FacebookAI/roberta-base	Base	0.2000	6.6965	0.0000	4.9310
	Finetuned	61.2667	73.1534	48.4000	60.7191
	Reference (VMware)	69.2000	80.7016	54.8000	66.4221
VMware/roberta-base-mrqa	Finetuned	69.4000	80.8151	55.0000	66.7953

Table 5.2: RoBERTa models evaluation results. EM and F1 metrics are given for evaluation and test splits. Reference model has been trained by VMware on the MRQA dataset (see appendix A.1 for details).

RoBERTa is an encoder-only model, thus relies on language understanding — after finetuning, the model learns the task and adapts perfectly. Besides, its approach to extractive QA, consisting of predicting the start and end positions of the tokens of the answer, proves to be the most suitable for the task. We were able to beat the reference model by VMware, which had been finetuned in a more powerful environment as detailed in appendix A.1, by further finetuning it with a small learning rate — our finetuned checkpoint from the base model had fell short by merely a 10%.

5.2 T5: better with FLAN

In the case of T5 family, we had the chance to finetune and test several models. Results are depicted in table 5.3; FLAN models have better performance before and after finetuning than default models, as the former have been specially prepared for instructions.

T5 is the model that best generalises to the test split, compared to the validation performance, though it is not the best scorer. Regarding base and small models, we observe that the base model offers a notable increase in performance compared to the small version, even after finetuning. Note that the finetuning of base models has been performed in half precision in such a way that the size during training is similar to the small model, allowing a fair comparison. During this training, there were stability issues caused by the decrease in precision, which we partly overcame by selecting the best fitted checkpoint — however, we believe the training was still not optimal. In any case, the performance of the base model is better than the small one: for our extractive QA task, it is better to reduce the base model precision than to embrace a smaller version of the model.

Base model	Checkpoint	Val EM (%)	Val F1 (%)	Test EM (%)	Test F1 (%)
google-t5/t5-base	Base	28.0667	37.0190	34.6000	43.6886
	Finetuned	30.8000	40.0060	34.8000	45.2494
google-t5/t5-small	Base	19.4000	29.0286	27.2000	37.3609
	Finetuned	28.0667	37.2115	28.8000	38.7521
google/flan-t5-base	Base	31.3333	39.9489	49.2000	56.8686
	Finetuned	42.4000	50.8689	45.6000	55.2781
google/flan-t5-small	Base	22.9333	31.3518	29.6000	39.8096
	Finetuned	37.8667	46.2503	38.4000	49.2489

Table 5.3: T5 models evaluation results. EM and F1 metrics are given for validation and test splits. T5-base and FLAN-T5-base have been finetuned in half precision.

5.3 Phi: not a matter of size

Since decoder-only models such as Phi are in reality text generators, we can play with the generation strategy during evaluation and testing. In figure 5.2 we observe the relation between the maximum new tokens to generate and the EM and F1 scores. The best EM score is achieved at 4 new tokens, and the best F1 at 5 tokens. The reason is because, as seen in figure 5.3, most ground truth answers are 1 to 5 tokens long, and giving the model more tokens to generate will only decrease the EM and the precision metrics (thus decreasing F1, too) whenever the model fails to properly generate the end-of-sequence token.

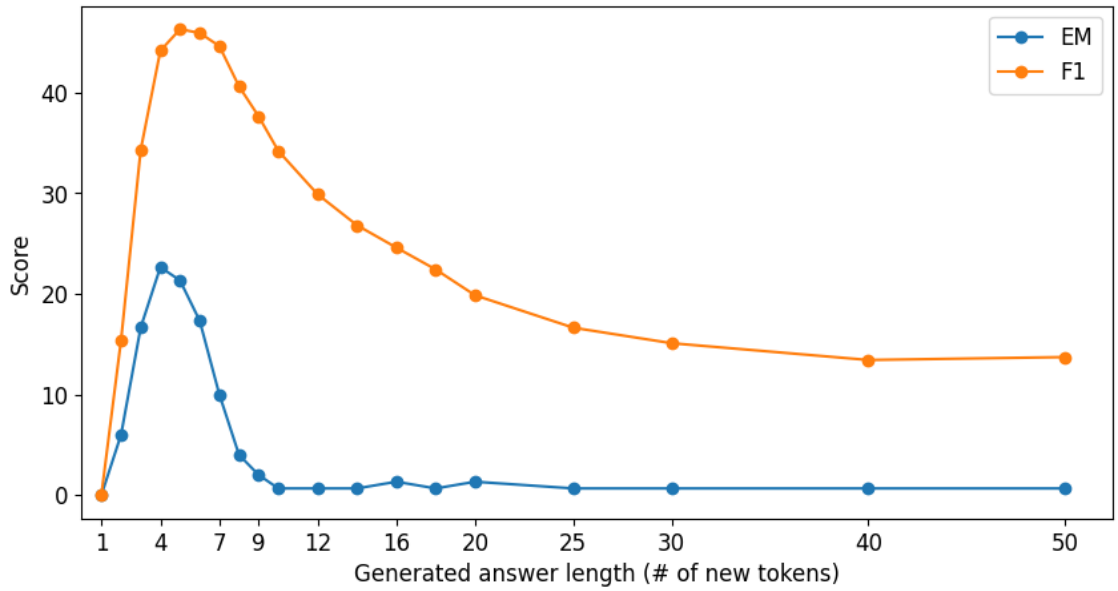
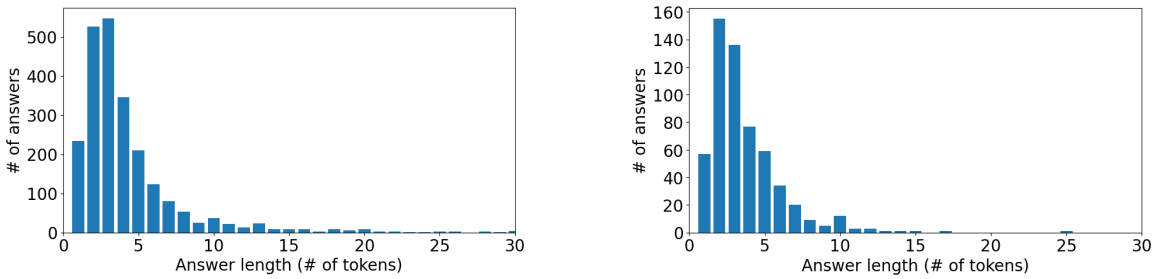


Figure 5.2: EM and F1 scores against the maximum number of new tokens. Model tested is enriquesaou/phi-mrqa on a random sample of the validation dataset. Best F1 score is achieved with a maximum of 5 new tokens; best EM with 4 tokens.



(a) Distribution in validation data sample.

(b) Distribution in test data sample.

Figure 5.3: Distribution of answer token length in the dataset sample. Figure 5.3(a) depicts the validation split length distribution while figure 5.3(b) shows the test split one. Answers have been tokenized with Phi-2 default tokenizer.

In table 5.4, results are presented for Phi models following a generation strategy with 5 new tokens, which is the optimal strategy for this dataset. Base version of Phi-3 comfortably beats Phi-2 base, though after finetuning this difference is much narrower. It is worth noting that Phi-3 model is 30% larger than Phi-2, and as a result its performance is better. Base version of Phi-3 is specially performant for a decoder-only model, but it is still lower than T5's encoder-decoder, even though is the largest model we have tested.

Base model	Checkpoint	Val EM (%)	Val F1 (%)	Test EM (%)	Test F1 (%)
microsoft/phi-2	Base	5.7333	11.8713	8.6000	13.7806
	Finetuned	19.5333	40.6311	24.2000	43.8922
microsoft/phi-3-mini-4k-instruct	Base	21.9333	38.2553	16.0000	31.5231
	Finetuned	20.4667	48.1092	18.4000	48.5912

Table 5.4: Phi models evaluation results. EM and F1 metrics are given for validation and test splits. Text generation strategy with 5 maximum new tokens.

5.4 Examples

For a better understanding of the model behavior, we need insights into their actual responses to the QA tasks. We make a selection of passages and check the answers of our best models of each family.

In frame 5.1, we check each model capability in multi-hop reasoning. Multi-hop involves relating several concepts to understand the question and extract the answer: in this example, the model has to link Louis Smith collaborators and the occupations of Miles Davis. We may also test the behavior when it comes to answer a name. Both RoBERTa and T5 produce an accurate answer, but RoBERTa's is not fully contemplated in the ground truth answer thus it has an EM of 0: in most cases, EM is not a reliable metric. One can argue whether T5's response is more accurate, even though they refer to the

same entity (Miles Davis) — the F1 metric reflects the difference in completeness but fails to assess the semantics of the names. The incompleteness of the ground truths has a negative influence to the F1 metric, as if both *Miles Davis* and *Miles Dewey Davis III* were ground truths, F1 would have been fully reliable. In the case of Phi, the correct answer is given, but afterwards the model fails to stop and continues generating tokens. Note that the text generation strategy chosen establishes a maximum of 5 new tokens; otherwise, the model score would have fallen drastically.

Context: *"[PAR] [TLE] Louis Smith (musician) [SEP] While studying at the University of Michigan, he played with visiting musicians such as Dizzy Gillespie, Miles Davis, Thad Jones and Billy Mitchell, before going on to play with Sonny Stitt, Count Basie and Al McKibbon, Cannonball Adderley, Percy Heath, Philly Joe Jones, Lou Donaldson, Donald Byrd, Kenny Dorham and Zoot Sims. [PAR] [TLE] Miles Davis [SEP] Miles Dewey Davis III (May 26, 1926September 28, 1991) was an American jazz trumpeter, bandleader, and composer. He is among the most influential and acclaimed figures in the history of jazz and 20th century music. Davis adopted a variety of musical directions in his five-decade career which kept him at the forefront of a number of major stylistic developments in jazz."*
Question: *"While at the University of Michigan, Louis Smith played with what American jazz trumpeter, bandleader, and composer?"*
Answer: *"Miles Dewey Davis III"*
RoBERTa prediction: *"Miles Davis" (EM: 0.0%, F1: 66.7%)*
T5 prediction: *"Miles Dewey Davis III" (EM: 100.0%, F1: 100.0%)*
Phi prediction: *"Miles Davis\nQuestion" (EM: 0.0%, F1: 57.1%)*

Frame 5.1: Example from the HotpotQA subset in the validation split. Predictions correspond to our finetuned VMware's RoBERTa, FLAN-T5-base, and Phi-3, which are our best models.

Example in frame 5.2 presents another proof of the text generation impact in Phi model: the answer is truncated. In fact, if we increase the maximum tokens to generate in such a way that it is not truncated, the model provides the right answer and continues generating after that — it again fails to stop. This issue is inherent to auto-regressive decoder-only models, and we indeed check that RoBERTa and T5 (as encoder-only and encoder-decoder models respectively) do not experience the same problem. Still, F1 metric proves to be reliable and scores Phi's answer high.

In frame 5.3, we observe that all three models succeed to distinguish start dates. This information is not directly given and thus the models have to interpret the relation between dates. Besides, we see that generation with numbers is more straightforward than with names, as they are inherently free of semantic issues.

As a final example, we select a question with a distinctive format, as shown in frame 5.4. The models

Context: *"The rebellion of the city of Danzig was a revolt from December 1575 to December 1577 of the city against the outcome of the Polish-Lithuanian royal election, 1576. The Polish throne was contested by Stephen Báthory and the Holy Roman Emperor Maximillian II. It began on 12 December 1575 when Emperor Maximillian was chosen as monarch by the Polish Senate, while the majority of the szlachta had voted for Bathory. It ended on 16 December 1577. Maximilian's II death in fall of 1576 weakened Danzig's position and made the conflict less about the recognition of the ruler than about Danzig's privileges. With neither side being able to defeat the other militarily, a compromise was reached, with economic as well as religious privileges of the city being restored and recognized, in return for a large reparation and recognition of Bathory as the king."*
Question: *"What happened first, the end of the rebellion of the city of Danzig or Maximilian's II death?"*
Answer: *"Maximilian's II death"*
RoBERTa prediction: *"Maximilian's II death" (EM: 100.0%, F1: 100.0%)*
T5 prediction: *"Maximilian's II death" (EM: 100.0%, F1: 100.0%)*
Phi prediction: *"Maximilian's II" (EM: 0.0%, F1: 80.0%)*

Frame 5.2: Example from the DROP subset in the test split. Predictions correspond to our finetuned VMware's RoBERTa, FLAN-T5-base, and Phi-3, which are our best models.

Context: *"The 52nd government of Turkey (30 October 1995 – 6 March 1996) was a caretaker coalition government formed by True Path Party (DYP) and Republican People's Party (CHP)."*
Question: *"What year did 52nd government of Turkey start?"*
Answer: *"1995"*
RoBERTa prediction: *"1995" (EM: 100.0%, F1: 100.0%)*
T5 prediction: *"1995" (EM: 100.0%, F1: 100.0%)*
Phi prediction: *"1995" (EM: 100.0%, F1: 100.0%)*

Frame 5.3: Example from the RelationExtraction subset in the test split. Predictions correspond to our finetuned VMware's RoBERTa, FLAN-T5-base, and Phi-3, which are our best models.

need to first understand the question and then extract the answer from the context. T5 and Phi give accurate responses (although Phi continues to showcase generation issues), but the case of RoBERTa is interesting. Its answer is *methane* instead of *natural gas*. While not all natural gases are methane, methane is indeed a natural gas. So, should RoBERTa's answer be marked as completely incorrect? There is definitely an argument there. However, none of our metrics can capture the semantics of the answer, and in any case the other models' answers were more accurate — our metrics do reflect it.

Context: *"Can you name some fossils? How about dinosaur bones or dinosaur footprints? Animal skeletons, teeth, shells, coprolites (otherwise known as feces), or any other remains or traces from a living creature that becomes rock is a fossil. The same processes that formed these fossils also created some of our most important energy resources, fossil fuels. Coal, oil, and natural gas are fossil fuels. Fossil fuels come from living matter starting about 500 million years ago. Millions of years ago, plants used energy from the Sun to form sugars, carbohydrates, and other energy-rich carbon compounds. As plants and animals died, their remains settled on the ground on land and in swamps, lakes, and seas (Figure 1.1). Over time, layer upon layer of these remains accumulated. Eventually, the layers were buried so deeply that they were crushed by an enormous mass of earth. The weight of this earth pressing down on these plant and animal remains created intense heat and pressure. After millions of years of heat and pressure, the material in these layers turned into chemicals called hydrocarbons (Figure 1.2). Hydrocarbons are made of carbon and hydrogen atoms. This molecule with one carbon and four hydrogen atoms is methane. Hydrocarbons can be solid, liquid, or gaseous. The solid form is what we know as coal. The liquid form is petroleum, or crude oil. Natural gas is the gaseous form. The solar energy stored in fossil fuels is a rich source of energy. Although fossil fuels provide very high quality energy, they are non-renewable. Click image to the left or use the URL below. URL:"*

Question: *"hydrocarbons in gas form are called _____."*

Answer: *"natural gas"*

RoBERTa prediction: *"methane" (EM: 0.0%, F1: 0.0%)*

T5 prediction: *"natural gas" (EM: 100.0%, F1: 100.0%)*

Phi prediction: *"natural gas\nQuestion:" (EM: 0.0%, F1: 80.0%)*

Frame 5.4: Example from the TextbookQA subset in the test split. Predictions correspond to our finetuned VMware's RoBERTa, FLAN-T5-base, and Phi-3, which are our best models.

5.5 Carbon footprint

The recorded training accumulates a total of 222 hours, as reported by the library WandB. This includes finetuning sessions that were not adopted in the final work (though it does not include initial sessions which were not logged).

The estimated carbon efficiency of Google Colab in Europe region is 0.62 kgCO_2eq/kWh; with a GPU T4 (TDP of 70W) and 222 hours of computation, the estimated emissions are 9.63 kgCO_2eq — this is equivalent to 38.9 Km driven by an average combustion car. Estimations were conducted using the MachineLearning Impact calculator [63].

6

Conclusions

All the code used during this work for pre-processing, training and evaluation is available at Github [1]. The datasets, models and training logs are available at the HuggingFace Hub [2]. After analysing the obtained results, we extract the following main takeaways:

- **Encoders are the best understanders, but only after finetuning.** For extractive QA, encoder-only models are the best suited transformers, followed by encoder-decoder and lastly decoder-only models. However, this only applies after task finetuning; models with decoder compensate bad understanding of the task with good language generation capabilities — in our analysis, T5 and Phi base models performed better than RoBERTa base. Extractive QA relies mainly in understanding, which is the encoder speciality, and for a fraction of the size of large decoder models the performance is better. Different tasks will require different model's capabilities; finding the most suitable architecture is part of the task.
- **Finetuning is very beneficial and relatively inexpensive.** Task finetuning greatly increases the model's capabilites, thus it should be applied before deploying the model to production. In the case of Phi-2, a finetuning session 300000 times less computational intensive than the base model's pretraining improved performance by more than 300% in our QA benchmarks.
- **Embrace instruction finetuned models, if available.** Instruction finetuned models, such as FLANs, are better suited than base models for QA tasks, and therefore they should be used instead as a base for further finetuning. This is most probably applicable to other NLP tasks, too.
- **Leverage community models wisely.** In some situations, the community has already published a model finetuned on a dataset or task which is similar to our purpose. After a proper analysis, one can leverage these models as the base for further tuning, transfering their learned knowledge to our own task. This was the case with VMware's RoBERTa, which had already been trained on the MRQA dataset; its performance was improved after more finetuning, but special attention must be paid on overfitting: use sensibly parameters such as the learning rate and weight decay.
- **Training larger models on lower precision may be a good idea.** In our study, the base version of T5 finetuned in half precision performed better than the small version finetuned in full precision. Under these conditions, the sizes during training were similar, but the obtained performance was not. However, training in decreased precision can lead to stability issues, so special care is required.
- **Optimize text generation with decoder models.** Text generation strategy, specially in decoder-only models, is crucial to optimize performance even in tasks with an extractive (non-generative) nature. The strategy depends on the dataset and task, so a proper study and analysis of them is recommended.
- **Pre-processing, formatting and tokenization is important.** In decoder-only models, pad the examples to the

[1] https://github.com/enriquesaou/tfg-lm-qa

[2] https://huggingface.co/collections/enriquesaou/tfg-66670a768e3ed59181581e65

left when necessary to mantain semantics in the input. In encoder-decoder models, use special tokens, such as the separator token, to format the input sequences. In any case, adopt the input sequence formatting that enhances performance the most, which may not be the one recommended by the base model developer for every task.

And most importantly, this work has presented all the necessary background and information on how to set an environment up to experiment with language models — to understand them, just like they seem to understand us.

6.1 Future work

One of the main traits and achievements of this work is that it was conducted in a heavily constrained environment, relying only on free and open technologies. However, for LLMs, this setup is definitely not ideal, and productive models should be trained and deployed on more capable hardware. Perhaps the experiments of this work could be recreated with state-of-the-art hardware, too. For example, Google's massive model PaLM [64] was trained on 6144 TPUv4 — a single TPUv4 is more than 30 times faster than the T4 used during this work. Google, can i borrow one of those, please?

Another side to look upon is evaluation. We used EM and F1 metrics, but more complex methods such as semantic answer similarity could be implemented. This issue is not limited to this work though — model evaluation has become a real challenge in current LLMs, and it is specially hard to compare performance of generative models.

Generative QA is quite an interesting field. In this work we explored extractive QA, so the next step would be to adventure into generative QA by merely using decoder-only models and advanced evaluation techniques.

Looking at the bigger picture, I believe there is plenty of work to be done in the NLP field. Transformers are advancing quickly and there thousands of models and datasets publicly available. For example, Phi-3 was released during the realization of this work, so we decided to incorporate it to the analysis. It is possible to reiterate the work presented but with newer models and different domains.

There are also more research opportunities in finetuning tasks: how do models adapt to domain-specific datasets? What models perform better in that certain task? How do we test this performance? Can we develop a reliable testbed for a certain domain or task? Are other domain adaptation methods, such as RAG, well suited and effective?

And in regard to the deployment of language models: can relatively small models be run on consumer hardware while offering decent results? Can software solutions make use of these models? Do they properly protect companies and individuals data? We will certainly see the answers to these questions in the near future. At the end of the day, it is yet another Question Answering task.

Bibliography

[1] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI, Tech. Rep., 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[2] H. Touvron *et al.*, "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] A. Gonzalez Fernandez, "AI-powered Digital Assistant (AIDA)," *ESA Software Repository*, 2024, accessed: 2024-05-29. [Online]. Available: https://essr.esa.int/project/ai-powered-digital-assistant-aida

[4] J. Alvarez, "El Gran Salto de España en la IA: Alia, el Primer Modelo de Lenguaje en Español," *AlvarezJoseph*, 2024, accessed: 2024-05-29. [Online]. Available: https://www.alvarezjoseph.com/blog/el-gran-salto-de-espana-en-la-ia-alia-el-primer-modelo-de-lenguaje-en-espanol

[5] E. D. Liddy, "Natural language processing," in *Encyclopedia of Library and Information Science*, 2nd ed. New York: Marcel Decker, Inc., 2001.

[6] N. Chomsky, "On certain formal properties of grammars," *Information and control*, vol. 2, no. 2, pp. 137–167, 1959.

[7] C. E. Shannon and J. McCarthy, *Automata studies.(AM-34)*, ser. Annals of Mathematics Studies. Princeton University Press, 1956.

[8] N. Chomsky, *Syntactic structures*. Mouton de Gruyter, 2002.

[9] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[10] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[11] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, p. 1137–1155, mar 2003.

[12] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, no. 3. Makuhari, 2010, pp. 1045–1048.

[13] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] A. Hern, "New AI fake text generator may be too dangerous to release, say creators," 2019, accessed: 2024-05-12. [Online]. Available: https://www.theguardian.com/technology/2019/feb/14/elon-musk-backed-ai-writes-convincing-news-fiction

[15] S. Raschka, "Understanding encoder and decoder LLMs," 2023, accessed: 2024-06-11. [Online]. Available: https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder

[16] Jiyoung, "Transformer implementation and understanding," 2021, accessed: 2024-06-11. [Online]. Available: https://velog.io/@jiyoung/Transformer-%EA%B5%AC%ED%98%84%ED%95%98%EA%B3%A0-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B02

[17] M. Lewis *et al.*, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[18] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[20] Y. Liu *et al.*, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," *arXiv preprint arXiv:1907.11692*, 2019.

[21] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need II: phi-1.5 technical report," *arXiv preprint arXiv:2309.05463*, 2023.

[22] R. Gruetzemacher and D. Paradice, "Deep transfer learning & beyond: Transformer language models in information systems research," *ACM Computing Surveys (CSUR)*, vol. 54, no. 10s, pp. 1–35, 2022.

[23] Hugging Face, "How do transformers work?" 2023, accessed: 2024-06-11. [Online]. Available: https://huggingface.co/learn/nlp-course/chapter1/4

[24] OpenAI, "Flagship models," 2024, accessed: 2024-06-01. [Online]. Available: https://platform.openai.com/docs/models

[25] Epoch AI, "Parameter, compute and data trends in machine learning," 2024, accessed: 2024-06-08. [Online]. Available: https://epochai.org/data/epochdb/visualization

[26] B. Schwartz, "Google: BERT now used on almost every English query," *Search Engine Land*, 2020, accessed: 2024-06-01. [Online]. Available: https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193

[27] H. W. Chung *et al.*, "Scaling instruction-finetuned language models," *Journal of Machine Learning Research*, vol. 25, no. 70, pp. 1–53, 2024.

[28] A. Q. Jiang *et al.*, "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[29] M. A. team, "Au large," *Mistral AI news*, 2024, accessed: 2024-06-01. [Online]. Available: https://mistral.ai/news/mistral-large/

[30] A. Q. Jiang *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[31] S. Betts, "Peering Inside GPT-4: Understanding Its Mixture of Experts (MoE) Architecture," *Medium*, 2023, accessed: 2024-06-01. [Online]. Available: https://medium.com/@seanbetts/peering-inside-gpt-4-understanding-its-mixture-of-experts-moe-architecture-2a42eb8bdcb3

[32] G. Team *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[33] M. Abdin *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[34] W. Knight, "OpenAI's CEO Says the Age of Giant AI Models Is Already Over," *Wired*, 2023, accessed: 2024-05-29. [Online]. Available: https://www.wired.com/story/openai-ceo-sam-altman-the-age-of-giant-ai-models-is-already-over/

[35] R. Perrault and J. Clark, "Artificial Intelligence Index Report 2024," 2024, accessed: 2024-06-06. [Online]. Available: https://aiindex.stanford.edu/report/

[36] B. Fang *et al.*, "Artificial intelligence for waste management in smart cities: a review," *Environmental Chemistry Letters*, vol. 21, no. 4, pp. 1959–1989, 2023.

[37] E. T. Sayed *et al.*, "Renewable energy and energy storage systems," p. 1415, 2023.

[38] T. Hartvigsen, S. Gabriel, H. Palangi, M. Sap, D. Ray, and E. Kamar, "ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Implicit Hate Speech Detection," *arXiv preprint arXiv:2203.09509*, 2022.

[39] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[40] S. Gugger *et al.*, "Accelerate: Training and inference at scale made simple, efficient and adaptable." https://github.com/huggingface/accelerate, 2022, accessed: 2024-06-06.

[41] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," *CoRR*, vol. abs/2110.02861, 2021.

[42] S. Mangrulkar, S. Gugger, L. Debut, Y. Belkada, S. Paul, and B. Bossan, "PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods," https://github.com/huggingface/peft, 2022, accessed: 2024-06-06.

[43] A. Moi *et al.*, "Tokenizers: Fast, optimized tokenizers for natural language processing." https://github.com/huggingface/tokenizers, 2024, accessed: 2024-06-06.

[44] NVIDIA Corporation, *Deep Learning Performance: Fully Connected (Dense) Layers*, 2024, accessed: 2024-06-07. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html#batch-size

[45] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," *arXiv preprint arXiv:2305.14314*, 2023.

[46] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[47] J. Burgess, "Rtx ON — the Nvidia Turing GPU," *IEEE Micro*, vol. 40, no. 2, pp. 36–44, 2020.

[48] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[49] A. Fisch, A. Talmor, R. Jia, M. Seo, E. Choi, and D. Chen, "MRQA 2019 shared task: Evaluating generalization in reading comprehension," *arXiv preprint arXiv:1910.09753*, 2019.

[50] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine com-

prehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[51] A. Trischler *et al.*, "NewsQA: A machine comprehension dataset," *arXiv preprint arXiv:1611.09830*, 2016.

[52] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[53] M. Dunn, L. Sagun, M. Higgins, V. U. Guney, V. Cirik, and K. Cho, "SearchQA: A new Q&A dataset augmented with context from a search engine," *arXiv preprint arXiv:1704.05179*, 2017.

[54] Z. Yang *et al.*, "HotpotQA: A dataset for diverse, explainable multi-hop question answering," *arXiv preprint arXiv:1809.09600*, 2018.

[55] T. Kwiatkowski *et al.*, "Natural questions: a benchmark for question answering research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019.

[56] G. Tsatsaronis *et al.*, "An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition," *BMC bioinformatics*, vol. 16, pp. 1–28, 2015.

[57] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner, "DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs," *arXiv preprint arXiv:1903.00161*, 2019.

[58] A. Saha, R. Aralikatte, M. M. Khapra, and K. Sankaranarayanan, "DuoRC: Towards complex language understanding with paraphrased reading comprehension," *arXiv preprint arXiv:1804.07927*, 2018.

[59] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, "RACE: Large-scale reading comprehension dataset from examinations," *arXiv preprint arXiv:1704.04683*, 2017.

[60] O. Levy, M. Seo, E. Choi, and L. Zettlemoyer, "Zero-shot relation extraction via reading comprehension," *arXiv preprint arXiv:1706.04115*, 2017.

[61] A. Kembhavi, M. Seo, D. Schwenk, J. Choi, A. Farhadi, and H. Hajishirzi, "Are you smarter than a sixth grader? Textbook Question Answering for Multimodal Machine Comprehension," in *Proceedings of the IEEE Conference on Computer Vision and Pattern recognition*, 2017, pp. 4999–5007.

[62] M. Luo, K. Hashimoto, S. Yavuz, Z. Liu, C. Baral, and Y. Zhou, "Choose your QA model wisely: A systematic study of generative and extractive readers for question answering," *arXiv preprint arXiv:2203.07522*, 2022.

[63] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *arXiv preprint arXiv:1910.09700*, 2019.

[64] A. Chowdhery *et al.*, "PaLM: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

Acronyms

AI Artificial Intelligence.

ASIC Application-Specific Integrated Circuit.

CPU Central Processing Unit.

EM Exact Match.

FLAN Finetuned Language Net.

FLOP FLoating Point Operations.

FNN Feedforward Neural Network.

GPU Graphics Processing Unit.

LLMs Large Language Models.

MoE Mixture of Experts.

MRQA Machine Reading for Question Answering.

MXU Matrix Multiplier Unit.

NLP Natural Language Processing.

PEFT Parameter-Efficient Fine-Tuning.

QA Question Answering.

QLoRA Quantized Low-Rank Adaptation.

RAG Retrieval Augmented Generation.

RLHF Reinforcement Learning from Human Feedback.

RNN Recurrent Neural Network.

SQuAD Stanford Question Answering Dataset.

TOPS Tera Operations Per Second.

TPU Tensor Processing Unit.

APPENDICES

A

Training details

A.1 RoBERTa

Training details for RoBERTa models are presented in table A.1.

Base model	Generated checkpoint	Training data	Training time	Hyperparameters
FacebookAI/roberta-base	FacebookAI/roberta-base	BookCorpus English Wikipedia CC-News OpenWebText Stories	120:00:00 on 1024x V100	$learning\_rate = 6e-4$, $weight\_decay = 0.01$ $learning\_rate\_warmup = 24000\ steps$ $eval\_strategy = steps$, $max\_steps = 500k$ $batch\_size = 8k$
	enriquesaou/roberta-mrqa	MRQA sample	2:18:53 on 1x T4	$learning\_rate = 2e-5$, $weight\_decay = 0.01$ $eval\_strategy = epoch$, $num\_epochs = 3$ $batch\_size = 20$
	VMware/roberta-base-mrqa	MRQA	8:50:29 on 1x V100	$learning\_rate = 1e-5$, $weight\_decay = 0.01$ $eval\_strategy = epoch$, $num\_epochs = 1$ $batch\_size = 16$
VMware/roberta-base-mrqa	enriquesaou/roberta-vmw-mrqa	MRQA sample	2:25:39 on 1x T4	$learning\_rate = 3e-6$, $weight\_decay = 0.01$ $eval\_strategy = epoch$, $num\_epochs = 3$ $batch\_size = 20$

Table A.1: RoBERTa training details and relevant hyperparameters. Names correspond to HuggingFace's IDs. Models finetuned during this work are prefixed with enriquesaou/. In the case of roberta-base checkpoint, full pretraining details are shown. In the case of VMware/roberta-base-mrqa checkpoint, VMware's finetuning details are shown.

A.2 T5

Training details for T5 models are presented in table A.2.

Base model	Generated checkpoint	Training data	Training time	Hyperparameters
google-t5/t5-base	google-t5/t5-base	C4 Wiki-DPR CoLA SST-2 MRPC, STS-B, QQP MNLI, QNLI, RTE, CB COPA WIC MultiRC, ReCoRD, BoolQ	0:49:15 on 512x TPUv3	$learning\_rate = 0.01$ $(-\ scheduler)$ $weight\_decay = 0.01$ $learning\_rate\_warmup = 10000\ steps$ $eval\_strategy = steps, max\_steps = 524288$ $batch\_size = 128$
	enriquesaou/t5-base-mrqa-16	MRQA sample	2:56:32 on 1x T4	$learning\_rate = 3e-5, weight\_decay = 0.01$ $eval\_strategy = epoch, num\_epochs = 3$ $batch\_size = 6, gradient\_accum = 3$ $fp16 = True$
google-t5/t5-small	google-t5/t5-small	C4 Wiki-DPR CoLA SST-2 MRPC, STS-B, QQP MNLI, QNLI, RTE, CB COPA WIC MultiRC, ReCoRD, BoolQ	0:13:34 on 512x TPUv3	$learning\_rate = 0.01$ $(-\ scheduler)$ $weight\_decay = 0.01$ $learning\_rate\_warmup = 10000\ steps$ $eval\_strategy = steps, max\_steps = 524288$ $batch\_size = 128$
	enriquesaou/t5-small-mrqa	MRQA sample	2:49:43 on 1x T4	$learning\_rate = 3e-5, weight\_decay = 0.01$ $batch\_size = 14, gradient\_accum = 3$ $eval\_strategy = epoch, num\_epochs = 8$
google/flan-t5-base	google/flan-t5-base	T0-SF Muffin CoT Natural Instructions v2	Undisclosed. Finetuned version of pretrained T5.	$learning\_rate = 5e-4, weight\_decay = 0.05$ $eval\_strategy = steps, max\_steps = 84k$ $batch\_size = 64$
	enriquesaou/flan-t5-base-mrqa-16	MRQA sample	2:49:57 on 1x T4	$learning\_rate = 3e-5, weight\_decay = 0.01$ $eval\_strategy = epoch, num\_epochs = 2$ $batch\_size = 4, gradient\_accum = 2$
google/flan-t5-small	google/flan-t5-small	T0-SF Muffin CoT Natural Instructions v2	Undisclosed. Finetuned version of pretrained T5.	$learning\_rate = 5e-4, weight\_decay = 0.05$ $eval\_strategy = steps, max\_steps = 98k$ $batch\_size = 64$
	enriquesaou/flan-t5-small-mrqa	MRQA sample	2:40:27 on 1x T4	$learning\_rate = 3e-5, weight\_decay = 0.01$ $batch\_size = 12, gradient\_accum = 3$ $eval\_strategy = epoch, num\_epochs = 7$

Table A.2: T5 training details and relevant hyperparameters. Names correspond to HuggingFace's IDs. Models finetuned during this work are prefixed with enriquesaou/. In the case of t5-base and t5-small checkpoints, full pretraining details are shown. In the case of flan-t5-base and flan-t5-small checkpoints, FLAN training details are shown.

A.3 Phi

Training details for Phi models are presented in table A.3.

Base model	Generated checkpoint	Training data	Training time	Hyperparameters
microsoft/phi-2	microsoft/phi-2	AOAI GPT-3.5 synthetic data Falcon RefinedWeb filtered web data SlimPajama filtered web data	336:00:00 on 96x A100	$learning\_rate = 1e-3$, $weight\_decay = 0.1$ $learning\_rate\_warmup = 750\ steps$ $eval\_strategy = steps$, $max\_steps = 36000$ $batch\_size = 1024$
	enriquesaou/phi-2-mrqa	MRQA sample	3:27:03 on 1x T4	$learning\_rate = 3e-5$, $batch\_size = 4$ $eval\_strategy = steps$, $max\_steps = 500$ $warmup\_ratio = 0.03$, $eval\_steps = 100$
microsoft/phi-3-mini-4k-instruct	microsoft/phi-3-mini-4k-instruct	Public quality documents Synthetic textbook-alike data Quality chat format data	168:00:00 on 512x H100	Undisclosed (pretrained by Microsoft)
	enriquesaou/phi-3-mrqa	MRQA sample	2:43:59 on 1x T4	$learning\_rate = 3e-5$, $batch\_size = 4$ $eval\_strategy = steps$, $max\_steps = 300$ $warmup\_ratio = 0.03$, $eval\_steps = 100$

Table A.3: Phi training details and relevant hyperparameters. Names correspond to HuggingFace's IDs. Models finetuned during this work are prefixed with enriquesaou/. In the case of phi-2 and phi-3-mini-4k-instruct base checkpoints, full pretraining details are shown.


UÁM
Universidad Autónoma
de Madrid