NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

RICARDO VALVERDE LOPES

B.Sc. in Computer Science

RICH LARGE-SCALE PORTUGUESE LANGUAGE MODELS FROM LARGE PORTUGUESE CORPORA

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
October, 2023

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

RICH LARGE-SCALE PORTUGUESE LANGUAGE MODELS FROM LARGE PORTUGUESE CORPORA

RICARDO VALVERDE LOPES

B.Sc. in Computer Science

Adviser: David Semedo
Assistant Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
October, 2023

Rich Large-Scale Portuguese Language Models from Large Portuguese Corpora

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v6.10.10) [1].

To my family, girlfriend and friends.

Acknowledgements

First off, I would like to express my gratitude to my advisor Prof. David Semedo, whose guidance, support, availability, and motivation were a key part in the development of this thesis. It is with gratefulness that I would like to thank the VisionBox Project (CC 04040101) and everyone involved in it for providing me with a scholarship for 9 months.

Secondly, I want to thank my colleagues João Arvana and Daniel Castanho, with whom I collaborated on the production of ArquiWiz.pt, as well as a massive thanks to the team behind Arquivo.pt for their support, suggestions, and helpfulness. Posteriorly, I would like to thank Prof.João Magalhães and the NOVA Vision & Language Research group, who provided me with the essential infrastructure for the development of this work and helped me with any technical issues.

I am also thankful to my girlfriend and my group of best friends (*M.S*) for their help in reviewing one of the produced datasets, and above this, for accompanying me throughout most of my academic life all the way from primary school.

Lastly, I would like to thank my family - father, mother & sister (and my dogs) - who have supported me and my work throughout my life.

"Se eu não morresse, nunca! E eternamente
Buscasse e conseguisse a perfeição das cousas! "
(Cesário Verde)

Abstract

Language is one of the most fundamental and important characteristics of human behavior. It enables us to express ourselves and communicate, as it is a powerful and crucial tool that has helped shape our thoughts and knowledge of the world around us, since the dawn of humankind.

In specific, the Portuguese language is the sixth most spoken language in the world with over 250 million speakers worldwide. Of those, over 40 million are potential European Portuguese speakers, but despite its widespread use, the development of natural language processing (NLP) tools for PT-PT has lagged behind other languages, like English or French. This is partly due to the lack of large-scale annotated datasets and the lack of computational resources dedicated to this variant of Portuguese.

The NLP field has improved greatly in recent years, leading to the development of innovative tools for language analysis and processing boosted by neural language models. This is achieved by training deep, transformed-based large language models that can perform language tasks like machine translation, sentiment analysis, summarization, or even simple reasoning.

This thesis aims to address these current problems and contribute to the development of PT-PT NLP tools by presenting our own generative model, GlórIA. It can properly model the intricacies of the Portuguese language and is proficient at multiple natural language tasks. We present training techniques and protocols, following proper evaluation and comparison against other recent Portuguese models on several downstream tasks.

Consequently, a PT-PT corpora, composed of different sources of data, was built to train it and is presented in this work to combat the lack of publicly available datasets for this language variant. Parallel to this, a new and small benchmark was also produced to evaluate a model's generative performance on a language modeling task.

Keywords: Large Language Models, Transformers, Portuguese, Natural Language Processing, Datasets

Resumo

A linguagem humana é uma das características mais importantes e fundamentais do comportamento e relações humanas. Esta permite que o ser humano seja capaz de se expressar e comunicar, fazendo desta uma ferramenta poderosa e crucial que nos tem ajudado e acompanhado desde o início da humanidade.

Em concreto, a língua Portuguesa é a sexta língua mais falada em todo o mundo, com mais de 250 milhões de falantes. Destes, estima-se que mais de 40 milhões são potenciais utilizadores do Português Europeu, mas apesar da sua larga utilização, o desenvolvimento de ferramentas de processamento de linguagem natural (*NLP*) tem perdido algum terreno comparado a outras línguas como o Inglês ou o Francês. Isto deve-se parcialmente à falta de conjuntos de dados anotados de grande escala nesta língua, tal como à ausência de recursos computacionais dedicados a esta.

Nos últimos anos, importantes melhorias foram alcançadas nesta área, possibilitando o desenvolvimento de ferramentas inovativas para problemas de linguagem natural. Isto é possível treinando modelos de linguagem de grande escala, capazes de executar tarefas como sumarização, análise sentimental, tradução ou raciocínios simples.

Esta tese procura contribuir para o desenvolvimento destas ferramentas, apresentando um novo modelo generativo de 1.3B parâmetros: GlórIA, capaz de modelar as complexidades desta língua e gerar texto robusto. Apresentamos detalhes de treino e protocolos, seguidos de avaliação e comparação com outros modelos em diversas tarefas.

Consequentemente, foi produzido um corpora PT-PT composto por várias fontes de texto, utilizado para treinar o modelo. Paralelamente, este trabalho deu origem a um novo *benchmark* para avaliar as capacidades generativas de um modelo deste tipo.

Palavras-chave: Processamento de Linguagem Natural, *Transformers*, Língua Portuguesa, Modelos de Linguagem, *Conjuntos de Dados*

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Context and Motivation

Language is effectively one of the most important tools and characteristics of our society, if not the most crucial one. Specifically, European Portuguese (PT-PT) is spoken by around 10 million (approximate population of Portugal at the time of writing) and a few million more people worldwide, without taking into account other different dialects (Mozambique, Guinea-Bissau, etc).

Despite its widespread use, the resources and models for this language are lagging behind when compared to the English counterpart in Natural Language Processing (NLP). This field leverages the usage of language in our daily lives to develop models that attempt to replicate and improve human-like capabilities in computational language understanding and generation.

One of the main motivations for training a European Portuguese language model is to fill the said gap of resources and to make sure that it can be used to improve several natural language tasks (question answering, summarization, named entity recognition, etc.).

Most of these models, like the GPT Radford et al. [14] and BERT Devlin et al. [4] families, are focused on the English language, but these are suboptimal for other languages due to differences in meanings, semantics, and syntax, among others.

Recent European Portuguese models were proposed (Rodrigues et al. [12]), but their architectures don't follow a generative nature (word-by-word), as we will talk about posteriorly in this work. So, a dedicated generative PT-PT language model could be able to better employ the unique characteristics and nuances of the language to freely generate sentences.

Additionally, such a language model could support the development of novel applications and services, tailored to the needs of European Portuguese speakers. Culturally, it would contribute to the preservation and rich literary patrimony of the language.

1.2 Challenges and Research Hypothesis

Currently in *deep learning*, one of the biggest issues is establishing cause-effect relations between architectures, algorithms, and how they tend to affect the final results. Discovering which combination of well-studied architectures and training methodologies (how the models are trained) constitutes a great part of ongoing research. This, when combined with the lack of research done on language modeling (text understanding and generation) for the Portuguese language (when compared to English), launches a diverse set of challenges that must be addressed:

- The first challenge of this thesis was finding which architecture, model size, or training scheme is the most adequate to create a European Portuguese language model, including how these choices affect performance on PT-PT natural language tasks - a challenge that is shared with other languages.

- To train a language model, we need a large amount of unlabeled text data. Since little research on deep learning models in NLP was done regarding the European Portuguese language, there currently only exist very few publicly available PT-PT labeled and unlabeled datasets for fine-tuning and downstream tasks. This led us to **look for and collect new sources of data, and even contribute with a novel benchmark**;

- The final challenge was evaluating and comparing our model to existing ones. This is due to the very low amount of diverse Portuguese datasets, consequently limiting the natural language tasks that we can choose - and **proving that our model works for a diverse collection of tasks is of utmost importance**.

To overcome these challenges, we'll aim to leverage modern and large generative language models like OpenAI's GPT-2 or GPT-3 (Radford et al. [14] and Brown et al. [15]) or other models with similar architecture. Their architectures (based on the Transformer by Vaswani et al. [3]) and the techniques developed to train and fine-tune them have been proven to be highly effective in a variety of tasks according to their results.

We hypothesize that, together with a large and diverse PT-PT corpus, we can use one of these models, **or another one with a very similar architecture** as a basis for this thesis to develop a **European Portuguese large language model (LLM), capable of generating text, and able to achieve decent performance on multiple downstream NLP tasks**.

In sum, this work aims to produce model that can **produce coherent and contextually correct PT-PT texts**, covering a large domain of topics and knowledge.

1.3 Contributions

The contributions for the work developed through this thesis are:

- **A rich (topic diverse) large-scale PT-PT generative language model, named GlórIA** , that uses state-of-the-art architecture and training techniques - with 1.3B and 2.7B parameters for two versions. This model can perform several natural language tasks in the Portuguese language, and is very capable of generating human-like text;

- **A large European Portuguese text corpus that can be used to pre-train European Portuguese state-of-art language models**. This dataset is built using diverse data from the Portuguese Web Archives, together with other sources like the Portuguese Wikipedia and transcripts from the EU parliament. Most of this dataset will be available, in the form of already post-processed smaller datasets, that bring about a large diversity domain when put together. This dataset was used to pre-train our PT-PT large language model;

- **A new and generative benchmark, CALAME-PT** (Context-Aware LAnguage Modeling Evaluation for Portuguese), an open-ended task used to evaluate Portuguese generative language models on a zero-shot environment;

- **A paper covering the developments of this thesis** related to pre-training a European Portuguese model. At the time of writing, this article was accepted for publication in PROPOR 2024[1], and is being prepared for publication.

Modern large models like GPT-3 (Brown et al. [15]) and GPT-4 don't have their training details and source code released (at the time of writing), so it is difficult for researchers to better understand how such a model works. With this in mind, all of these contributions will be open-sourced (publicly available), to combat the lack of European Portuguese models and corpora - so future researchers can either use this as a baseline or perform more experimentations on top of it, and improve it further.

1.3.1 ArquiWIZ - A parallel contribution

GlórIA 1.3B and 2.7B are to be integrated into ArquiWiz[2]. ArquiWiz is a platform developed in partnership with two other Master's theses, whose goal is to democratize access to Arquivo.PT's data (Gomes et al. [16]) through advanced features. This is done by integrating a Question-answer paradigm, supported by language models, that allows users to query for archived news articles using natural language. It also supports different graph visualizations, that map entities and their relation throughout time. GlórIA will come to improve a Q&A ensemble of models.

ArquiWiz was also submitted to the annual Arquivo.PT Award[3]. This award aims to recognize innovative works based on the historical information preserved by Arquivo.pt. The produced works should be practical applications or studies based on this information,

[1] https://propor2024.citius.gal
[2] http://arquiwiz.pt/
[3] https://sobre.arquivo.pt/pt/colabore/premios-arquivo-pt/premio-arquivo-pt-2023/

with the goal of demonstrating the usefulness of this public service and the importance of preserving information published on the web.

Note: Some text required for the training corpus of this thesis was crawled and scrapped from Arquivo.PT in partnership with these other two theses (Arvana [17],Castanho [18]), as both their work and ArquiWiz are also dependent on its data.

1.4 Document Organization

Chapter 1 - Introduction: This is an introductory chapter that goes over the motivation and challenges that led to this thesis, together with its goals and expected contributions.
Chapter 2 - Related Work: The goal of this chapter is to lay out the current architectures and techniques used by these models in existing literature, that are relevant to this work.
Chapter 3 - Gathering & Building a PT-PT Corpus: Describes the process of gathering, filtering, and processing the text data that composes our pre-training data. It includes a description of its composition, as well as some statistics.
Chapter 4 - Pre-Training a Large PT-PT Language Model: Overview of the actual pre-training process of GlórIAIt starts with some notes on initial testing and methodology, followed by our chosen parameters, protocols, and training parameters. In the end, some pre-training results are shown and analyzed.
Chapter 5 - Assessing GlórIA 's Text Generation Capabilities: Introduces the first part of the evaluation process. Here, we exhibit some examples of text produced by our model. Posteriorly, we introduce CALAME-PT, our own benchmark, and the process behind building it - followed by a robust evaluation of our model and another on this benchmark.
Chapter 6 - Evaluating Models on Supervised & Discriminative Tasks: After evaluating GlórIA 's most relevant ability (its text generation), the sixth chapter introduces 3 new downstream tasks. We evaluate both GlórIA 1.3B and other Portuguese models and compare them against each other.
Chapter 7 - Final Conclusions: In this final chapter, we present some possible future work goals, together with a final analysis and conclusion of the work performed in this thesis.

2

Related Work

In the following chapter, we will come to grips with the background knowledge that serves as a basis for this thesis. We'll go over the main concepts and recent, relevant work, starting with the steppings stones that lead to the Transformer architecture, and how the models based on it have been evolving over time, up to adequate and novel techniques to train and create such a model.

2.1 Language Representations

2.1.1 Word Representations

In order to tackle this project, it is necessary to grasp how words and their context/relations can be represented. Recently, the predominant approach is to represent a word as an n-dimensional vector, called an embedding. These vectors are what fuel state-of-the-art language models since these embeddings are learnable and can carry syntactic and contextual meaning between words. Due to being able to represent multiple features and being n-dimensional, word similarity can be checked by how close they are when placed in a multidimensional space.

2.1.1.1 Word2Vec

Word2Vec (Mikolov et al. [2]) is a technique that, by using a shallow neural network and a large corpus of text, produces these embeddings by trying to learn word relations or associations via Continuous Bag-of-Words (CBOW) or Skip-gram models. In the CBOW model, the objective is to predict a word given its context - looking at the surrounding words, and averaging their vectors. On the contrary, the Skip-gram model aims to predict the context of a word given the word itself - predicting words within a certain range before and after the input word.

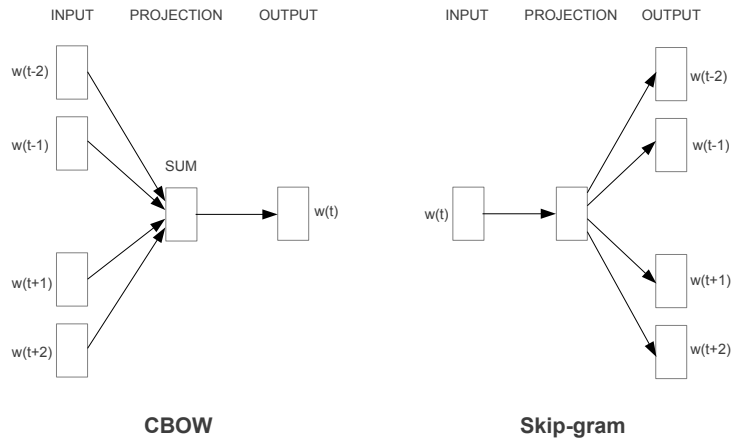

Figure 2.1: CBOW and Skip-gram models - respectively. (Mikolov et al. [2])

2.1.1.2 GloVe

Following up on what Word2Vec was built on, **Glove** (Pennington, Socher, and Manning [19]) was proposed - an unsupervised log-bilinear regression model that achieves a compromise between the Word2Vec model and statistical analysis of the words, by looking at word co-occurrences over the whole text corpus. Both these last two models focus on producing embeddings for single words.

2.1.2 Contextualizing Textual Representations

2.1.2.1 ELMO

An important problem with language that existed was polysemy - the ability for the same word to have multiple meanings or contexts. The embeddings obtained by the Word2Vec and GloVe models were not able to discern the various contexts a word could have and so, **ELMO** (Peters et al. [20]) showed a possible, technical way, to try to solve this problem by modeling complex characteristics of word usage and how these can vary upon different contexts. ELMO is a bi-directional deep language model (biLM) trained on predicting the next word on a sequence of words (called Language Modelling). Each layer is composed of a forward pass and a backward pass. The forward pass contains information about the current word (and context), and all the words that come before it. The backward pass is the same, except it contains information about all the words that come after it. Between each layer, the information from these passes are called the intermediate vectors, which are passed into the next layer, and in the end, the final result is the weighted sum of raw word vectors that act as input and the intermediate ones - in other words, the contextual representation of a token is the sum of its backward and forward passes representations.

2.1.2.2 Recurrent Neural Networks

Since text is sequential and its order matters - words follow after one another - Recurrent Neural Networks become ideal when trying to process and learn this type of data. They diverge from a standard feed-forward neural network by taking advantage of a loop mechanism. This mechanism allows information to flow back into the node encoded in a hidden state, and due to this, the nodes receive the output vector of the previous step, together with the new input vector. This way the network can attend to current and previous sequence vectors at the same time.

As they process step after step, RNNs begin to have trouble retaining information about previous steps (Zhang et al. [21]). As the network uses the error calculated by the loss function to perform backpropagation by calculating and updating the gradients in each node, the gradient can shrink or "explode" exponentially. This is known as the vanishing gradient problem. Essentially, if the sequence is long enough, information pertaining to the earlier steps that are represented in the gradient can end up being lost in the hidden states. Long Short-Term Memory (Hochreiter and Schmidhuber [22]) or Gate Recurrent Units are an extension of RNNs that are able to alleviate this problem by using operations that can choose which information to add or remove from the hidden state. Intuitively, these can detect an important feature in an input sequence, and carry this information over a long distance, better capturing long-distance dependencies without their loss (Chung et al. [23]).

2.2 Transformers

Due to the vanishing gradient problem, previous models had trouble leveraging dependencies in long sequences of text. Later, Vaswani et al. [3] proposed the **Transformer**, a state-of-the-art architecture that focuses on using self-attention mechanisms to draw dependency relations between all the positions of the input sequences - overcoming the RNNs difficulty, and being more computationally efficient since the Transformer is optimized to work in a parallelized way.

The self-attention mechanisms will be explained in a moment, but before that, it is important to know that the input consists of positional encodings together with the token's embeddings. The positional encodings describe the position that a token takes in a sequence. Without the positional encodings, the model would not be able to take into account where certain words are located in a sentence and how that can influence the context.

The Transformer architecture, according to Figure 2.2, follows an encoder-decoder stack structure, where the encoder maps an input sequence to another sequence of representations, which are then used by the decoder to generate an output sequence, one element at a time. This model is auto-regressive due to previously generated representations being used as additional input when progressing to the following steps.

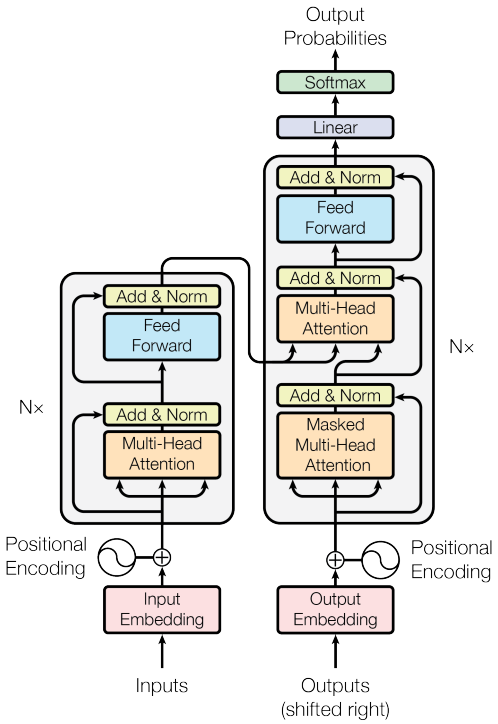

Figure 2.2: Transformer Architecture (Vaswani et al. [3]).

2.2.1 The encoder stack

The encoder stack is composed of 6 (the number can vary) identical layers, where each layer contains two sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network. The sub-layers are wrapped in residual connections that are followed by a normalization layer - these allow to relate the sub-layer's input with their own output, attending to previous calculations. In the beginning, the sequence that the bottom encoder takes as input are word embeddings, together with the position encoding of each word. Then, each word goes through its own path in the encoder according to its position, into the self-attention layer, and then into a feed-forward neural network. The outputs are normalized after each of these steps and sent to the next encoder up the stack once it finishes going through an encoder block.

2.2.2 The decoder stack

Like the encoder stack, it is composed of 6 (again, it can vary) identical layers. The decoder unit contains the encoder's two sub-layers together with the residual connections, plus a third one that performs multi-head self-attention to the output of the encoder stack. The stack of decoders processes the encoder stack's output representation, and in the final output sequence, it is converted into word probabilities. Then, the loss function uses this output together with the training data to calculate the gradients through back-propagation.

2.2.3 Self-Attention

Self-attention is the mechanism by which the representation of a sequence is computed by relating/attending to the different words in the same sequence. This allows the model to focus on other words in the input that are more closely related to the word being processed.

The self-attention layers in the encoder and decoder have different behaviors. The encoder is bidirectional. This means the self-attention layer attends to words/tokens both on the left and right at the same time. Contrary to this, the decoder is only allowed to attend to earlier positions of the output sequence - this means the decoder stack can only look at previous words (words on the left) of the current word being processed. This type of self-attention is called masked self-attention due to the "masking" of the tokens to the right of the current position. So, in order to predict a token for position i, we can only depend on the known predicted tokens to the left of it - having a unidirectional behavior.

According to Vaswani et al. [3], the attention function is essentially a mapping of a query and a set of key-value pairs to the output. In this case, the queries, keys, and values are all vectors. A weight is assigned to each value by computing a compatibility function of the query and key, and the end result is the weighted sum of the values.

In an attempt to offer insight into these 3 components, we can imagine the Query as a specific word in a sentence, and the Keys as the remaining words, while their Values could be viewed as the representation of their "meaning" in that context. In other terms, a specific Query is multiplied with the Keys to obtain a "map" that represents how important every word is to the Query word. Then, this resulting "map" is multiplied with the Values to obtain a final matrix that takes into account their importance when compared to the original Query word. The transformer calculates the attention using **Scaled Dot-Product**

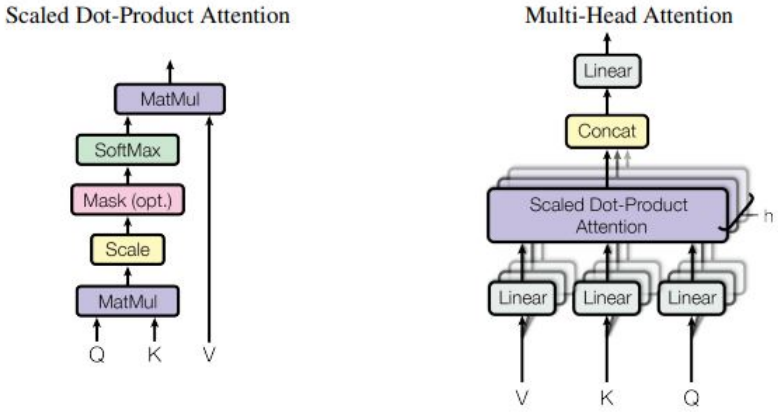

Figure 2.3: Scaled Dot-Product Attention and Multi-Head Attention representation (respectively) (Vaswani et al. [3]).

Attention. The dot products between the query and all keys are calculated, divided by the square root of the dimension of the vectors and then a softmax function is applied to obtain the weights of the values. The attention function is calculated simultaneously, in

practice on a matrix Q which is a set of queries - consequently, the keys and values are also bundled in matrices K and V respectively.

This mechanism is then applied in parallel, where each attention head will produce their own attention matrix. Since the feed-forward Layer is not expecting multiple matrices, the output matrices of the attention heads are concatenated and multiplied by a weight matrix (learned by the model), producing a single matrix with the attention information from all the heads - this is called **Multi-Head Attention**, as shown in Figure 2.3

2.3 Tokenization

Models don't work directly with raw text data. This data first has to be encoded so the models are able to process them accordingly. To put it simply, tokenizers are responsible for preparing the input for the models by turning a string of text into tokens that are represented by integers - these are not embeddings. These units can be words, subwords or even single characters. Having these three types of possible subunits, we need to choose which ones would be best suited to our use cases. Since our NLP transformer-based models work with text on a word-level basis, it makes sense that we aim for tokenization techniques that try to split our text data into single words or sub-words - essentially pieces that allow us to reconstruct said words.

2.3.1 Preparing a Tokenizer

Tokenizers need to be prepared, and this type of setup can be explained as a statistical process. This process looks through an entire corpus to try and identify which words or subwords are the most relevant. How to choose which words or subwords we want is entirely dependent on the algorithm we choose, and since tokenization algorithms are statistical and deterministic we will always get the same result if we apply the same algorithm to the same dataset or corpus.

We can then infer that, for different datasets or corpus, we need different tokenizers arranged for each one - for example, for the same model but with different datasets, we will need to prepare different tokenizers, since using only the same one could result in sub-optimal tokenization - this means we could create a dictionary that in reality would not represent the statistical distribution of our tokens present in our corpus.

2.3.2 Text Pre-processing

Usually, before preparing a tokenizer, we may want to clean our text (corpus) to make sure that we remove unnecessary or unwanted characters - whitespaces, accents, etc. This is called Normalization [1].

One other important factor to take into account is if the words or characters are upper or lowercase. Training a tokenizer on a corpus with both upper and lower-case characters

[1] https://huggingface.co/course/chapter6/4 - Presents a good visualization of this process.

will most likely lead to the production of different embeddings for the same character or word, for both its upper and lower-case versions. So lowercasing our text data may be another step we may want to perform before pushing it into our tokenizer preparation.

2.3.3 Dealing with out-of-vocabulary tokens

Originally developed by Schuster and Nakajima [24], WordPiece is a tokenization algorithm designed to split words into subwords (pieces) with the goal of balancing vocabulary size with the ability to represent words that are not part of said vocabulary - this is the main issue when attempting to prepare a tokenizer for every word that can exist. In other words, it is a data-driven approach that maximizes the LM likelihood of the training data, achieving a compromise between the flexibility of characters and words (Wu et al. [25]).

In WordPiece, words are first tokenized into subwords using a combination of heuristics, such as the frequency of the subwords in the training data, and the degree of overlapping with other existing words. These subwords are then combined into a vocabulary. This vocabulary is a list of all the unique subwords in the training data.

WordPiece and subword tokenizers in general are able to overcome the limits of algorithms that are only word-level based. Working only at word level limits the models according to Sennrich, Haddow, and Birch [26], as they become unable to generate unseen words due to the lack of morphological modeling, like suffixes or prefixes. Also, certain word relations that may be common to multiple words, may only be mapped to a short few. For example, the model may learn the relations between the words 'tall' and 'taller', but not 'short' and 'shorter', even though they have the same adjective degree difference.

These issues are addressed due to this algorithm's capability of creating a dictionary with both words and character aggregations that are able to represent compositional word components like suffixes. WordPiece is not the only algorithm capable of this, so is the Byte-Pair Encoding (BPE) algorithm. This is one of the most well-known algorithms for tokenization and it is another formula to target the same out-of-vocabulary problem WordPiece does, while also working at subword level.

The BPE algorithm is fairly similar to WordPiece, except instead of using heuristics to split words into subwords, BPE works by representing each word in the data as a list of the characters that compose them and iteratively count and merge all the symbol pairs, replacing the most frequent ones with their aggregation: *('t','a') -> 'ta'*.

2.4 Transformer-based Language Models

Thanks to the transformer's architecture focusing on the parallelization of data, more data can be trained in less time. Together with the different encoder and decoder blocks, new models were developed, using these blocks as their foundation depending on their objective. Most of the following models that will be discussed center their goals on Natural Language Processing tasks, where in most cases you need the bi-directionality of

the encoders to be able to understand the context of words and sentences since trying to understand these relations by looking in just one direction may be not ideal. However, some only use the decoder stack, for purely generative approaches.

2.4.1 Deep Bidirectional Transformer - BERT

BERT was proposed by Devlin et al. [4] as a 110M parameter language model that trains deep bidirectional representations from unlabeled text - taking into account both the left and right context of the tokens and across all of its layers. Historically, the previous models were only able to read input sequentially, either from left to right, or right to left, but never at the same time. BERT changed that by using only the encoder blocks from the Transformer, which is what allows the model to be fully bidirectional.

BERT can be pre-trained on a large corpus of unlabeled text, and then fine-tuned with just one additional output layer for a specific downstream NLP task, like question answering or language inference - where the weights are updated but without significant modifications to the architecture.

2.4.1.1 BERT's Pre-training Data

The corpus used to train BERT was constructed from the English Wikipedia and the BooksCorpus for, respectively 2500M and 800M words. The extracted data is only composed of text passages, avoiding artifacts like lists tables, and anything else that is not on a document level. All of the collected training data is unlabeled, making the learning process fully self-supervised.

2.4.1.2 Input and Output Representations

The input tokens are obtained via a WordPiece-based Tokenizer, with a 30k vocabulary. These tokens are bundled with special tokens when sent to the model like the *CLS* (start of sentence or classification) token, the *SEP* token that separates pairs of sentences, or the *MASK* token, whose use is explained in the next section 2.4.1.3. BERT's inputs also aim to represent both a single or a pair of sentences - these can be actual sentences or a large sequence of text - in a single token sequence (a token being a word).

The input is the sum of the token embeddings, the sentence embeddings (to indicate to which sentence it belongs), and the positional embeddings, as observed in Figure 2.4.

2.4.1.3 Pre-training BERT

To take advantage of its bi-directionality, BERT is pre-trained on two unsupervised tasks, whose sum of the mean equals the training loss: a **Masked Language Model** (MLM) and a **Next Sentence Prediction** (NSP) training objectives.

- **Masked Language Model** - For the first objective, the model masks 15% of the input tokens randomly and then attempts to predict only the masked words, instead of

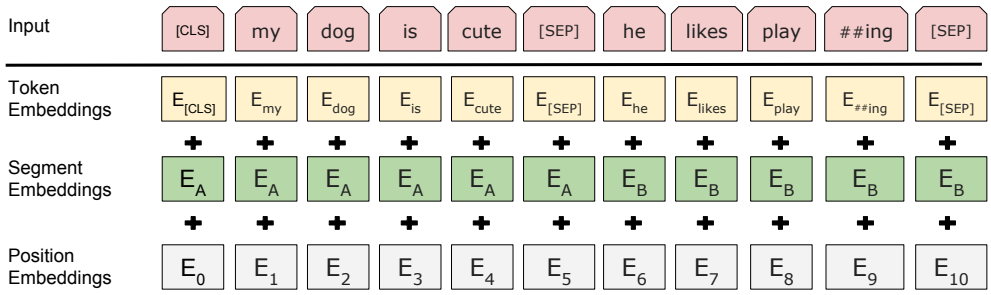

Figure 2.4: Representation of BERT's input. (Devlin et al. [4])

generating the entire input from scratch - for each chosen token, there's an 80% chance it is replaced by a MASK token, a 10% chance it's replaced with a random token, and for the remaining percentage the token is not changed at all. The reason for sometimes not replacing the token with the MASK token is to reduce potential mismatches between fine-tuning and pre-training since this token is unique to pre-training and won't appear in any fine-tuning tasks. Then, the model only predicts the masked tokens (with cross-entropy loss), instead of attempting to generate the entire input from scratch.

- **Next Sentence Prediction** - The second objective is essential for tasks like Question Answering or Natural Language Inference. NLI is a general task where you check if a sentence (hypothesis) logically follows another one (premise), and can be labeled as entailment or contradiction for example. The Next Sentence Prediction task then allows the model to understand the relationship between two sentences. To implement this, two sentences or large sequences of text are sampled from the data with a combined length <=512 tokens, sentence A and B respectively, for which sentence B has a 50% chance of being a random sentence (being labeled as NotNext) instead of the actual next sentence. The model receives these two sentences as input.

BERT was originally trained with a batch size of 256 sequences (256 sequences * 512 tokens), for 1M steps. This is approximately 40 epochs over the entire corpus. The size of the batch of samples that is fed to the model and the number of steps, for example, are not the only hyperparameters we need to look out for when training such a language model.

The optimizer and its learning rate are the first ones we focus on since the optimizer is the algorithm responsible for updating the parameters of the model to reduce the loss function over time. The learning rate associated with the optimizer is what determines the size of the step at which the optimizer updates the parameters - it can be described as a value that multiplies the gradient of the loss with respect to the parameters - if this value is too large or too small, the optimizer can overshoot the optimal solution or cause very slow convergence, hence why setting an appropriate value is important for optimization. BERT in specific uses an Adam optimizer (Kingma and Ba [27]) - this optimizer differs from the normal gradient descent optimization by leveraging the idea of momentum, an adaptive learning rate, and memory decay which allows it to forget possible past noisy gradient

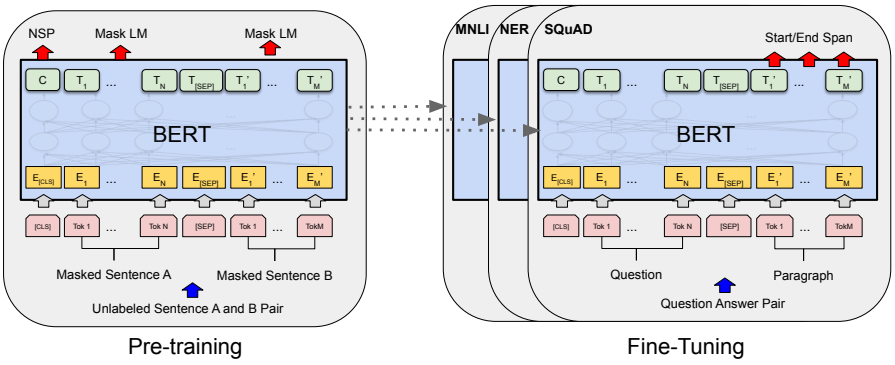

Figure 2.5: BERT overview of pre-training and fine-tuning workflows. (Devlin et al. [4])

information - making it usually the main choice when choosing an optimizer. A learning rate of 1e-4 was used, with an L2 weight decay of 0.01 and a learning rate warm-up over the first 10k steps (meaning the learning rate will increase linearly over these initial steps).

Weight decay is another relevant concept since it is a regularization technique that adds a penalty to the loss function proportionally to the model weights - with the goal of simplifying the model in an attempt to reduce overfitting. The optimal values for these types of hyperparameters are not set and are deduced from experimentation.

2.4.1.4 Finetuning BERT

It's important to assert that a deep learning model like BERT is further enabled by transfer learning. **Transfer learning** is the ability to use the knowledge the model learned from one or more tasks (usually during pre-training) and apply it to another downstream task via fine-tuning. In concrete, fine-tuning a model takes advantage of the base parameters learned from pre-training, and adjusts them so the model becomes capable of fulfilling whatever downstream tasks it is finetuned for.

The main difference between fine-tuning and pre-training is that fine-tuning is much faster since it uses much fewer data and smaller batch sizes as well as a smaller number of epochs and different learning rates. It is also important to notice that, for some fine-tuning tasks, the data used is labeled, unlike the data used for pre-training.

To show how versatile these models can be, Devlin et al. [4] fine-tuned BERT for several downstream tasks. First, the model was evaluated on the **GLUE** benchmark (Wang et al. [28]). GLUE is a large and diverse collection of natural language processing and understanding tasks, containing natural language inference tasks, and binary classification tasks, among others. Then both versions **SQuAD v1.1** (Rajpurkar et al. [29]) and **SQuAD v2.0** (Rajpurkar, Jia, and Liang [30]), which are large collections of question/answer pairs. Given a question and a passage from Wikipedia, the objective is to predict the answer text in the passage. The last task BERT was evaluated on was **SWAG** (Zellers et al. [31]). SWAG is a dataset containing pairs of sentences where the model has to choose the most likely continuation among other choices given a sentence.

2.4.2 Decoder Only Transformer - OpenAI GPT-2

Shortly after BERT, the 1.5B parameter OpenAI GPT-2 model was introduced (Radford et al. [14]). GPT-2 uses only the Transformer's decoder blocks behind the scenes, contrary to BERT. This makes the model unable to look both ways when it comes to text, considering the decoder blocks use masked self-attention - making it unidirectional and auto-regressive in nature since an output word is dependent on the words that came before it. We can therefore classify this model as a sentence-generative language model, since its goal is to generate grammatically fluent and correct text, according to the probabilities it calculates using what it learned during training.

GPT-2 was also shown to have some zero-shot capabilities.**Zero shot learning** is when the model is capable of understanding and performing a task when no examples of it were used during training. This capability is supported by the model's inner nature of being able to predict the next word given a sequence of previous words. A very simple example would be a simple Question-Answering (QA) task. If the model did not see QA text data during pre-training, we can try to prompt the model with both a context and a question, in an attempt to have the model generate the correct answer. This answer would be the result of the calculated probabilities, given the text/prompt we fed the model.

2.4.2.1 GPT-2's Pre-training Data

GPT-2 was trained on WebText, a 40GB dataset, containing over 8 million documents. These documents were scraped from the web, and the goal was only to obtain text that had been filtered or curated by humans in order to ensure the quality of the content.

2.4.2.2 Input and Training

Just like BERT or any other language model that uses Transformers, the input text must be converted into tokens. For GPT-2, the input text is tokenized via Byte Pair Encoding-based tokenizer (more on this in Section 2.3), with a vocabulary size of 50257 tokens.

The pre-training objective is fairly simple. Due to the auto-regressive nature of GPT-2, the training goal is to predict the next word, given a sequence of prior words. Compared to BERT, a larger batch size of 512 was used, with an increased context size (max sequence length) of 1024 tokens. According to the referenced paper (Radford et al. [14]), without extra training or fine-tuning, GPT-2 was able to achieve state-of-the-art zero-shot performance results **at the time the model was first released**, for a number of tasks: summarization, translation, reading comprehension and question answering.

2.4.3 Encoder and Decoder: Best of Both Worlds - T5

While previous models were based on only one of the Transformers' stacks, meaning they either used the encoder or the decoder, the Text-To-Text Transfer Transformer (also known as T5) used both stacks in its architecture. Proposed by Raffel et al. [5], its goal

is to introduce a new unified framework, able to convert any text-based problem into a text-to-text format - using text as input and producing text as output. Being able to cast problems into this format allows the model architecture to stay the same, using the same training methods (same loss function, hyperparameters, etc.) for whatever task we choose.

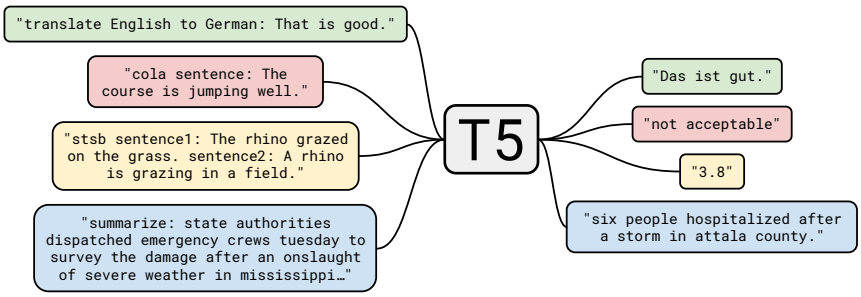

Figure 2.6: Diagram of the T5 framework[5].

2.4.3.1 T5's Pre-Training Data

T5 was trained on a cleaned version of the Common Crawl[2] data - named the Colossal Clean Crawled Corpus. The Common Crawl is a nonprofit organization dedicated to building a large web-extracted text dataset. This is done by crawling the web and extracting only the text, removing non-text items like HTML tags. Due to its nature, it's bound to be full of raw text that can be used to train language models, but it may also contain offensive text, duplicated text, text from "menus" or "sidebars", etc. Raffel et al. [5] leveraged this data by defining some criteria (removing pages with code, "Lorem Ipsum" placeholder texts, offensive words, etc) that allowed the authors to filter the Common Crawl data. The final result is a 750GB clean text dataset used for pre-training.

2.4.3.2 Input and Output Representations

Before feeding a sequence input to the model, a task-specific tag (a prefix) is concatenated to its beginning, as we can observe in Figure 2.6. For example, in order to ask the model to perform a translation task, one would add the prefix "translate English to Portuguese:" to the input sequence. For tasks like text classification, the model is able to output a single prediction label given the input sequence and the correct prefix. According to its authors, the prefixes are essentially hyperparameters, but it was observed that changing them had limited to no impact so no other experiments were performed on this topic.

2.4.3.3 Pre-training T5

The T5 model is based on the size and configuration of the base BERT model stack, with the exception that the T5 includes a decoder stack. The T5 boasts a total of 11 billion

[2]https://commoncrawl.org/

parameters, around 7.8x the parameters used by BERT(base). According to the authors, there are two equally viable ways one can train a T5: an unsupervised and a supervised method - the latter haven been chosen as the most relevant one.

T5's unsupervised training goal is inspired by BERT's denoising masked language training objective. T5's objective randomly chooses and removes 15% of the tokens of the input sequence, placing "sentinel tokens" where these removed words were located. These removed words can be in a sequence, being treated as a "span" (with a maximum length of 3), and a span is replaced by a single token instead of replacing every word in it. These tokens also each have a unique ID for the input sentence. Afterward, the model "surrounds" the dropped-out tokens with the sentinel tokens, and attempts to predict them, as seen in Figure 2.7.

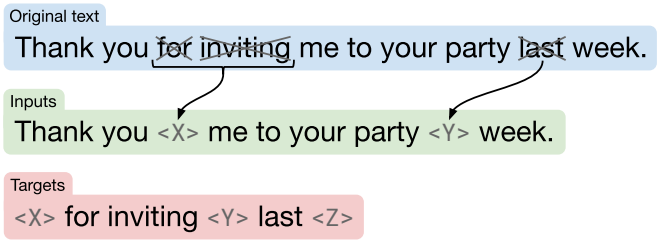

Figure 2.7: Diagram of the T5 training objective. (Raffel et al. [5]).

The supervised method used by the T5 follows multi-task pre-training approach, since according to its authors it has the advantage of being able to monitor and evaluate downstream task performance, instead of just doing it during fine-tuning - not forgetting the authors declare that pre-training the model only on the unsupervised task and then fine-tuning on the supervised tasks is equally viable. Besides this, this type of approach can leverage cross-task data, leading to more general representations of knowledge (Liu et al. [32]). **Multi-task training** is when a model is trained on several tasks at the same time with the goal of performing multiple downstream tasks. In order to do this, multiple datasets are joined and mixed together - mixing supervised and unsupervised tasks together, and batches are composed of samples from the datasets.

In practice this is simplified by the T5's way of casting problems into a text-to-text format since if we wanted to train the model for a translation task, we would only need to feed the model *"translate English to Portuguese: Welcome to Lisbon"* and its corresponding labels *"Bem vindo a Lisboa"*. The model is able to easily learn which task to perform according to the prefixes of the examples that are fed to the model when training.

The T5 uses a cross-entropy loss function and a maximum likelihood objective, with "teacher-forcing" enabled (Che et al. [33]) - replacing the output of previous steps in the network with a ground-truth or an expected value. This is done independently from the task that we choose to train the model on.

For the actual pre-training (using the supervised method), the model is trained on approximately 1M steps with a maximum sequence length of 512 and a batch size of 2048.

Table 2.1: Amount of seen data in training between BERT(*base*) and T5 11b.

Model	Dataset Size	Tokens (pre-training)	Num.Steps	Batch Size	Seq.Length
BERT(*base*)	~27GB	128B	1M	256	128-512
T5 11B	~750GB	1T	1M	2048	512

This results in an estimate of 1 trillion tokens seen, much more than BERT, as observed in Table 2.1. This increase in seen tokens and batch size is due to the increase in scale performed according to the authors, since their initial baseline model was about the same size as BERT, and ended up being scaled up, reaching 11B parameters.

The learning rate starts at 0.01 and is constant during the first 10k steps, after which it exponentially decays according to an "inverse square root" learning rate schedule (Zhang et al. [21]) $1/\sqrt{max(n,k)}$ (n is the current iteration and k is the number of warmup steps).

2.4.3.4 Fine-tuning T5 & Downstream Tasks

For fine-tuning, sequence length remains the same while the learning rate is changed to a constant value of 1e-3, but the batch size is decreased according to the task and benchmark to be fine-tuned on. For example, the original T5 was fine-tuned individually for the GLUE and SuperGLUE (Wang et al. [34]) (another bigger version of GLUE with more difficult tasks) benchmarks, and since the original batch size is too large and susceptible to having a lot of dataset repetition due to the number of available samples, it was decreased to 8.

The chosen downstream tasks to evaluate the T5 mainly include summarization, text classification, translation, and question answering. Summarization tasks were evaluated on the **CNN/Daily Mail** (Nallapati et al. [35]) benchmarks, question answering was evaluated on the **SQuAD** (Rajpurkar et al. [29] and Rajpurkar, Jia, and Liang [30]) benchmark, translation was evaluated on the **WMT** (Bojar et al. [36]) benchmark and text classification evaluation was done through the **GLUE** (Wang et al. [28]) and **SuperGLUE** (Wang et al. [34]) benchmarks.

2.4.3.5 Causal with prefix attention

It is also important to know that a slight but important change was made to the architecture, namely to the self-attention masking pattern. While BERT uses a fully visible (bidirectional) pattern - meaning a word can attend to every other word in the input at every generated output word - the T5 implements a "causal with prefix" pattern due to the decoder stack's causal nature (unidirectional). This attention pattern allows a word to fully attend to a portion of the input sentence, independently from it only being allowed to attend to previous tokens.

2.5 Ahead of GPT-2

It is important to know that a much larger and improved version of GPT-2 has been proposed - the GPT-3. GPT-3 (Brown et al. [15]) is a 175B parameter model and was trained on much more data than the GPT-2 model. The main objective of this model was to show that scaling up large models can lead to relevant improvements in task-agnostic and few-shot performance. **Few-shot** performance is when a model is fine-tuned with a small amount of labeled data for a given task and expected to generalize the knowledge it learned for other never-seen tasks or examples - much like a human can do unseen tasks by using knowledge gained from other tasks. This is similar to the **zero-shot** case, except in this case, the model has seen examples of the task at hand, either in the training data or in the prompt / text we feed to our model - and once again this ability if enabled by these model's prediction of the next most probable words.

More recently, the authors of GPT-3 optimized this model for dialogue and launched InstructGPT (Ouyang et al. [37]), from which ChatGPT was based and developed. The rise and success of this new tool proved the massive capabilities and potential that large language models (LLMs) like this can achieve when performing a wide range of tasks. Since neither InstructGPT nor GPT-3 are currently not open-source, their training details are left undisclosed. Tools like the mentioned ChatGPT currently use newer iterations of these models, such as GPT-3.5 and GPT-4, both available to be used via a paid API.

Inspired by the GPT family of models, a large group of researchers collaborated and developed a 176 billion-parameter, decoder-only language model, called BLOOM (Workshop et al. [38]). It was trained on 46 natural languages and 13 programming languages, achieving results comparable to GPT-like models. Thorough experimentation was made with this model with the end goal of providing valuable insight into how a large language decoder-only model like this can be trained, as well as to facilitate future research and development of applications using this type of model.

2.6 Knowledge generalization in LMs & other recent improvements

More recently, new and diverse techniques have been developed and proposed to increase the performance of very large transformer-based language models.

Recall that Brown et al. [15] have shown with GPT-3 that scaling up a language model in size and parameters can lead to great improvements in the ability to generalize knowledge - especially in increased performance for few-shot tasks, further expanding the capability of models to perform well in tasks they've seen a few of during training (few-shot), or none at all (zero-shot). But there is still some space for improvement before reaching the same performance for zero-shot tasks, and human performance in tasks that require reasoning like question-answering tasks for example.

2.6.1 Turning LLMs into Zero-shot learners

Propelling the ability of LLMs to generalize knowledge is one of the principal objectives of this area of research since if a model is able to perform tasks that it has not yet seen, its range and possibilities are increased.

FLAN (Wei et al. [6]) is a technique that was proposed, which considerably improved the performance of fine-tuned models on zero-shot performance - even surpassing GPT-3's performance in this matter. This is achieved via instruction tuning, where the model is provided with a set of instructions related to the task to be learned.

To evaluate this method, a large number of datasets, each dedicated to a certain downstream task, are grouped. Let's say our data collection has datasets to train the model for tasks X, Y, and Z. To finetune the model, we randomly pick the datasets that are used for tasks X and Y (we could choose Y and Z for example) and leave out the datasets for task Z for evaluation. Naturally, the datasets for task Z are never seen during the finetuning process. The idea is that if the model learns how to follow instructions for certain tasks, it will become capable of following instructions from unseen tasks.

In practice, a handful of templates are created for each dataset. These templates are used when generating the input needed for the model, with the correct instructions and their variations (different phrasing for the same task and instruction). Figure 2.8 demonstrates how these templates are prepared.

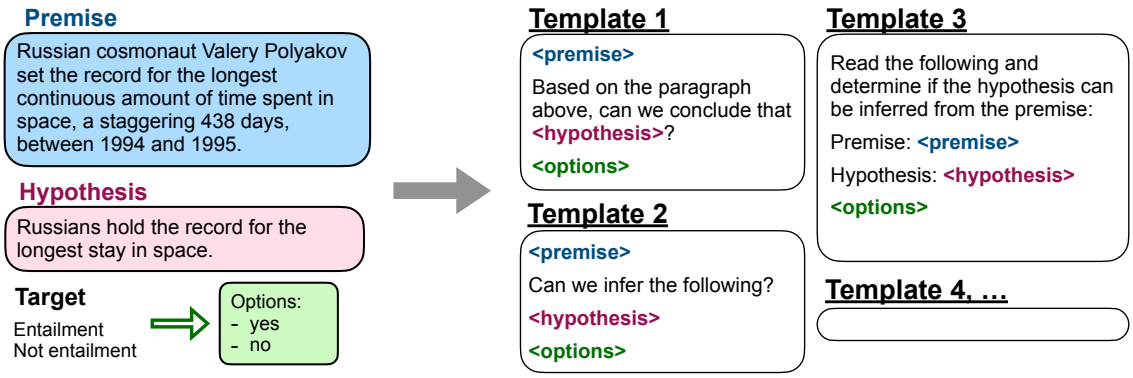

Figure 2.8: Examples of templates for a language inference task. (Wei et al. [6]).

2.6.2 Prompting reasoning in large models

Among the several capabilities previously seen, GPT-3 (Wei et al. [39]) also proposed that, for a large enough language model, complex reasoning abilities can be unlocked naturally through adding a chain-of-thought reasoning prompt. This prompt contains an example of the necessary reasoning for a certain problem in the form of natural language.

Prompting is when, instead of fine-tuning a model for different tasks, we feed some exemplars with a certain input-output format that shows how the task is performed. We can observe an example of arithmetic logic with prompting in Figure 2.9.

More work on prompting has been made, and due to large language models being able to perform quite well in few-shot environments, Kojima et al. [7] came up with a Zero-shot Chain of Thought prompt. This prompting technique differs from the previous one. It is split into a two-step prompting pipeline. The first one uses the prompt "*Let's think step by step*" to extract from the model the necessary reasoning to answer a question. The second step uses the reasoning text obtained in the previous step as a prompt to have the model output the final answer. Refer again to Figure 2.9 for a visualization of the differences between few and zero-shot Chain of Thought techniques.

The importance of these techniques lies in the ability of large language models to reason, making them able to decompose problems that can be solved via natural language, as well as allowing the researchers to see how the model behaves and rationalizes.

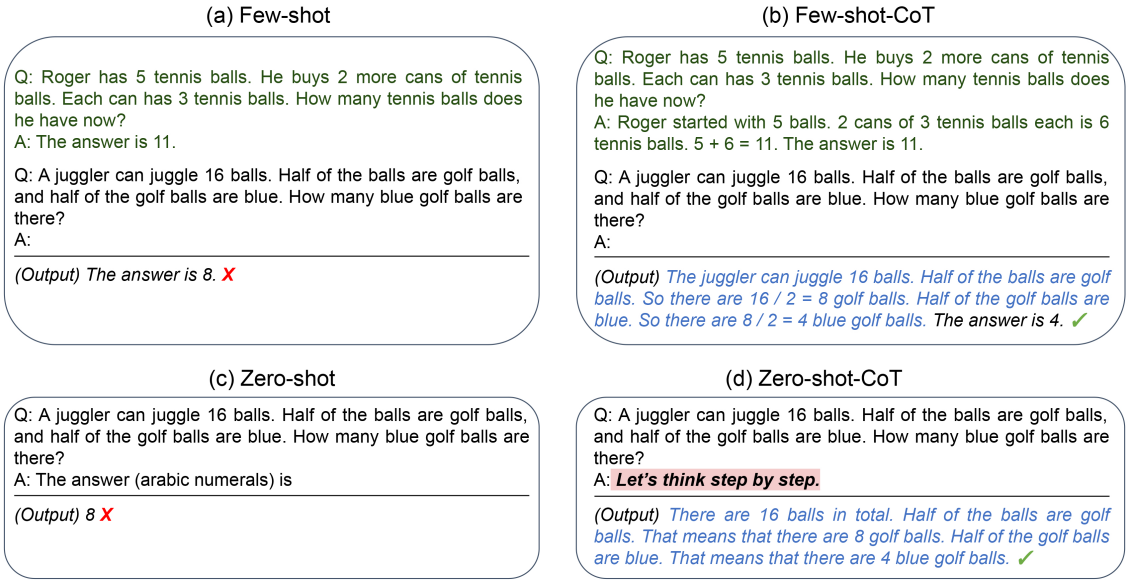

Figure 2.9: Conceptual differences between Few-Shot CoT and Zero-Shot CoT. Adapted from Kojima et al. [7].

2.6.3 Scaling Models, Data & Tasks

Some of the success of the previous techniques is due to them being built upon very large language models and their ability to generalize knowledge. Most notably, we have PaLM (Chowdhery et al. [40]), a 540B parameter language model (transformer-based), inspired by GPT-3's (Brown et al. [15]) takeaway that increasing model scale leads to better performance in most natural language tasks. PaLM, at the time of its proposal, achieved state-of-art results across hundreds of natural language tasks, especially in the same NLP tasks the GPT-3 model was benchmarked on, continuing the improvements gained from scaling. These tasks include the ability to mathematically reason, and code generation.

However, scaling the model size by itself is suboptimal. Hoffmann et al. [41] found that scaling the model is only optimal when you scale the data equally: if the number of the parameters of a model is tripled, the number of tokens needed for training should also

Table 2.2: Model sizes.

Model	Parameters	Layers	Attention Heads	Embeddings Dimensionality
BERT(*base*)	110M	12	12	768
BERT(large)	340M	24	16	1024
GPT-2	1.5B	12	12	768
GPT-3	170B	96	96	12288
T5 (initial baseline)	220M	12	12	768
T5 11B	11B	24	128	768
Palm 8B	8B	32	16	4096
Palm 540B	540B	118	48	18432

be tripled. Looking again at PaLM as a prime example of a very large LM, its enormous training dataset consisted of approximately 780 billion tokens - half of the dataset being social media conversations, while the other half is a mixture of webpages, books, news, some code, and Wikipedia pages.

The evolution of the size of language models, specifically the ones that we approached in this work, can be viewed and compared in Table 2.2. Some "intermediary" versions of the models were removed to reduce clutter, while sometimes including the initial models' baselines and their largest, best-performance achiever versions.

Putting the notion of increasing scale to increase performance together with Flan (Wei et al. [6]) and the key ideas from chain of thought prompting, we can enhance finetuned models to a greater extent by scaling the number of tasks. Both T5 and Palm achieve better scores in performance when compared to their regular finetuned versions when finetuned via Flan for 1.8k tasks - respectively Flan-T5 and Flan-Palm (Chung et al. [8]) - outperforming them in chain of thought, zero and few-shot abilities without any degradation. The fact that the performance on both those models is improved, taking into account the different architectural natures (one uses both encoder-decoder stacks while the other only uses the decoder), shows that instruction finetuning is generalized across models while improving usability.

We've seen these very large-scale models work with relatively large batch sizes when compared to models like BERT. Recently, You et al. [42] proposed a new technique based on a layerwise large mini-batch approach, that enables training a model with batch sizes up to 32k, while maintaining efficiency in both execution time and memory usage. Such a technique was tested on BERT, proving it could be used to massively increase the amount of information a model can see during a step in training.

2.7 Assessing Language Models Quality

Having a broad number of available and proposed language models, one has to ask how can we compare these models. What baselines do we establish? What metrics and concrete values can we use to observe improvements or degradation in performance?

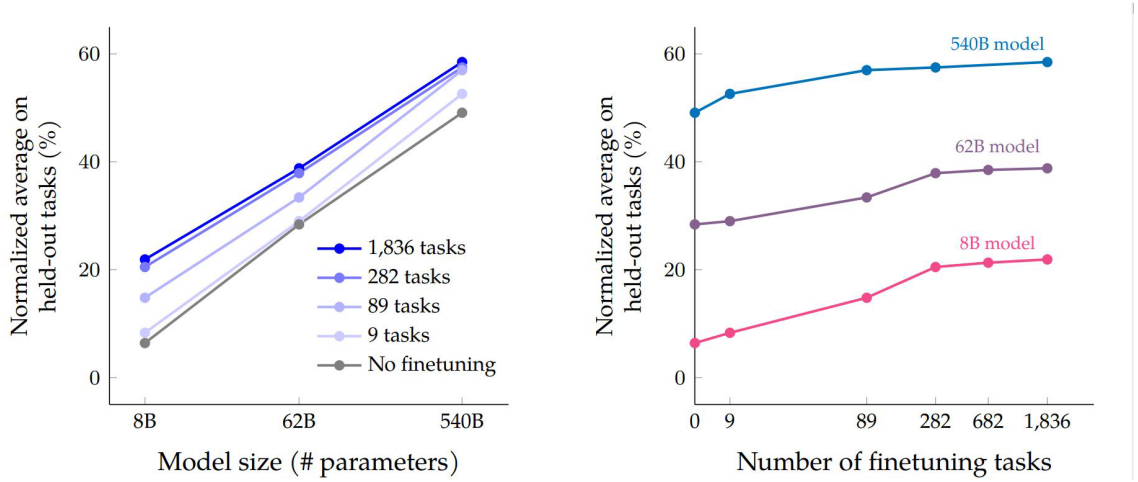

Figure 2.10: Graph demonstrating how performance evolves regarding model size and the number of finetuning tasks. Performance was evaluated on a few-shot environment with prompting on multi-task datasets. Taken from Chung et al. [8].

The first step to ensure a fair comparison of results is using the same data. Specifically using the same test sets for each one of the benchmarks we choose to evaluate a model on. These test sets are composed of data not seen by the model so they can be used to correctly evaluate them, without incurring biased results.

After establishing the grounds for comparability, other differences and similarities in training methods must be taken into account when discussing and analyzing results. Model architecture and size are two examples that can influence the results. For example, Raffel et al. [5] compared BERT to their T5 model, to see how the increase in scale and difference in architecture affects performance on a set of chosen benchmarks. Chung et al. [8] also performed experiments that observed a performance increase when increasing a model's scale (maintaining the architecture), as well as an increase in data size and diversity. Refer to Figure 2.10 for a visualization of how performance increases according to size for example.

2.7.1 Quantifying performance with metrics

Firstly, it is important to define that there isn't a general best metric for evaluating NLP models. The appropriate metric to be used would depend on a specific task or goal the model is trying to achieve. We've seen what datasets exist for finetuning and evaluating a language model, yet their performance results need to be quantified - that is the metrics' role. Depending on the dataset's task, different metrics may be used.

Besides the metrics' usefulness in evaluating and in assessing a model's quality, they also come in handy during model training. As the training loop progresses, checkpointed evaluations can be made to see how a certain metric or value is evolving, reflecting the model's own evolution. For deep learning models in general, we would intuitively assume that the value of the loss is what we would want to monitor to see if the training is going

well, but it is not a metric. Loss is an error measurement that quantifies how well the model can predict the correct output, and during training, we aim to minimize this value in hopes of improving the model's ability to make better predictions on unseen data. Different functions can be used to compute it, like the cross-entropy loss function used by the T5. This function finds the differences between the actual and predicted probabilities, applying a log function over the models' probabilities. This leads to penalizing confident wrong answers more than unsure wrong answers - meaning the model may be more motivated to learn from those confident but incorrect predictions.

2.7.1.1 Perplexity in text generation

Since language models work with sequences of words, we want to try to predict the likelihood that a model will output a specific sequence. This is where **Perplexity** comes in. Perplexity is calculated as the exponentiated average negative log-likelihood, as follows:

$$Perplexity = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i|w_1,w_2,\ldots,w_{i-1})}$$

N is the number of words in a sample, and P(*w_i | w1..wni-1*) is the probability of the *ith* word given the previous words in the sequence.

In a way, perplexity measures how confused ("perplexed") the model may be when asked to generate a certain sequence. The lower the value (the less confused), the better the model is at predicting the current sample, hence better performance. The main reason why perplexity is useful is that it takes into account the overall probability of a sequence being generated, instead of the prediction of a single word - it is best used for the evaluation of language generation tasks.

2.7.1.2 Evaluating text classification

A simpler but well-known metric is **accuracy**. Accuracy is a measure of how well the model is able to predict the correct output for a certain input and can be defined by the number of correct predictions made by the model divided by the total amount of predictions. It is mostly used for text classification tasks that require the model to output a single label, like sentiment classification - this task is when the model attempts to predict the emotion (positive, negative, or neutral for example) of a text.

An improved way of measuring the performance of a model in classification tasks is to use **precision** and **recall**. Precision is the fraction of correct predictions, out of all positive predictions made. A high value means the model is good at avoiding false positives. Recall is the fraction of correct positive predictions, out of all actual positive cases, and a high value means it is good at detecting positive cases. Maximizing both these metrics may not be possible, so in order to counter this difficulty, we can use the **F1 score**.

This metric combines precision and recall in an attempt to balance their trade-offs. If a model has a high F1 score value, it means it has both a high recall and precision, while if

it was a low score, it can either have low recall or low precision. The F1 score is defined as the harmonic mean of precision and recall, as follows:

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall}$$

2.7.1.3 *N-gram* based metrics

The BLEU (Bilingual Evaluation Understudy) (Papineni et al. [43]) is a metric used to measure the quality of translated text. BLEU is calculated by comparing the translated text with several reference translations, computing its score based on the number of contiguous sequences of n words (*n-grams*) that are shared between the reference translations and the model's translated text. Mathematically, it is a weighted geometric mean of the precision of the translated text for different levels of *n-grams*. A low value indicates that the translated text does not correspond to the reference translation, while a high value means the opposite. More importantly, its computation is fairly fast and it correlates with human evaluation of translated texts.

Summarization is another prominent task in NLP. Summarization presents similar difficulties to translation - for a given document, multiple reasonable, fluent, and complete summaries can be written, so in order to evaluate the quality of a written summary one needs to compare it to a human-written "ground-truth" summary. This comparison can be measured via ROUGE (Lin [44]) (Recall-Oriented Understudy for Gisting Evaluation), which is conceptually similar to BLEU (Papineni et al. [43]) since it uses the same idea to measure similarity via word and subword overlapping between the model's output summary and an (ideal) human written summary. Due to their close and similar nature, BLEU and ROUGE are not exclusively used for, respectively, translation and summarization tasks, and can be used interchangeably for each task.

2.7.2 Fine-tuning on Downstream Tasks & Benchmarks

Throughout this work, we've talked a lot about "downstream" or "natural language" tasks. These tasks are simply possible use cases that we can perform with language. We can write a cohesive sentence (word generation), we can analyze two sentences and see if they're related and make sense (inference or entailment), or we can do something as simple as translating text or answering a question. These are all valid possible downstream tasks that we may want our models to excel at.

Nowadays, most models similar to the ones we've seen, are pre-trained and posteriorly fine-tuned and evaluated on a collection of NLP tasks, since the training objectives for pre-training are not appropriate or directly related to an actual task. For example, BERT's pre-training objective is predicting the correct word for masked tokens - this does not represent a real-life NLP task, and this is where finetuning comes in.

Evaluating different models for the exact same task requires using the exact same dataset and metrics. These datasets that are used to finetuned and evaluate models are

Table 2.3: Several benchmarks used to finetune and evaluate models for different NLP tasks and their brief descriptions.

Benchmark	Task Objective and Description	Metric
SQUaD v1.1	Collection of question-answer pairs whose objective is to teach a model how to predict an answer for a question.	F1 Score or 'Exact Match'
SQUaD v2.0	An extension of SQUaD v1.1 that adds unanswerable questions so the model can learn how to detect questions that can't be answered.	F1 Score or 'Exact Match'
GLUE	Big collection of several datasets for different tasks that act as benchmarks: sentence inference, semantic equivalence, sentiment classification, etc. It is calculated by averaging the score of each task/dataset in the collection.	Varies according to the task.
SWAG	Collection of sentence-pairs that complete each other to teach the model common sense inference (e.g.: model may learn that a building can't fit in a backpack due to physical size commonsense).	Accuracy
LAMBADA	Collection of narrative passages of which humans can guess the last word only if the entire text is read instead of the last sentence. Evaluates a model on its ability to identify long-distance dependencies in text.	Accuracy or Perplexity
WMT-14	Collection of datasets for translation tasks in multiple categories (news, medical text, etc.). Used to evaluate a model's multi-lingual and translation capabilities.	BLEU
CNN & Daily Mail	Collection of news stories and abstract summaries, with fill-in-blank questions for the model to predict. Evaluates summarization capabilities.	ROUGE

what we call benchmarks. Each benchmark is based on a specific task and the metrics they're evaluated on depend on the task at hand. Table 2.3 lists only a handful of known and widely used benchmark datasets for a variety of tasks for the English language, with a brief description of what they contain and their objective.

2.7.3 Resources & Environment

To train a language model it needs large amounts of computational power and lots of memory - consequently, it requires lots of powerful hardware. To train these models rapidly and efficiently, GPUs or TPUs (specialized hardware for deep learning domain (Jouppi et al. [45])) should be used since they are better at performing parallel matrices calculations. To use a GPU, we must move the model itself and the data over to the GPU memory, implying the model must fit. Using a gross estimate that each parameter in a model is of type *float32* (4 *bytes* each), the T5 would occupy roughly 44GB of memory - this is just for the model size alone, excluding the extra space needed to allocate for a set of samples (a batch). Using the same estimate, BERT(*base*) would use 4GB.

Depending on the size of the model and the amount of data, training a language model

on a GPU may take hours up to days until it is finished. This amount of time would be exponentially higher if trained on a CPU.

For reference, BERT(*base*) (Devlin et al. [4]) was pre-trained on 4 Cloud TPU Pods totaling 16 single TPUs, and BERT(large) was pre-trained on 16 Cloud TPU Pods (64 TPUs) - each training lasting up to 4 days. Pre-training takes the bulk of the time taken in total to prepare a language model. Fine-tuning is much faster thanks to lower batch sizes and less training data in general and can take only a few minutes up to a few hours.

Taking all these factors into account, the impact on carbon emissions related to the training of large language models has not gone unnoticed. Attempts at quantifying the emissions produced by the required hardware have been made, and recent work motivates future research to find new ways to optimize computation and benchmark utilized resources (Patterson et al. [46] and Yusuf et al. [47]).

2.8 Decoding Strategies

When working with a **generative, decoder-based model**, putting it into practice involves having the model generate text.

To do this, some input (could even be an empty string) is sent to the model. After forwarding our input through it, what we get is essentially a form of conditional probabilities over the entire dictionary tokens. These are not readable text, they're just a representation of the text, so to obtain it we need to **decode** them. There are multiple decoding strategies that we can use, that take a look at the probabilities, and *choose* the next / most probable tokens given our input text - hence the text that is generated can and will depend on which decoding strategies are applied.

Having chosen a strategy, we send the text to the model, and for every newly generated token, we concatenate it to the original text and forward it through the model again until a stop parameter is achieved - like an *end-of-text* token or a maximum number of generated tokens. Every time this is done, we call it a *step*.

2.8.1 Greedy Search

The first decoding strategy we will be going over is greedy search. This is the simplest strategy due to the model choosing only the token with the highest probability during generation. Due to its simplicity, it's computationally more efficient than the rest. However, it can lead to problems like text repetition or very low diversity and content richness. The example below demonstrates how greedy search works - the input is the text we feed the model, the dictionary is the list of tokens that are part of our model's known words/subwords, and the probability distribution is the representation of the output of the model for that given input.

```
Input: "Eu gosto de"
Dictionary: ["eu","gosto","de","saltar","cozinhar", ...]
```

```
Probability Distribution: [0.05, 0.1, 0.05, 0.4, 0.1, ...]
Selected Word: "saltar"
Output: "Eu gosto de saltar"
```

2.8.2 Beam Search

Beam search (Vijayakumar et al. [48]) is another, more complex, decoding strategy. It uses the concept of "beams" to keep track of multiple generation possibilities with variable probabilities. In other words, we keep different sequences of generated text and track the ones with the highest probability of being generated.

Beam search can end up generating text that is more cohesive since it may find "paths" or sequences of words that we would never be able to find using greedy search - meaning we may find combinations of tokens that we would not consider due to the first one not having the highest probability for example. The number of "beams" is how many sequences/combinations we are keeping track of.

The key point in beam search is to find the highest probable *combinations* in order to choose the most adequate tokens for our text generation. Check the example below for a first-step example of a beam search with 2 beams.

```
Input: "Eu gosto de"
Dictionary: ["eu","gosto","de","saltar","cozinhar", ...]
Probability Distribution: [0.05, 0.1, 0.05, 0.4, 0.2, ...]
Selected Words: "saltar","cozinhar"
Sequences tracked:
1) "Eu gosto de saltar"
2) "Eu gosto de cozinhar"
```

2.8.3 Sampling

The last strategy is based on sampling. Their goal is to introduce randomness to the process of selecting the newly generated token by selecting a token through sampling instead of going for the most probable token like a greedy search.

There are different sampling strategies one can use. A **sampling technique with temperature** can be used, where the temperature is a parameter that controls how deterministic the sampling process can be (higher values increase randomness).

A slightly more complex sampling technique is **Top-k sampling**: the top-k most likely words are selected according to the model's probability distribution at each step. Then, the probability mass is redistributed only among these top-k words. The goal of this is to effectively reduce the number of candidate words and make the sampling process more focused on more relevant words.

Top-P sampling is another decoding strategy following a similar logic to top-k sampling. Unlike top-k sampling, where a fixed number of the most probable words are

selected, the top-p sampling technique uses a probability threshold to filter out candidate words. For example, we can choose to select every word with a probability greater than or equal to 80%. So for every step and chosen token, the new next token would be sampled from the few selected ones that respect the threshold.

These sampling techniques can be combined with beam search.

2.9 Towards Portuguese Language Models

Unfortunately, the majority of modern state-of-art language models were trained only in the English language. This English protagonism led to the community focusing mostly on producing sets of exclusively English data. This neglect of other languages encouraged researchers to prepare multilingual versions of existing models. The mT5 (Xue et al. [49]) is a multilingual version of the T5, that was pre-trained using mC4 and fine-tuned for cross-language understanding tasks. mC4 is a variation of the Colossal Clean Crawled Corpus, where the authors accepted text passages from different languages instead of only including English text. XNLI (Conneau et al. [50]) is an example of an evaluation set that can be used to fine-tune and evaluate a multilingual model. Previous to mT5, a multilingual version of BERT - mBERT[3] - was also released. The results of these models prove the capability that language models are able to model different languages and establish common ground between them - called cross-lingual language understanding (XLU). However, even though these types of models are multilingual, they still don't have a significant focus on European Portuguese.

2.9.1 Recent work on PT LMs

When talking about Portuguese language models, Souza, Nogueira, and Alencar Lotufo [51] released one of the first PT-BR transformer-based models named **BERTimbau**. As the name indicates, it is based on BERT, following the same pre-training objective, with slight changes in the training parameters: the total number of steps remains the same as the original BERT, as well as the same warmup logic for the first 10k steps followed by a linear decay. A batch size of 128 with a max sequence length of 512 is used throughout the entire duration of the training. It used a vocabulary of 20k tokens. A baseline and a larger version of it were pre-trained, having the initial weights for the baseline version initialized from a checkpoint of the multilingual version of BERT[4].

Posteriorly, the authors fine-tuned and evaluated their baseline on a Named-Entity Recognition task (NER[5]) against its own larger version and the "vanilla" multilingual BERT. Both their versions outperformed **mBERT**, with the larger version coming on top performance-wise, but experiencing performance degradation in a feature-based approach

[3] https://github.com/google-research/bert/blob/master/multilingual.md
[4] Released in BERT's official repository: https://github.com/google-research/bert
[5] When the model looks at a word and tries to guess if it is a noun, a verb, or an organization, etc.

(where embeddings are extracted from the model's layers and fed to a Bidirectional LSTM network).

BioBertPT (Schneider et al. [52]) follows the footsteps of BERTimbau, but only by fine-tuning mBERT for a clinical domain NER task in Portuguese. It was compared to BERTimbau, outperforming it in that specific "closed domain" NER task - since BERTimbau had not been trained for that specific clinical domain. The authors concluded that enriching a model with some domain literature can improve its performance on tasks pertaining to that domain.

Another more recent model was trained (Miquelina, Quaresma, and Nogueira [53]) using the original BERT(*base*) as its foundation. This model used a vocabulary of 30k tokens and also followed the same pre-training procedures as BERTimbau and the original BERT, maintaining the same objective, only altering the training parameters: 2e-5 learning rate and a weight decay of 0.01. No other training parameters are disclosed so we're assuming that they used the same values as the ones disclosed in the original BERT paper (Devlin et al. [4]) - and no evaluation was performed or published according to our knowledge.

2.9.1.1 First European Portuguese LM

More recently, Rodrigues et al. [12] **developed the first exclusively European Portuguese encoder-based model**, named **Albertina-PTPT**. It is based on the 900M parameter DeBERTa (He et al. [54]), since according to the authors it has shown resilience and improved efficiency on multiple strong encoder models, sometimes even surpassing human performance on the SuperGLUE benchmark (Wang et al. [34]). A counterpart PT-BR variant of Albertina was also produced, using a different set of data and slightly different training parameters.

Albertina-PTPT was trained on a corpus of 8 million documents. These 8 million documents are the result of joining multiple European Portuguese datasets - a PT-PT filtered version of OSCAR's PT-subset (Abadji et al. [55]), a collection of transcripts from the European Parliament (Europarl, Koehn [56]), the PT-PT subset of the multilingual corpus of all the documents published on the European Parliament website (DCEP, Hajlaoui et al. [57]) and a new dataset made of publicly available documents with transcriptions of debates from the Portuguese Parliament. Albertina's PT-BR variant used the br-Wac (Wagner Filho et al. [58]) corpus as the pre-training data, estimating 3.5 million documents.

Looking at the training details, its PT-BR counter-part was trained with a batch size of 896 samples - for 56 samples per GPU without using gradient accumulation - with a starting learning rate of *1e-5* with linear decay, for approximately 200k steps ($\tilde{5}$0 epochs). The PT-PT variant followed the same parameters, except it used a smaller batch size of 832 and lasted 245k training steps. These 245k steps translate to roughly 25 epochs, since they had more PT-PT data than PT-BR. In total, **the PT-PT variant saw an estimated 200M samples during pre-training**. Both variants used the original DeBERTa tokenizer with a sequence length of 128.

Albertina-PTBR was trained for roughly one day on an *a2-megagpu-16gb* Google Cloud A2 VM with 16 GPUs, 96 vCPUs, and 1.360 GB of RAM - for a total of 1280 GB of GPU Ram available. The PTPT counterpart used a smaller VM with 8 GPUs and less RAM.

Both Albertina's variants and BERTimbau were compared in several tasks: both ASSIN-2 (RTE and STS) tasks, and four GLUE tasks (RTE, WNLI, STS-B, MRPC) - GLUE-PTPT and PLUE for, respectively, PT-PT and PT-BR language branches. We describe these benchmarks in the following section 2.9.3. ASSIN-2 and PLUE are focused on PT-BR and GLUE-PTPT is focused on PTPT.

In the ASSIN-2 tasks, Albertina-PTBR came on top with a slightly increased performance when compared to BERTimbau. For the four PLUE tasks, Albertina-PTBR had better performance in three of those tasks when compared to BERTimbau and Albertina-PTPT. However, the PT-PT variant clearly takes the crown in the GLUE-PTPT evaluation (compared only to the PT-BR variant). **This cross-evaluation demonstrated that a model trained specifically for PT-PT is justified when working with European Portuguese - it further proves that a model trained for the Brazilian branch is suboptimal when working with the European Portuguese variant.**

	RTE	WNLI	MRPC	STS-B
Albertina PT-PT	**0.8339**	**0.4225**	**0.9171**	0.8801
Albertina PT-BR	0.7942	0.4085	0.9048	**0.8847**

Table 2.4: Albertina-PTPT and Albertina-PTBR performance on the four translated tasks of GLUE-PTPT. RTE and WNLI scores use accuracy, MRPC uses F1 and STS-B uses Pearson. Values reported from the original paper, adapted from Rodrigues et al. [12].

For this thesis, we also want to contemplate another language model, published by the same research group as Albertina-PTPT's. This model is called **Gervasio-PTPT**, and to the extent of our knowledge, no current academic work has been submitted or published, with the only source of information being the HuggingFace's page of this model[6]. Gervasio-PTPT is a **1B decoder-based LLM**, trained on the same data as Albertina-PTPT's. According to the available information, it was trained with a batch size of 1024 (sequence length of 2048) using a learning rate of *3e-4* with a linear decay, for one epoch over 1.7M samples during 1 day, in the same hardware as Albertina-PTPT. It's based on Pythia's 1B checkpoint - produced as a suite of multiple-sized models of the same architecture to study how a model performance evolves along its size (Biderman et al. [59]). Pythia's architecture is based on GPT3's with few differences.

How vocabulary size affects performance is not evaluated by these previous works, thus being left for future research.

Albertina and BERTimbau take a starter step in training a Portuguese language model, using BERT-like architectures as its foundation, and using datasets that were obtained from crawling the Portuguese parts of the web. Taking their similarities in training and

[6]https://huggingface.co/PORTULAN/gervasio-ptpt-base

especially the results obtained by BERTimbau and Albertina, their methodology can serve as a good starting point example of how to train a generally purposed Portuguese language model. However, the existing work only contemplates and evaluates encoder-based language models, so we expect to encounter different challenges and performance differences.

Our target methodology would begin by obtaining the dataset, and either continue the pre-training of an existing language model (or pre-train it from scratch) with the respective training objective while adjusting the training parameters to achieve the best performance, and finally fine-tuning on a set of annotated data to be used for evaluation and comparison with other models.

2.9.2 Gathering unlabeled text data

A crucial challenge in training a European Portuguese language model is where and how to get data. While unlabeled text data may exist in large quantities on the web, it must first be crawled and cleaned.

We can look at the datasets used in BERTimbau (Souza, Nogueira, and Alencar Lotufo [51]), in Miquelina, Quaresma, and Nogueira [53] and in Albertina (Rodrigues et al. [12]) to provide us with an insightful view on the process of building unlabeled datasets for pre-training.

BERTimbau is trained on a Brazilian Portuguese (PT-BR) dataset with 2.68 billion tokens and an estimated 3.53 million documents called brWac (Wagner Filho et al. [58]). It is a multi-modal dataset with syntactic annotations that can be used to evaluate models on NER tasks. Covering a modest number of domains, its documents were sourced from roughly 120k websites, making sure that one or more websites did not correspond to an elevated percentage of the dataset to maintain domain diversity.

The dataset used by Miquelina, Quaresma, and Nogueira [53]'s model (unnamed PT BERT model), while currently not publicly available, was obtained by crawling and scrapping pages through ArquivoPT (Gomes et al. [16]). The dataset build process was composed of several steps. The process starts by considering a set of known Portuguese journals or periodicals and obtaining links to their articles through ArquivoPT's functionalities that offer insightful metadata information. Then, these links are crawled and text information is retrieved. Afterward, this text is cleaned - removal of duplicates and confirmation that it is in Portuguese - and used for pre-training and statistics gathering. No annotations on this dataset were performed, and Miquelina, Quaresma, and Nogueira [53]'s model's performance was not evaluated in any way since it was not fine-tuned, being left for future research.

The objectives of this dataset somewhat align with our target corpus, where we strive to achieve a relevant corpus size and European Portuguese specialization and exclusivity. However, it is still small in comparison to the English corpora used by state-of-the-art LMs and it is focused mostly on periodicals (news & journals), so more diversity is left to

desire. Due to this, we hope to increase domain diversity for the corpus we have in mind.

The same or similar difficulties are to be expected when building the dataset for this thesis, especially in the text extraction and pre-processing (for example, removing CJK[7] characters) and making sure that the text is (almost) exclusively in European Portuguese.

To further enrich a dataset, one can also follow the example of Rodrigues et al. [12] and resort to other collections of documents, like DCEP (Hajlaoui et al. [57]) or Europarl (Koehn [56]). Both of these datasets have a European Portuguese set and are rich in documents like transcripts, meeting minutes, etc., from the European Parliament. These are made of pure text which we can use to train our model.

2.9.3 The struggle with Portuguese benchmarks

Recall that we need an annotated dataset (benchmark) to fine-tune a model for a specific task in order to evaluate it. Annotating data is costly, and even though these types of datasets are much smaller than a pre-training dataset, they may still have thousands of samples. With this amount, performing the annotations consumes too much time, and if one were to hire professional annotators, a relevant budget would be required. These difficulties are the main fuel that drives the lack of Portuguese language models - if these annotated datasets do not exist, it becomes hard to fine-tune a model and evaluate it.

SQUADPT v1.1[8] and PLUE (GOMES [60]) are two examples of Portuguese benchmarks. Refer to AnnexII and AnnexI for examples and samples from these benchmarks. These come with the downside of being machine-translated and not being exclusive to the European Portuguese branch of the language - meaning it may contain both European Portuguese and Brazilian Portuguese (PT-BR). WikiLingua (Ladhak et al. [61]) is another cross-lingual benchmark, that contains pairs of languages (ENG-PT) of WikiHow articles and their summarization.

Fortunately, recently 4 tasks from the original GLUE benchmark (RTE, MRPC, WNLI, STS-B) were translated specifically to PT-PT. These 4 tasks are the same tasks that were translated for PLUE. **The result benchmark was denominated GLUE-PTPT** (Rodrigues et al. [12]), which we'll aim to be one of our most relevant benchmarks due to it being exclusively PT-PT.

The rest of the benchmarks present a difficulty since their Portuguese is machine-translated and not European Portuguese - hence the score obtained by evaluating a model may not represent its full potential. **The difference between European Portuguese and Brazilian Portuguese is empirically known and recognized**, and should be considered.

However, due to the lack of other PT-PT benchmarks, these PT-BR datasets can be used as tools to kickstart the training and development of our target PT-PT language model and they will be serving as our base approach for this thesis.

[7] Chinese, Japanese and Korean
[8] https://forum.ailab.unb.br/t/datasets-em-portugues/251

2.9.4 Potential of very large Portuguese LMs

Until now we've looked at Portuguese models with a "moderate" size - being based on BERT-like models. Pires et al. [13] broke this "imaginary" size barrier and pre-trained a LLamA (Touvron et al. [9]) on Portuguese texts, giving rise to Sabiá - a 65B parameter Brazilian Portuguese language model.

For context, LLamA is another large generative language model, trained on billions of tokens (with exclusively publicly available datasets), whose 13B parameter version outperforms GPT-3 175B on a collection of comprehension and reasoning natural language tasks. The main reason LLamA hasn't yet been mentioned is due to this thesis's focus on models that we can actually consider using as a starting point, especially when looking at model sizes (resources).

Sabiá used 3% or less of data (in quantity) than LLamA, getting its training data from the Portuguese subset of the ClueWeb 2022 dataset (Overwijk, Xiong, and Callan [62]). It significantly outperformed many large, state-of-the-art English-centric and multilingual models on a collection-suite of 14 Brazilian Portuguese benchmarks called *Poeta*, both originally conceived and translated - mostly based on classification tasks (sentiment classification, entailment, similarity, etc.). The positive results obtained from its evaluation allowed the authors to conclude that specializing a model for a specific language (even if using an already extensively trained baseline) can lead to significant improvements.

3

Gathering & Building a PT-PT Corpus

This chapter's objective is to analyze and understand the first stepping stone required for this thesis - the data. We'll be going over its importance, diversity, size, processing, and other details that are wholly necessary to ensure model quality.

3.1 Sourcing the Data

This thesis's first step was to prepare a diverse and large enough collection of text data. We expected this to be one of the major contributions to the PT-PT NLP community. However, due to licensing issues, our contributions may only be used for research purposes, and the training data can't be publicly be publicly released - but the data pre-processing code will. When compared with the English language, the availability of European Portuguese text data in scale is very small - one can easily get terabytes of English data due to existing public datasets, but this is not the case for the PT-PT language.

In order to ensure quantity and diversity, we chose several sources of data. We will go over their domains, collection, and processing strategies in detail in the posterior sections of this chapter.

The following are the data sources we targeted:

1. **ArquivoPT**[1] (Gomes et al. [16]) - The Portuguese Web Archive
2. **Portuguese Wikipedia** ([2]) - The entire Portuguese Wikipedia
3. **OSCAR-2201 Dataset** (Abadji et al. [55]) - A collection of web pages
4. **European Parliament Transcripts** ([3]) - Transcripts from meetings from the European Parliament
5. **OpenSubtitles PT-PT Dump** ([4]): PT-PT Movie subtitles from OpenSubtitles up to 2016 (Lison and Tiedemann [63])

[1] https://arquivo.pt/
[2] https://dumps.wikimedia.org/
[3] https://www.statmt.org/europarl/
[4] https://opus.nlpl.eu/OpenSubtitles.php

6. **ClueWeb22 Dataset** (Overwijk, Xiong, and Callan [62]) - Another collection of web pages.

3.2 Ensuring Corpora Domain Diversity

This thesis **aimed at gathering rich and diverse datasets, to create a solid, large, and wide-spanning European Portuguese corpus**. If we look at the data used by some of the most relevant language models we've seen in this work (like GPT-3 - Brown et al. [15]), we notice that they don't use a single source of data. That is, their pre-training data is usually composed of multiple datasets. Each of these datasets is usually focused on a specific set of domains - be it Wikipedia data, books or novels, or conversations (forums) for example.

The main consensus is that a rich collection of domains allows the model to better understand the language (and its intrinsic knowledge), thus improving the quality of the generated text. This is further validated by more recent large LMs like LLAMA (Touvron et al. [9]), which have a big focus on data size and diversity, whose 7B model checkpoints have beaten GPT-3 175B checkpoint on several generation tasks (QA, code generation, among others).

To contribute to our goal, we chose the following sources and/or datasets:

- **Arquivo.pt Dataset** is a collection of data scrapped from mostly periodicals and news websites archived by Arquivo.pt over a span of years. It contains text written and reviewed by professional journalists.

- **OSCAR** and **ClueWeb** are *web crawls*. They essentially contain a bit of everything that can be found on the internet. They both give us text from blogs, forums, company websites, shops, etc. These represent the bulk of our data.

- The **PT-Wikipedia** gives our model reviewed, well-written encyclopedic knowledge in neutral and reviewed Portuguese text. Wikipedia presents us with concept definitions, factual texts, and explanations of a large variety of concrete and abstract things.

- The **European Parliament Transcripts (EuroParl)** are composed of transcripts from diverse sessions that occurred in the European Parliament, from colloquial conversations to argumentations on diverse topics, as said by professional deputies.

- The **OpenSubtitles**[5] **PT-PT dump** is pretty self-explanatory: comprised of subtitles from multiple movies, giving us conversations, narrations, etc.

[5] http://www.opensubtitles.org/

3.3 Collecting the Multiple Datasets

All the datasets were publicly available, each on their own chosen platform. OSCAR was available through HuggingFace, while PT-Wiki, and EuroParl were obtained from their respective websites for releasing data dumps. ClueWeb was acquired via the responsibles' website (hard drive disk), under a research-only license, which was mentioned at the beginning of this chapter. OpenSubtitles was obtained through Lison and Tiedemann [63].

The only data not currently available on the web as an actual dataset is the data from Arquivo PT and the official OpenSubtitles dump. The collection of data we used was obtained by crawling the archived websites available in Arquivo PT. However, we discriminated against which data to crawl and collect - **meaning we focused only on periodicals and news websites**. This decision was made for two main reasons - time and data quality. For the first reason, crawling, collecting, and processing the entirety of the Arquivo PT data would take colossal amounts of time, being left for future work since this platform has enormous untapped potential for creating new European Portuguese datasets.

For the second reason, text written by journalists and news outlets is generally known to be of a certain level of quality: it is simple and has been reviewed (very low probability of having grammatical errors). So we compiled a list of well-known Portuguese news and periodical domains and used them to gather our data. It is important to note once again that this crawling process was made in collaboration with two other Master's theses since the data was used to develop ArquiWiz's prototype for the annual ArquivoPT contest.

3.4 Processing & Cleaning The Data

Once the datasets were gathered, they still had to be filtered, cleaned, and processed, since datasets like OSCAR and ClueWeb have subsets for multiple languages, and we only wanted to filter the Portuguese subsets for example. The pre-processing applied to the

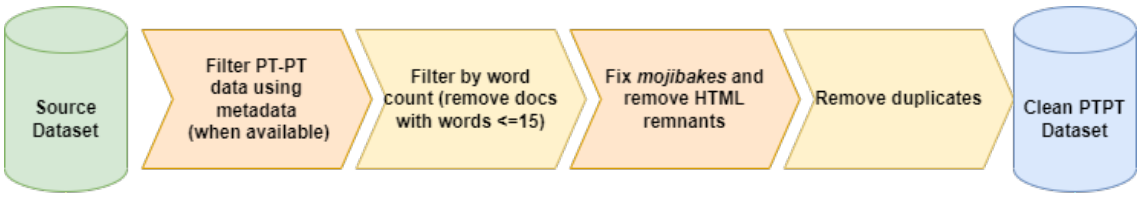

Figure 3.1: Representation of the general pre-processing applied to the datasets

datasets was split into the following steps: filtering only PT-PT documents using available metadata (documents in OSCAR and ClueWeb whose URL contains ".pt" in its domain), removing documents with a low word count, fixing *mojibakes* (encoding errors), removing remnant HTML tags and removing duplicates.

For example, the first step could only be applied to OSCAR and ClueWeb since they're the only ones with available metadata (since they're web crawls), and it helps us try to ensure that these two datasets only contain PT-PT texts - or at least data that originates from a European Portuguese source. The remaining sourced datasets are already known to be in Portuguese. For ClueWeb, the final duplicate removal step was skipped since it was already performed by the original developers of the dataset. An extra processing step of removing documents with bad words had to be applied to the OpenSubtitles data due to alarming quantities of insults, swear words, and racist terms, among others being present. A list of Portuguese bad words was handwritten, and used to filter out samples. Refer to figure 3.1 for a visualization of the pre-processing pipeline.

3.5 Choosing a Sampling Strategy

Mixing large and diverse corpora from multiple sources, each one with different sizes, lit up an important question: how should the model view the data - which dataset should it view first? Should it focus more on encyclopedic knowledge? Should the model view data with an equal probability?

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

Figure 3.2: Datasets used to pre-train LLAMA and their sampling weights. Taken from Touvron et al. [9]

Contrary to the data used by LLAMA, we have no code nor scientific articles or papers, but it doesn't mean they're impossible to get for future work - just like further crawling Arquivo.pt for more data from new sources besides news and periodicals.

Due to this and to the fact that performing multiple trainings, fine tunings, and evaluations for different data setups would be expensive in both time and resources, we pondered using a sampling strategy similar to the one used by LLAMA (Touvron et al. [9]). This strategy consists of constructing batches using sampling probabilities from multiple datasets, where these probabilities (or weights) can be arbitrary. We can look at Figure 3.2 to check how the weights were distributed according to the dataset and their size (for LLAMA). This strategy is used in our training through a custom data loader, which we will see in detail in the next chapter when we look at the pre-training scheme and other internal details.

3.6 Post-processing Statistics

After processing the source datasets, we obtained a clean version for each one of our chosen sources. These clean, PT-PT versions are the ones used to pre-train our model - with the small exception of the PTWiki since pages directly related to Brazil (customs, people, etc.) have a very slightly different type of writing. However, since this written text closely follows the actual grammatical and semantic rules, the difference from PT-PT becomes almost negligible. By looking at Table 3.1, we conclude that the ClueWeb dataset constitutes the bulk of the data, with an astounding total of 29M documents.

Dataset	Domain	Total Documents	Total Tokens	Avg. Words p/ Doc	Size (GB)
ClueWeb PTPT	Web Crawl	29M	31.6B	610	111GB
OSCAR PTPT	Web Crawl	1.5M	1.8B	784	7.6GB
ArquivoPT	News and periodicals	1.5M	846M	434	3.9G
OpenSubtitles PTPT	Subtitles from movies	1.2M	1B	496	3.7G
PTWIKI	Encyclopedia	820k	233M	204	1.1G
EuroParl PTPT	Transcripts from the European Parliament	1.3M	54M	31	0.3G

Table 3.1: Collected datasets and post-processing statistics.

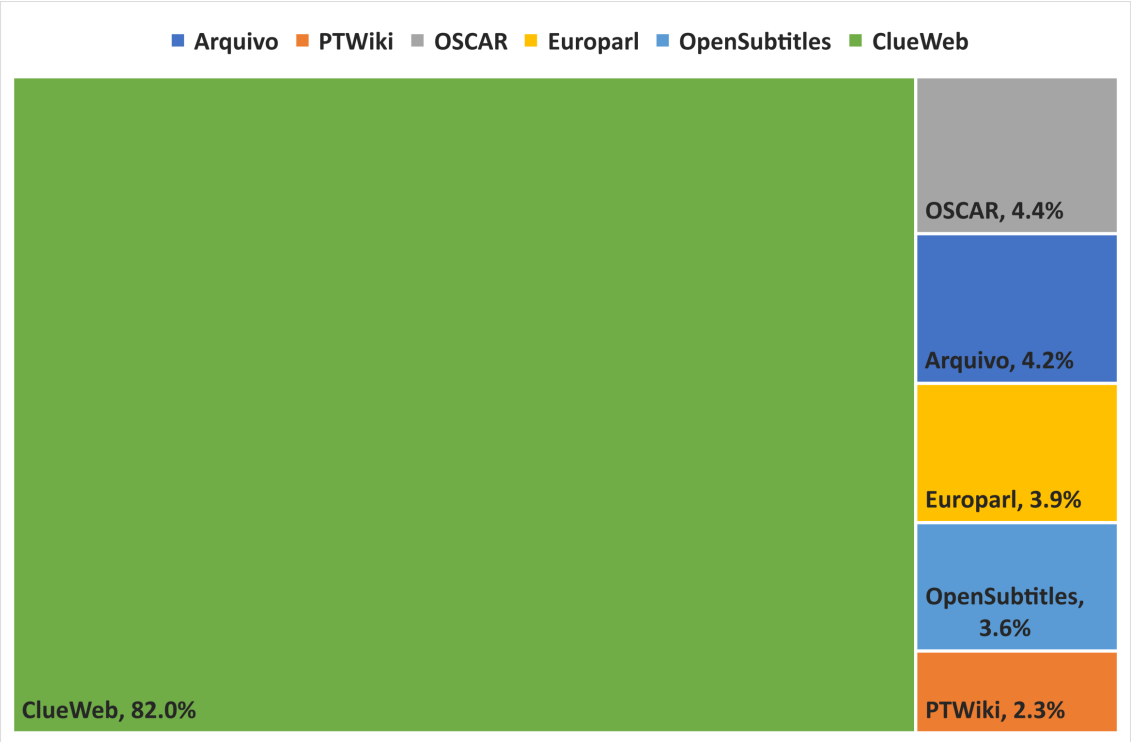

Figure 3.3: A tree map to visualize the final composition of our text corpora (to scale). The values presented are each dataset's weight/percentages related to the total number of samples.

We also collected the total number of words, and the average words per document, to help us make an estimate of how many tokens the model could see. Seeing as one word

can be broken into one or more tokens when being tokenized and processed by the model, and having a total of **19.7B words in our dataset**, which **translates to our account of a total of 35.6 billion tokens**. To count the tokens, we used our own tokenizer, prepared on our text corpora, details of which we'll be seeing in Section 4.2.

4

Pre-Training a Large Decoder-Based Portuguese Language Model

In this chapter, we will analyze and detail different training and model configuration schemes. We'll be looking at the parameters, which optimizer and scheduler were chosen, and some internals like the data loading mechanics, among other details - as the respective pre-training results.

In Chapter 5 (next chapter), we will begin talking about our first model evaluations.

4.1 Choosing a Decoder-based Model

The first crucial decision that needed to be taken, was which base model/architecture to choose. This choice ultimately impacts the training scheme (parameters, seen data, etc.), resources, and the ability to perform certain downstream tasks - let's go over some details of how and which model we chose.

At the start of this thesis, specifically during its preparation, some preliminary work was done using a BERT model. This preparatory work aimed to net a first assessment of the code base, metric reporting tools, and learning and analyzing how a model could behave. However, BERT has long been surpassed as state-of-the-art, with its performance lacking in generative capabilities due to the usage of encoders. Most recently, the success and improved performance of generative models like ChatGPT (GPT-3.5 & GPT-4 for e.g) and LLAMA (Touvron et al. [9]) have shown that their generative capabilities are immensely versatile by demonstrating that they could achieve humanlike performance in a diverse domain of tasks - using their ability to predict sequences of words left-to-right. Such tasks include but are not limited to question-answering, following user-prompted instructions, or performing simple reasoning for example.

Since the goal of this thesis is to train a large generative language model, the chosen base model was the **GPTNeo**, developed by Black et al. [64]. GPTNeo is an open-source model trained on the Pile dataset (Gao et al. [65]), in an attempt to produce an available *GPT3-like* model (since GPT-3 is not publicly available), as it competes with it in linguistic, physical and scientific reasoning tasks - part of the reason why we chose this model over

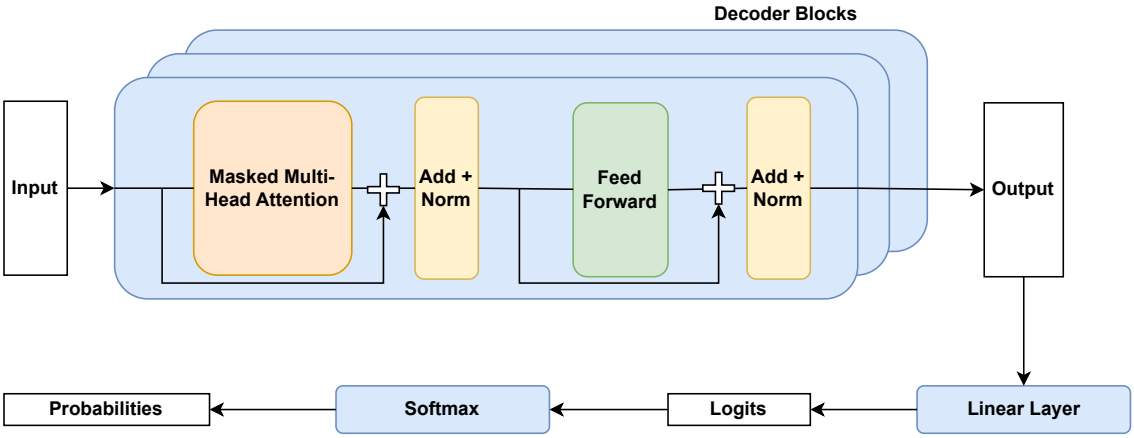

Figure 4.1: Representation of the inner architecture of an auto-regressive, decoder-based model. Composed of several multiple decoders, the input is processed in parallel. Masked self-attention is calculated inside the decoders, followed by normalization and feed-forward layers. Posteriorly, the output (which is essentially a vector of floats) goes through a linear layer and a softmax to obtain the probabilities for each word in the vocabulary.

the T5 for example. At the time of writing, another ideal choice would be a 7B LLAMA model, but not only did a relevant part of this work had begun before it was released, but our available resources were not enough to pre-train it due to its very large size. Since we aimed at using an open-sourced model, a Pythia model (Biderman et al. [59]) could also be a reliable choice due to its open-sourced nature of both model, data, and support for training reproducibility, but once again Pythia was released after having already chosen GPTNeo. **Ideally, time and resource permitting, at least two base models would have been chosen and pre-trained using the same protocols and methodology so they could be compared**. Figure 4.1 provides a fairly simple representation of the architecture of our chosen autoregressive model.

Having used HuggingFace's implementation of GPTNeo, it additionally makes use of **Local Attention** (Beltagy, Peters, and Cohan [10]) and **Linear Attention** (Zhuoran et al. [66]).

GPTNeo's **Local Attention** replaces the standard self-attention mechanism in the transformer and combines a dilating sliding window strategy with pre-selected global attention on some input locations. The "regular" self-attention **scales quadratically** with the sequence length of the input, so the implemented sliding window strategy tries to address this issue: for a window size w, each token attends to $\frac{1}{2} * w$ tokens in both directions and for a sequence length n, this **scales linearly** with n since the complexity is $O(w * n)$ (when compared to the "regular" mechanism).

Building on the previous mechanism of optimizing the self-attention, the **linear attention** (Zhuoran et al. [66]) they refer to in GPTNeo is an optimization of the dot-product calculation that, in very simple terms, changes the calculation of the dot-product attention, so that the size of intermediate matrices depend on values that can be controlled (such as the dimensionality of the embeddings). This way, it **provides linear memory and**

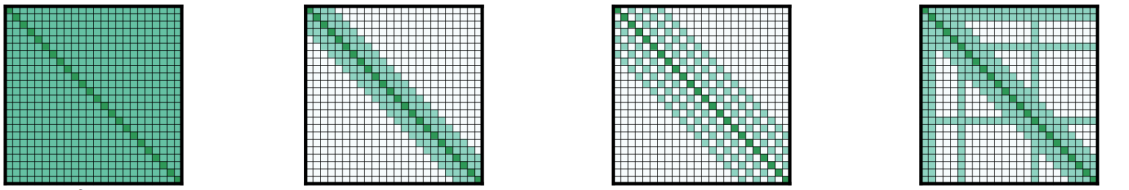

Figure 4.2: Representation of several self-attention patterns. The first one is the "regular" pattern as we've seen, and the last one is the one implemented as the local attention mechanism in GPTNeo. Adapted from Beltagy, Peters, and Cohan [10].

Table 4.1: Model size comparison between the chosen GPTNeo checkpoints and recent Portuguese language models.

Model	Parameters	Layers	Attention Heads	Embeddings Dimensionality
BERTimbau(*base*)	110M	12	12	768
BERTimbau(*large*)	335M	24	16	1024
Albertina-PTPT (*large*)	900M	24	12	1536
Gervasio-PTPT (*large*)	1B	16	8	2048
GPTNeo 1.3B	1.3B	24	16	2048
GPTNeo 2.7B	2.7B	32	20	2560

complexities, while making sure the representational capability is maintained.

Taking these optimizations into account, the 1.3B and 2.7B (as a bonus) parameter checkpoints from GPTNeo were chosen as the bases for our model - **GlórIA**. Table 4.1 presents a size comparison between our versions of GlórIA and of some Portuguese models. These specific checkpoints were selected according to the available hardware resources, training time estimation, and making sure the model size is accessible to researchers (since increasingly larger models would be more difficult to fit in GPUs - we will talk about available hardware posteriorly).

4.2 Tokenization: Input Preparation

As we've seen in Section 2.3, data needs to be prepared before being sent into the model - this is the tokenization step. Since our model is focused on the Portuguese language, our main concern when dealing with tokenization is that a tokenizer that has been prepared for English would be suboptimal when attempting to tokenize words in different languages. So we prepared our own tokenizer on part of our PT-PT text corpora. The HuggingFace's Tokenizers library was used for this effort.

Following the original details for the GPT-2 tokenizer, we prepared a BytePair-Encoding-based tokenizer with a vocabulary size of 50258, trained on a part of our corpora.

We limited the **max sequence length to 512**, truncating documents when necessary. For documents with a sequence length < 512, we padded the input to make sure every input has a sequence length of 512 since the model deals with absolute sizes - meaning every input must have the same size. We chose this value taking into account how many

samples we could fit into memory, as well as the average number of words per document in each pre-training dataset.

4.3 Preparing a Training Recipe

In the following section, we're going to start discussing the different training scheme modules and the choices behind certain aspects.

4.3.1 Methodology

DATA SETS	LIST	FILTERED PT-PT
A	PTWiki, ClueWeb, Arquivo	No
B	PTWiki, ClueWeb, Arquivo, Oscar, Europarl	Yes
C	PTWiki, ClueWeb, Arquivo, Oscar, Europarl, OpenSub	Yes

Table 4.2: These are the different dataset compositions that were tested on and better processed throughout this thesis.

To investigate how the model would evolve during pre-training, along with the development of this thesis, multiple runs were performed. These runs would have their metrics (loss, perplexity, memory usage, CPU usage, GPU usage, among many others) reported to Wandb[1] in order to easily visualize the model's behavior, and they would last a relatively short time when compared to the actual pre-training duration.

At the start of testing, not all datasets we used were available and processed, so the first detail to point out, is that not only the hyperparameters (learning rate, batch size, weight decay, etc.) were changed, but as some of the datasets would become available, they would be added to the training corpora (Table 4.2). We noticed that adding new datasets to the corpora would lead to small changes to the metric's curves - with this in mind, we tried to tweak the training hyperparameters to make sure the training remained stable as our corpora composition would change during preliminary testing. We go over how the datasets are handled, specifically dataloading and shuffling in Section 4.3.9.

As was pointed out, the main variation in these runs would be the hyperparameters. We tested several learning rates, batch sizes, sequence lengths, warmup steps, weight decay, etc. The objective of these runs was to stabilize the hyperparameters and metrics that we chose to monitor the model's quality.

We can look at Table 4.3 for an example of the type of pre-training runs performed - notice that the number of steps differs greatly from each one - this is due to the variations in the composition of the datasets, and their batch sizes, as well as only wanting to compare runs for their first 200k steps for example, since their behavior was very similar and the convergence trend was the same. This diversity in experiments was also a goal to see if we could find any unexpected behavior when altering the data or parameters. To monitor the

[1] https://wandb.ai/

#Run	Tot. BS	BS p/ GPU	GA	Micro BS	Steps	Restarts	Data-Sets	Weights
1	128	32	32	1	7.3M	2	A	Equal Distribution
2	128	32	32	1	3.6M	2	A	Equal Distribution
3	128	32	32	1	670K	2	B	[0.1, 0.1, 0.15, 0.05, 0.6]
4	128	32	8	4	255K	2	B	[0.2, 0.1, 0.15, 0.05, 0.4]
5	128	32	32	1	1.8M	2	A	Equal Distribution
6	128	32	4	8	176K	2	B	[0.2, 0.1, 0.15, 0.05, 0.4]
7	256	64	8	8	200k	2	B	[0.2, 0.1, 0.15, 0.05, 0.4]
8	256	64	8	8	200k	4	B	[0.2, 0.1, 0.15, 0.05, 0.4]
9	512	128	16	8	1M	4	C	[0.1, 0.08, 0.63, 0.06, 0.08, 0.06]

Table 4.3: Logs of initial pre-training runs made for the 1.3B version. Excluding test runs (codebase testing), and "sanity check" runs (duplicated runs to confirm results are reproducible). The data sets correspond to the different compositions of our pre-training data throughout the development, as seen in Table 4.2. The weights correspond respectively to PTWiki, Arquivo, ClueWeb, Europarl, Oscar, and OpenSubtitles. **BS**=Batch Size, **GA**=Gradient Accumulation

.

model's quality during pre-training, we chose **the perplexity as primary metric**, and the loss as secondary (if one grows, so does the other) - since using perplexity makes more sense for generative models due to the causal nature of their training objective. In sum, it allows the researchers to get a better idea of how much trouble the model is having when asked to generate text.

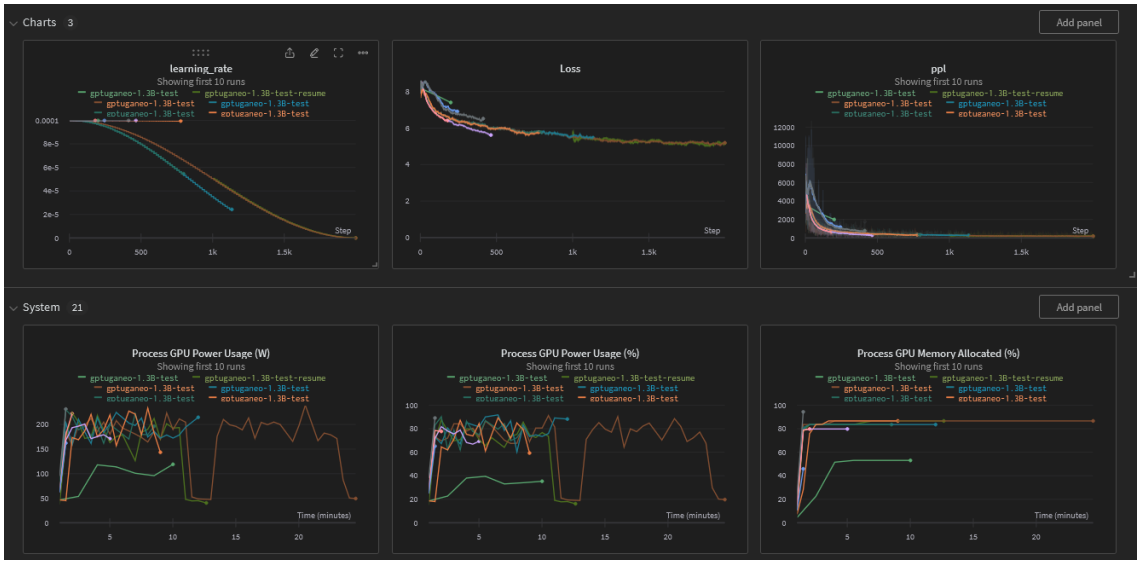

Figure 4.3: Example of charts reported to Wandb. In specific, these are pre-training experiments performed to test the codebase, meant to demonstrate if the reporting is working. Multiple runs could be viewed if necessary and compared.

4.3.2 Defining the Pre-Training Objective

Using the Transformers' decoder blocks at its core, our model follows a generative nature. Consequently, the model employs a causal language modeling objective. It aims to predict the next word in a sentence, given a sequence of preceding words. It's a causal objective since the model can only use contextual information from the "past" - words that came before the current word.

So, for a sequence of words:

$$W = (w1, wi, ...wT)$$

where *wi* is the *i*-th word and T is the total amount of words in the sequence, the goal is to predict *wi* given *(w1,w2,...,wi-1)*. We can express this as:

$$P(w_1, w_2, \ldots, w_T) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2) \cdot \ldots \cdot P(w_T|w_{T-1}) \tag{4.1}$$

To compute this, consider *hi* as the hidden state (logits) of the model at position i - the probability of the next word *wi* given the context *hi-1* is computed using a softmax function as follows:

$$P(w_i|w0, ..., w_{i-1}) = \text{softmax}(h_i) \tag{4.2}$$

4.3.3 Choosing Hyperparameters

Next, we're going to be looking at the hyperparameters used to pre-train GlórIA and some of the reasoning behind the choices for certain values.

Through the initial pre-training runs performed (once again in Table 4.3) we concluded that a **learning rate of 1e-4** seemed like the ideal starting value, together with **10k warm-up steps**. Parallel to these hyperparameters, the model was trained in a **BF-16 mixed precision format**.

We also extended our training scheme and methodology to GlórIA's 2.7B version, with a few small differences in batch size and total training steps, as we will see in the following sections.

Check Table 4.4 for a compilation of the hyperparameters used to pre-train GlórIA 1.3B and 2.7B.

Size	Batch Size	GA	Seq.Len	Weight Decay	MixedPrecision	Warmup Steps	Total Steps
1.3B	512	16	512	0.01	BF16	10k	3M
2.7B	504	16	512	0.01	BF16	10k	1M

Table 4.4: Compilation of the used hyperparameters to pre-training GlórIA.

4.3.4 Training Steps

Due to the large scale of training data and the available resources, pre-training for GlórIA 1.3B was split into 3 parts. We first trained for 1M steps and then resumed training to reach 2M steps. Posteriorly, we resumed training once more to **reach our final 3M steps**. Periodical checkpoint evaluations would be performed between stopping and resuming. We maintained the same seed and shuffled the data when resuming and at the end of every epoch. For the 2.7B version, we had to limit pre-training to 1M steps due to time and resource logistics.

4.3.4.1 Mixed-Precision Mode

The BF16 mixed-precision format is a numerical representation ("b" for binary and "f" for "floating-point") designed to achieve a balance between memory efficiency and precision. It reduces the range of representable values when compared to *float32*, retaining the same 16-bit storage size while allocating fewer bits to the fractional part of a number - while arithmetic operations are still done with 32 bits. This reduces precision, but the impact is small to the point where it's acceptable, as the benefits of reduced memory usage outweigh this consequence. BF-16 is particularly useful for training large language models like GPTNeo due to them having billions of parameters (requiring enormous quantities of memory), and resources are limited.

By using this mixed-precision format, we are able to reduce the memory footprint of the model significantly, enabling more efficient usage of available hardware.

4.3.4.2 Notes on Batch-Size

Tests on batch size were also performed, and the main conclusion is that the larger the batch size, the more information the model sees, thus an improvement in metrics and quality was seen. Referring back to Table 4.3, notice that the Micro BS increases from run 6 onwards, being increased 8x fold. This is thanks to the integration of DeepSpeed which is related to our distributed training setup described in Section 4.3.8.

The micro-batch size is the number of samples that could fit in the model at the same time without going over the memory limit - this value is calculated by dividing the batch size per process/GPU by the number of gradient accumulation steps (GA). Gradient accumulation (GA) is a technique that allows us to essentially split a batch into smaller batches that are fed sequentially to the model, and only update the gradients of the model after these are processed. This allows us to save memory and simulate larger batch sizes. For example, for a batch size of 128 and 2 gradient accumulation steps, we can simulate a batch size of 256 - so the gradients are only updated after every 2 steps.

After performing our starting tests, we set a **total batch size of 512**. Since we used 4 GPUs for GlórIA 1.3B, we had a **batch size of 128 per GPU**. To achieve this batch size on a single GPU, we used **a gradient accumulation value of 16**, which would give us a **micro-batch size of 8 samples**. For the 2.7B version, we used 7x GPUs - thanks to this, we could only approximate the total batch size up to 504 - being the closest value we could get due to the number of GPUs. This translates into a batch size of 72 per GPU, with 18 gradient accumulation steps (micro-batch of 4).

4.3.5 Optimizing parameters and learning rate

Optimizers and schedulers are vital components in training language models. Optimizers adjust model parameters, aiding convergence and generalization, while schedulers adjust learning rates, preventing local minimums and enhancing performance. Choosing an adequate and robust optimizer as well as its partner scheduler go a long way in unlocking the potential of a model. **Ideally, to choose the correct optimizer and/or scheduler we would perform extensive testing, but due to the resource limitations**, we had to choose an optimizer and scheduler based on empirical knowledge, known usages by other Portuguese models, and our few preliminary tests. Thus we aim to explain how our optimizer and scheduler both work.

4.3.5.1 Choosing an optimizer

The chosen optimizer was **AdamW** ([**Adam**, 67]). As seen in section 2.4.1.3, Adam (Kingma and Ba [27]) is an adaptive learning rate optimization algorithm - a cost-minimizing function. In simple terms, Adam adjusts the parameters of a model, so that it (the model) can learn more efficiently from the provided data. These updates are based on the calculated gradients of the loss function. Alongside the basics, it leverages the idea of a

first and *second* "moments" of the gradients to perform the updates. We can look at the *first* momentum as the tendency the gradients have to move in a given direction, while the *second* momentum can be interpreted as the variance of the gradients - helping Adam know when to reduce or increase the learning rate. For example, if the gradients present a higher variance, Adam will reduce the LR to make sure the taken steps are smaller. Essentially, taking into account past gradient's behavior, Adam is able to efficiently and adaptively adjust the learning rate for the parameters.

AdamW (Loshchilov and Hutter [67]) differentiates from the regular Adam optimizer in how weight decay works. In regular Adam, the weight decay is multiplied by the LR and subtracted from the parameter update, acting directly in the parameter update step. AdamW however, decouples the weight decay from the parameter update step. This change serves the goal of not interfering with the adaptive LR behavior by ensuring it only affects the parameter regularization.

These details, together with the fact that some recent Portuguese LLMs also chose to **use AdamW as an optimizer** (BERTimbau - Souza, Nogueira, and Alencar Lotufo [51]) for their pre-training, make it a robust and flexible choice for our model and experiments - hence our optimizer of choice.

4.3.5.2 Choosing a scheduler

Due to large amounts of data, our pre-training is expected to run for a long time, and have lots of steps. Choosing the right scheduler is a difficult task, since **the way the learning rate changes has a direct impact on how well the model converges**. Mixing the impact on convergence and the expected long duration of our pre-training, it was hypothesized that having the learning rate start at a "high" value for a short period of time and then slowly decay into a smaller value would be the optimal approach. This way, the model would converge slightly faster at the beginning, while stabilizing and steadily reaching an optimum state. This was somewhat confirmed in early experiments, where **a cosine annealing scheduler** gave the best results (in loss and perplexity convergence). Picking up again on the predicted long duration of the training, one main concern was the possibility that the model could be "stuck" in a local minimum.

To combat this possibility, we **considered warmup steps** (learning rate increases linearly to the initial value over X steps) **and hard restarts for our scheduler** - meaning that for every Y steps, the learning rate would go back to the starting value and start decaying again.

We can look at Figures 4.4 and 4.5 to confirm this behavior and concern on the test runs *GlórIA-1.3B-deepspeed-nd* and *GlórIA-1.3B-deepspeed-nd -v2* - **runs 6 and 7 in Table 4.3**. Also notice how the **second test run, with increased batch size, became much more stable** than the first one.

In the end, we chose a **cosine annealing scheduler**, that would perform a **hard restart every 500k steps**.

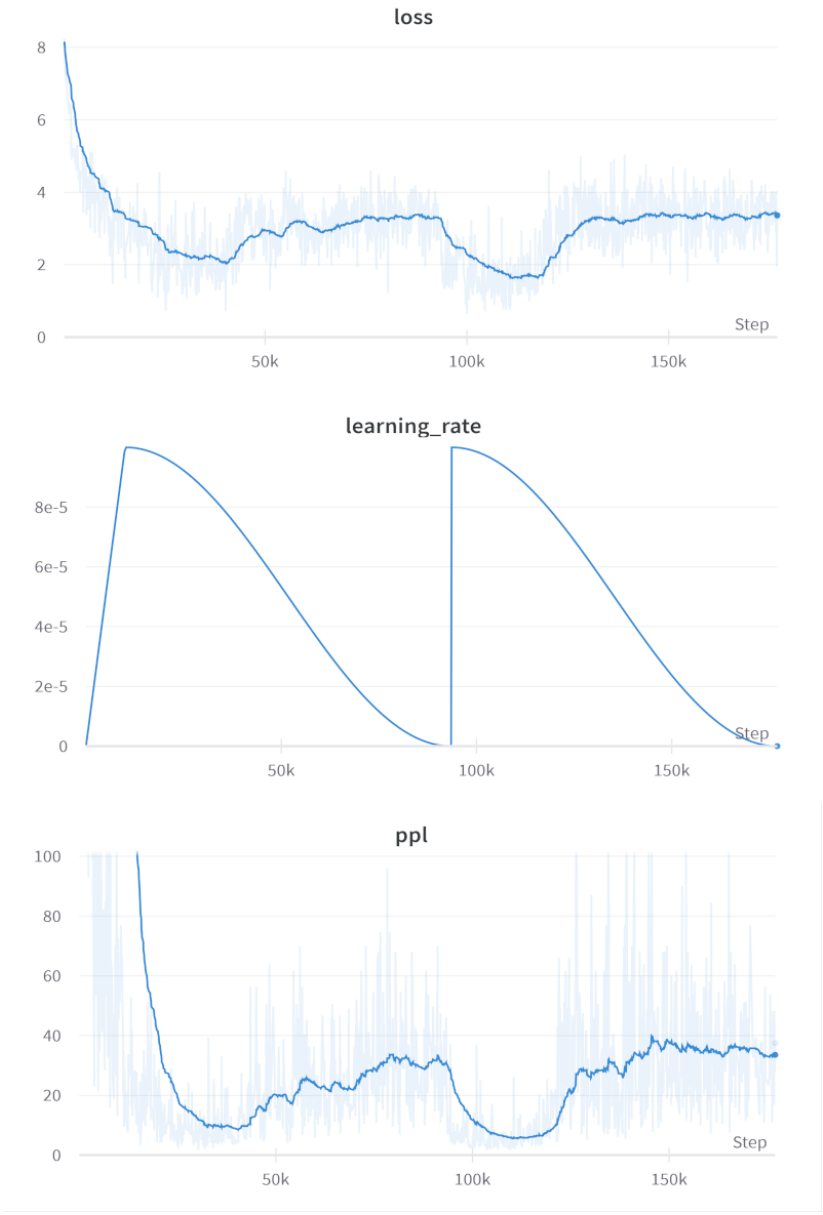

Figure 4.4: Loss, perplexity, and learning rate curves for *GlórIA-1.3B-deepspeed-nd* - one of the test runs performed. Here we can see that the scheduler hard restart at 100k steps helps the model attempt to leave a possible local minimum.

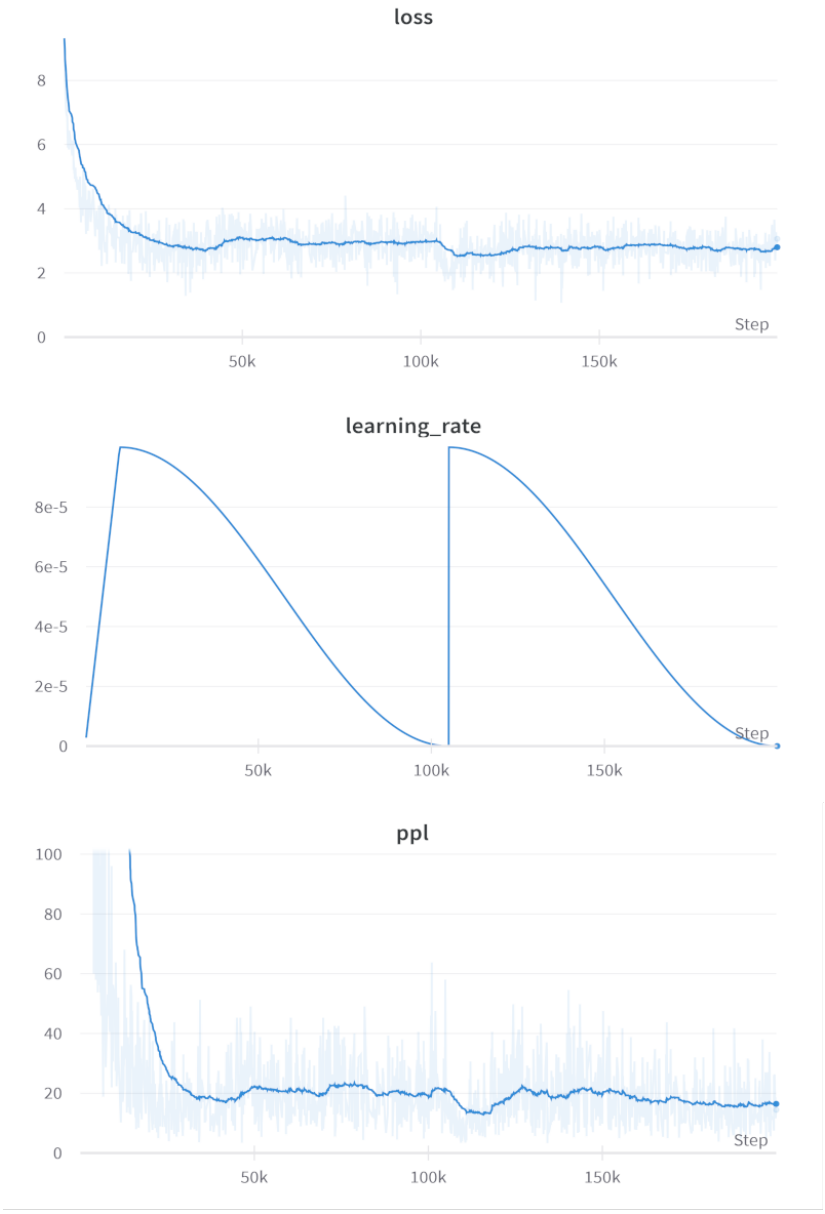

Figure 4.5: Loss, perplexity, and learning rate curves for *GlórIA-1.3B-deepspeed-nd-v2*. Now with a more stable perplexity, we can still notice a decrease when performing a hard restart at 100k steps.

4.3.6 Preparing a Codebase

To pre-train GlórIA, a custom Trainer was implemented. This Trainer has some specificities, where it aggregates functionalities to not only pre-train the model for a causal language modeling objective but also support fine-tuning for pre-selected tasks. As expected it was developed in Python using libraries such as PyTorch and some of HuggingFace's. This codebase was used and extended as necessary during the development. The following sections will expand on this - like the need to distribute training across GPUs.

4.3.7 Hardware & Duration

For this thesis, a cluster with two machines (let's call them machine Alpha and Beta) was available to perform tests and run the training. Machine Alpha has 8x Nvidia A100s[2] 40GB PCie GPUs, with around 1TB of CPU RAM available, while machine Beta has 4x Nvidia A100s 40GB SXM4 GPUs with 384GB CPU RAM available. Machine Beta is better for 4x GPU simulations due to the GPUs being linked via an SXM4 interface, making it approximately 30% faster than machine Alpha according to our tests.

Since these machines were shared amongst other researchers and students, a responsible usage policy of the resources was followed. Ideally one would use 8x A100s, but for GlórIA 1.3B we had to restrict to using machine Beta's **4x A100s** for the first 1M steps of pre-training, and other 4 GPUs in machine Alpha when resuming training up to the 3M - while **7x A100s in machine Alpha were used for its 2.7B version** posteriorly.

Using this hardware, we needed approximately 7 days of training for 1M steps, which translates to a total of 21 days of pre-training. For GlórIA 2.7B, due to the increased size and resource requirements, we were only able to train it for 1M steps, lasting around 10 days. We maintained the same seed and shuffled the data when resuming for every pre-train.

Any remaining GPUs in machine Alpha would be used for running finetuning simulations and evaluations. Between 160GB to 200GB of CPU RAM was allocated to each pre-training run.

4.3.8 Distributing Training Across Multiple GPUs

When working with large language models, making sure we can fit our model into the hardware is one of the first problems that appear - as you not only need space for the model's weights, you also need space for the gradients and optimizer states that occupy the bulk of used memory during pre-training. The solution is to distribute these components across multiple GPUs, and that is what we did. Let's now dive into how this was done.

Inside the custom trainer's internals, the **Accelerate** (Gugger et al. [68]) library was used. Accelerate abstracts boilerplate code needed to run training loops of PyTorch models in a distributed setup. Through very few code changes, Accelerate permits us to

[2]https://www.nvidia.com/en-us/data-center/a100/

run our training in a single or distributed node setting (single CPU or GPU, multi-GPUs or multi-TPUs), including the management of mixed-precision formatting. It also handles the device placement for the developer, to reduce cases where errors might appear due to the developer simply forgetting to move a batch of data to the GPU - since both the model and the input data need to be on the same device.

This library allows us to go even further through **DeepSpeed** integration, which was used to train models like BLOOM-176B (Workshop et al. [38]) and GPT-NeoX (20B checkpoint identical to GPT-Neo, but larger and more recent - Black et al. [69]).

DeepSpeed is a library that implements innovations and technologies for training, inference, and compression. For this thesis, we take advantage of the training pillar of DeepSpeed, since it implements every ZeRO stage (Rajbhandari et al. [11]). Its focus is to further optimize memory usage by eliminating memory redundancies in data and in distributed models. A high computational granularity and low communication volume are maintained to ensure constant high efficiency during training.

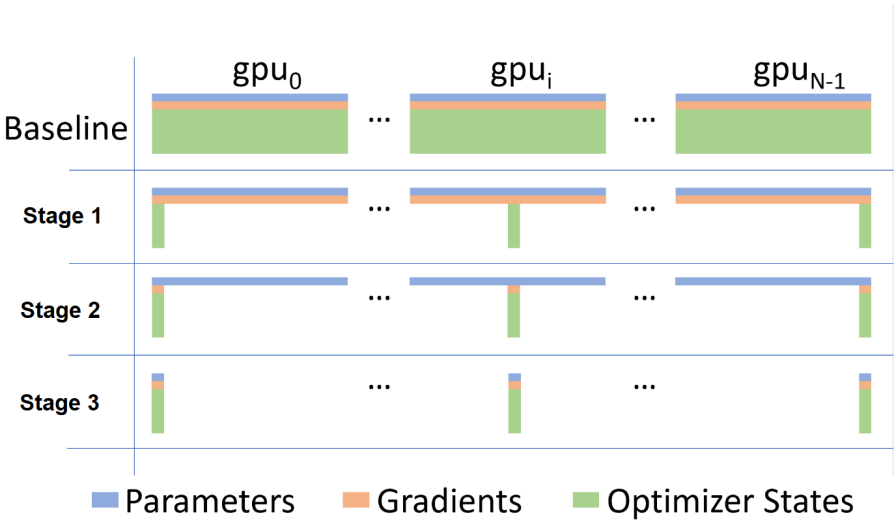

Figure 4.6: Visualization of each DeepSpeed stage. Each stage "splits" and distributes different memory-consuming elements - the baseline demonstrates the redundancy of having the same optimizer states (for example) replicated across GPUs when they can be correctly distributed. Adapted from Rajbhandari et al. [11].

Figure 4.6 demonstrates the several stages of DeepSpeed that are supported. Stage 1 only shards the optimizer states across GPUs. Stage 2 shards both optimizer states and gradients. Stage 3 shards optimizer states, gradients and model parameters across data parallelized GPUs. The higher the stage, the higher the sharding, thus more memory available inside each parallel GPU. In this thesis, we use **stage 2 of ZeRO to distribute the optimizer and gradients since they occupy the bulk of used memory**, and in our experiments using stage 3 provided little difference in our setup, as it wouldn't make a big enough difference that would allow us to further increase batch size or sequence length.

In practice, the multiple GPUs process batches in parallel, and every X gradient

accumulation steps, the gradients from the processed batches are gathered and the model weights are updated via backward propagation. Refer to Figure 4.7 for a visualization.

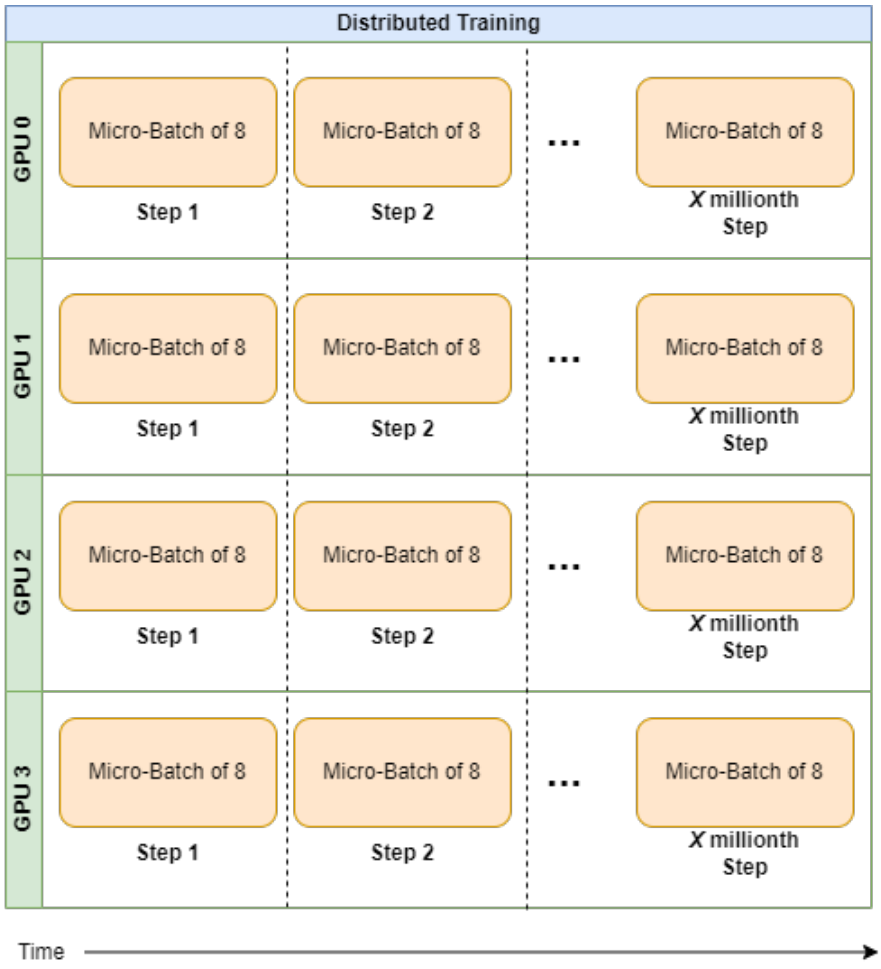

Figure 4.7: Distributed training visualization. Micro-batches are processed in parallel, and every X steps the model weights are updated by gathering the gradients from the batches that were processed, thus simulating larger batch sizes.

4.3.9 Preparing Dataloading & Sampling

Since we're training a model in a distributed setup, we must make sure that the data is correctly distributed across processes. Not only this, but it's imperative that we're able to build our batches of data according to the weights/sampling probabilities we attribute to our datasets.

To achieve this, two custom classes were implemented. A *MultiDataset* class and a *SingleDataset* class. The **SingleDataset** class is responsible for managing a single dataset - a SingleDataset is instantiated for each of our datasets that composed our corpora: 1 for OSCAR, 1 for ClueWeb, 1 for Arquivo.pt, etc. This class handles document counting and iterator instantiation. It also stores statistics, specifically how many samples were seen. According to the processes' rank, each *SingleDataset* will only load a split of the data, to

make sure that **each process only reads non-redundant data**. This also **stops different processes from seeing the same, duplicated data, as each other**.

The **MultiDataset** class leverages multiple *SingleDataset* classes. It receives an array of these and their respective weights (or sampling probabilities) as input. This class works by being bundled inside a PyTorch Dataloader, together with a WeightedSampler. What happens with this combination is the following: the PyTorch WeightedSampler uses the given weights to generate an index. This index will correspond to one of the datasets (not their documents) that we passed to our *MultiDataset*. Inside it, the indexes are associated with each dataset's iterator, allowing this class to loop over their data - using this index, the *MultiDataset* class retrieves a document using the corresponding dataset's iterator. Accelerate wraps our dataloader and makes sure that the constructed batches are thus distributed across processes.

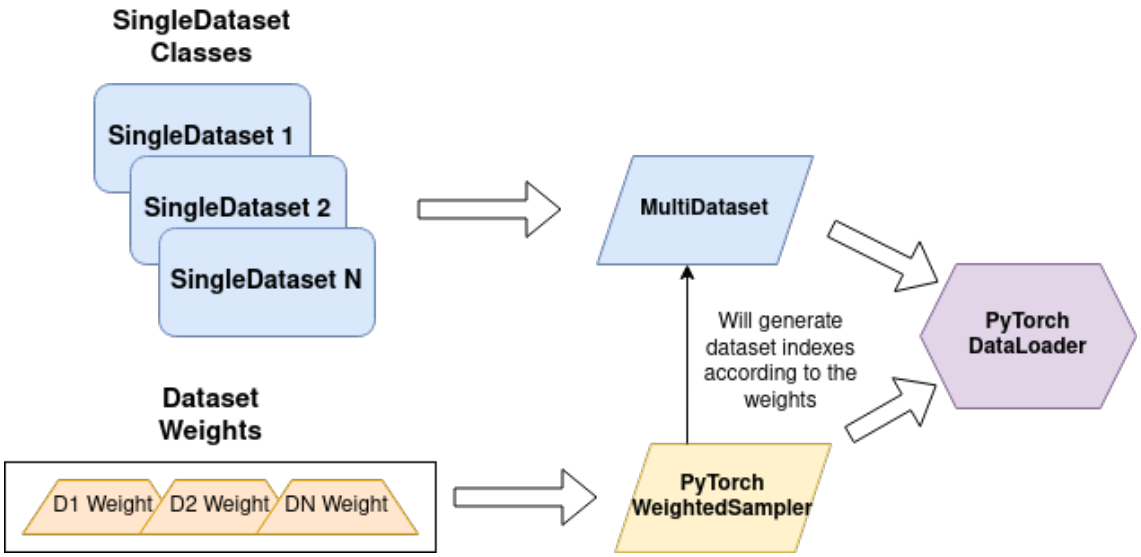

Figure 4.8: Data Loading scheme representation.

4.3.10 Miscellaneous Details

Some relevant miscellaneous details related to training and the codebase:

- **Data shuffling**: The Trainer was programmed to shuffle the pre-training data at the end of each epoch, using the number of the epoch as a shuffle seed for reproducibility. Data shuffling would also be performed when resuming training.

- **Gradient Clipping**: gradient clipping was applied during every training phase (pre-training and fine-tuning) with a maximum threshold of 3 for the gradients' normals, as a way to prevent very large updates to the model weights which could lead to instability and other difficulties when converging.

4.3.11 Pre-Training Quality Monitoring

Pre-training quality monitoring was performed by observing the model's behavior while training specifically, by observing the pre-training perplexity and its relation with the loss. Throughout our experiments and tests, the perplexity convergence aligned with the loss's, having established the first condition for stable training.

The perplexity metric was chosen for in-training quality assessment due to the causal nature of our model's pre-training objective. As explained in Section 2.7.1.1, it allows researchers to get an idea of how "stuck" and perplexed the model is when asked to generate a sequence of text - lining up with the pre-training objective.

4.4 Pre-training Results

Reaching the end of this chapter, we perform an analysis of the pre-training results, making observations on behavior and stability.

4.4.1 GlórIA 1.3B

For GlórIA 1.3B, we identified a relatively fast convergence and stabilization of both the loss and perplexity at around 300k steps. Onwards, the training remained stable and no noticeable remarks could be made, other than that the loss and perplexity seem to have a trend to decrease. Figures 4.9, 4.10 and 4.11 represent, respectively, the evolution of the loss, perplexity, and learning rate for the entirety of the 3M pre-training steps (using Exponentiated Moving Average). Refer to Annex III for the same graphs but with the original data for comparison.

For pre-training, we implemented a counter for documents seen for each dataset. The goal was to get an idea of the seen documents distribution among the datasets - to get statistics on how many epochs of data were seen. These values are presented in Table 4.5.

Dataset	Total Documents	Seen Documents	Epochs	Sample Weight
ClueWeb PTPT	29M	59,870,058	2.06	0.63
OSCAR PTPT	1.5M	7,609,929	4.88	0.08
ArquivoPT	1.5M	7,598,007	5.07	0.08
OpenSubtitles PTPT	1.2M	5,706,099	4.41	0.06
PTWIKI	820k	9,515,529	11.6	0.1
EuroParl PTPT	1.3M	5,700,378	4.08	0.06

Table 4.5: Documents seen per each dataset during the first 3M steps of GlórIA 1.3B's pre-training - with a **total of 96M documents seen**.

4.4.2 GlórIA 2.7B

As previously mentioned, we replicated our pre-training methodology, training scheme, and parameters, and applied it to GPTNeo's 2.7B parameter checkpoint, producing

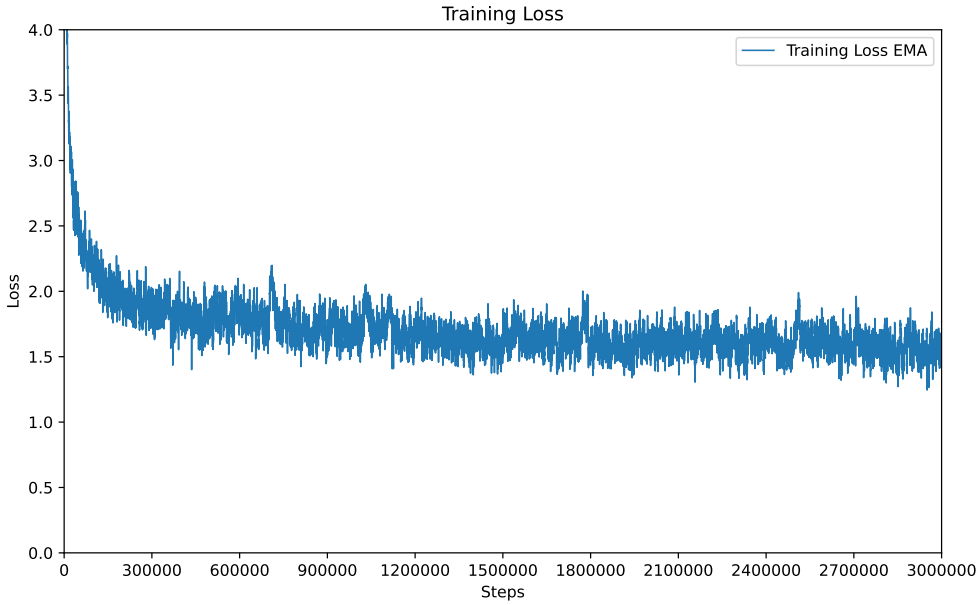

Figure 4.9: **Loss** for the 3M steps of pre-training for GlórIA 1.3B. The blue line is the exponential moving average of the loss with a smoothing factor of 0.90 (*alpha* of 0.1).

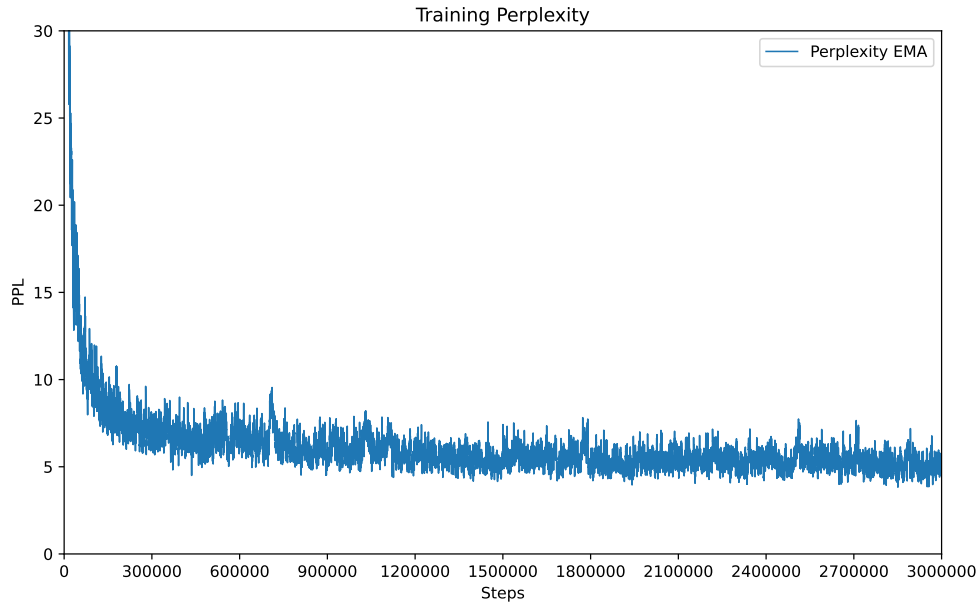

Figure 4.10: **Perplexity** for 3M steps of pre-training for GlórIA 1.3B. The blue line is the exponential moving average of the perplexity with a smoothing factor of 0.90 (*alpha* of 0.1).

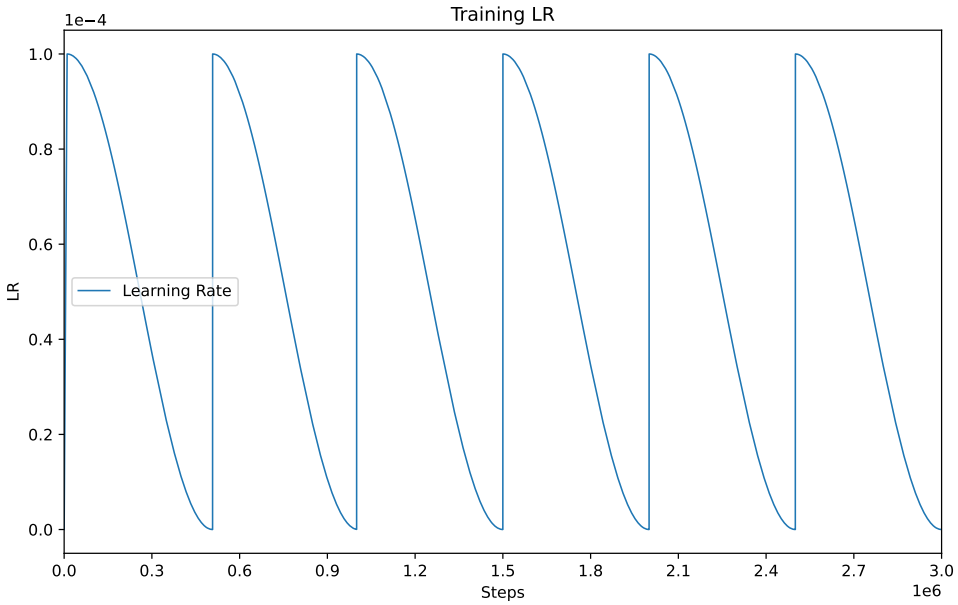

Figure 4.11: **Learning rate** for the 3M steps of pre-training for GlórIA 1.3B - with a hard restart every 500k steps.

GlórIA 2.7B. Compared to the 1.3B version, this one only saw 28M documents - approximately 30% of the total documents seen by the 1.3B version.

Respectively, Figure 4.12 and Figure 4.13 show the pre-training graphs for loss and perplexity. Its behavior and stable training are similar to the 1.3B counterpart, with the exception that the scheduler hard restart may have caused some very small degradation in performance. Nevertheless, the model seemed to regain its momentum, showing a tendency to decrease in both loss and perplexity.

Model	Docs in Corpora (Tokens)	Seen Documents	Tokens Seem (est.)	Epochs	Resources
GlórIA-1.3B	35M (35B)	96M	~95B	2.74	4x A100 40GB
GlórIA-2.7B	35M (35B)	28M	~28B	0.8	7x A100 40GB
Gervasio-PTPT	8M (2.2B)	1.7M	N.A	0.22	8x A100 40GB
Albertina-PTPT	8M (2.2B)	200M	~55B	25	8x A100 40GB
BERTimbau (L)	3.53M (2.68B)	28.24M	~21.44B	8	8x TPU 16GB

Table 4.6: Comparison between both GlórIAversions, and other Portuguese model's corpora size, resources used (GPUs), and seen data during training. The document counts were calculated by multiplying the number of epochs seen per total number of documents, while GlórIA's count was performed by us during training. The token counts are merely estimates given the number of tokens in the original corpora and the documents seen.

We also present in Table 4.6 a rough estimate of how many tokens both our model versions (1.3B and 2.7B) saw during pre-training, in comparison to the other models and their respective estimates. It's **vital to reinforce that these are merely estimates**, calculated from the available information in their respective works and/or papers, and could have a

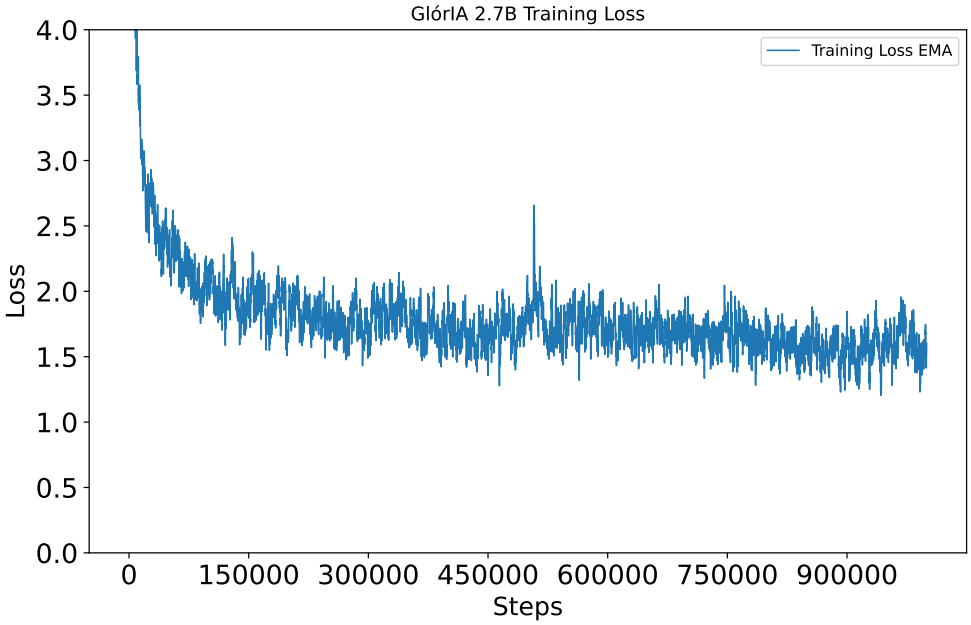

Figure 4.12: **Loss** for the 1M steps of pre-training for GlórIA 2.7B. The blue line is the exponential moving average of the loss with a smoothing factor of 0.90 (*alpha* of 0.1).

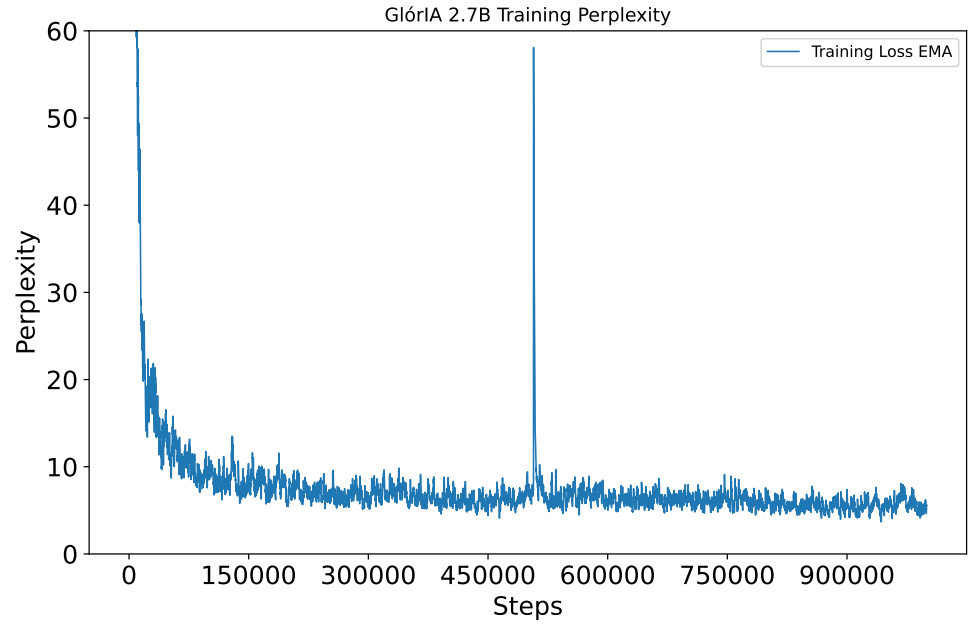

Figure 4.13: **Perplexity** for 1M steps of pre-training for GlórIA 2.7B. The blue line is the exponential moving average of the perplexity with a smoothing factor of 0.90 (*alpha* of 0.1).

slight deviation from the real values.

In sum, GlórIA 1.3B saw (approximately) 95B tokens, close to double the tokens seen by Albertina-PTPT, while its 2.7B counterpart saw only 28B tokens.

5

Assessing a PT Generative Language Model's Text Generation Abilities

Training an autoregressive and generative model without attempting to evaluate its generative capabilities is not only a missed opportunity to demonstrate what it has learned, **but it is a requirement of scientific rigor**. As we've gone over several times over this thesis, there is not, at the time of writing, a Portuguese benchmark that we can use to evaluate tasks based on text generation.

To solve this, we built our own benchmark for the purpose of evaluating generative models - the details of which we'll be seeing in the following sections - but before that, a first and quick look is taken at samples of generated text to give an idea of what the model can actually do.

While this chapter contemplates a qualitative evaluation and a language modeling evaluation, the next one will be focused on evaluating GlórIA on a variety of discriminative downstream tasks.

5.1 Qualitative Evaluation of Text Generation

At the beginning of our evaluation process, neither our own benchmark nor others were available, so the first logical step was to perform an initial qualitative evaluation.

A few 'text prompts' (sentences, small texts, etc.) were created by hand for the model to generate text upon. These prompts were diverse, with the purpose of trying to understand which capabilities or knowledge emerged within the model, given our training and data. Then, we simply looked at the examples of generations given our prompts to try to identify qualities and problems.

The examples we're going to be looking at belong to GlórIA 1.3B. We chose only this version of the model since it was the only trained up to 3M steps, which was our target goal.

5.1.1 Strategies & Parameters Considered

We took advantage of HuggingFace's text generation built-in pipeline for our text generation since it supports the previously described decoding strategies and their respective parameters out of the box.

We defaulted our generated text to a maximum of 50 new tokens. We experimented with different values for the number of beams used, as well as sampling and respective temperatures, ending up finding our best generations when using a mixture of beam-search and top-k sampling - this choice will be reaffirmed in the following sections when we begin talking about an actual quantifiable evaluation of language modeling.

A few *text prompts* were handwritten in Portuguese for this first viewing of generated text. We called these prompts 'completion prompts'. The completion prompts are based on the most simple task, which is to have the model complete the prompt and generate text with the goal of seeing if the written text is coherent.

5.1.2 Displaying Text Generation Examples

We'll now show and analyze some examples of the type of text the model can generate with our prompts, as well as the respective decoding strategies and parameters used. The following samples were generated using GlórIA 1.3B's 3M steps checkpoint, and we chose two different decoding strategies.

5.1.2.1 Greedy Decoding

For our first samples, we chose greedy decoding, observing how the model behaves with temperatures of 1.0 and 3.0. We hypothesized that, by setting the temperature to a higher value, the model would end up generating text that would have more chances of going "off track" with the theme/topic of the prompt/sentence due to increased randomness. This goes for any decoding strategy, not just this one.

For every example that we are going to show, if they would end in the middle of a sentence due to the maximum number of tokens, we added *'(...)'* to the end of the "halfway" sentence. We also substituted *next-lines* with '[NL]' just to show the text in a more compact way. Look at Table 5.1 for this comparison. The prompt is in the **bold**, so the rest of the text is generated by the model. By looking at those few starting examples we can start to observe that the model has correctly learned how to generate solid and meaningful sentences. This means it is accurate in understanding the semantics and morphology of the European Portuguese language.

Changing the sampling temperature parameter leads to very noticeable changes in the generated text, whereas by looking at the first and second prompts, increasing the temperature leads to a significantly different text, sometimes going off-track and away from the initial context as hypothesized previously, or even demonstrating a slight "non-sensical" tendency.

Temperature = 1.0	**Temperature = 3.0**
PROMPT 1.1: **Fernando Pessoa foi um dos poetas mais relevantes de** *toda a história da literatura portuguesa. A sua obra é vasta e diversificada, abrangendo vários géneros literários (romance histórico; poesia lírica), mas também uma vertente ensaística que o próprio autor definiu como "poesia breve" ou prosa poética em verso (...)*	PROMPT 2.1: **Fernando Pessoa foi um dos poetas mais relevantes de** *O Último Esquadro - por Carlos Tê (autor do poema, a abrir para o fado Camané Canta Ravi; tem também temas publicados no suplemento Fado Bipolar que faz em exclusivo este recital -, uma versão das Boavantes/Luxuriç (...)*
PROMPT 1.2: **Os computadores hoje em dia são muito** *mais do que apenas um meio de comunicação. Eles estão a tornar-se cada vez menos pessoais e, por isso mesmo é importante ter uma boa ligação à Internet para poder aceder aos seus conteúdos preferidos ou simplesmente navegar na web sem interrupções constantes!*	PROMPT 2.2: **Os computadores hoje em dia são muito** *simples, rápidos e poderosos por que isso é extremamente útil. Alguns de voces descobriram facilmente esses elementos quando estuda seus note ou até mesmo a uma taxa quase zero na primeira aula: Computadores : como usar o Mac os passos estão listados Como aumentar (...)*
PROMPT 1.3: **A literatura Portuguesa é das mais ricas de todo o mundo. Temos imensos autores de renome nas mais variadas áreas da escrita. Para tirar vantagem do nosso património literário, as escolas** *têm vindo a desenvolver um trabalho notável na promoção e divulgação desta arte tão portuguesa![NL] O projeto "Literatura em Viagem" tem como objetivo principal levar os alunos dos 1º ciclo ao estrangeiro para que conheçam melhor Portugal através deste meio privilegiado: viajar (...)*	PROMPT 2.3: **A literatura Portuguesa é das mais ricas de todo o mundo. Temos imensos autores de renome nas mais variadas áreas da escrita. Para tirar vantagem do nosso património literário, as escolas** *não ficaram a ignorar isto tão simples dos escritores Portugueses serem muitos os talentos no panorama Internacional[NL] Contém informações para quem tenha uma educação literária e artística ao mesmo tempo excelente na capacidade narrativa... em termos teóricos são muitas vezes melhores uns que outros - há casos (...)*

Table 5.1: Comparison between the same prompts, using **different sampling temperatures for Greedy Decoding**.

5.1.2.2 Beam-Search With Top-K Sampling

As for our second strategy, we used a mixture of **4 beams with top-k sampling for the first 50 tokens**, introducing non-determinism to our generation. Observe our examples and sampling temperature comparison for this strategy in Table 5.2, and a direct comparison between our two chosen decoding strategies with new prompts in Table 5.3. As expected, changing the decoding strategy to beam search and top-k leads to a general increase in the quality of the generated text, since we track multiple possible generations through the beams - maintaining its good and solid quality when considering European Portuguese syntax, semantics, and coherence.

However, this time, increasing the temperature does not make the model go completely off track when generating text. While its text is different, the model is more capable of staying grounded in context and coherence, when considering some present ambiguity

Temperature = 1.0	**Temperature = 3.0**
PROMPT 3.1: **Fernando Pessoa foi um dos poetas mais relevantes de** *toda a história da literatura portuguesa.[NL] Biografia.[NL] Nasceu em Lisboa, no seio de uma família ligada ao meio intelectual e literário. O seu pai era o escritor José Maria de Eça de Queirós (1825-1890) e a (...)*	PROMPT 4.1: **Fernando Pessoa foi um dos poetas mais relevantes de** *qualquer uma das gerações, e talvez o único autor universal. E é muito interessante ver os resultados da leitura dessa imensa biblioteca, traduzida até à geração (?) que não sabia como se referir àquele poeta, ou àquela cidadezinha na terra distante.*
PROMPT 3.2: **Os computadores hoje em dia são muito** *do que apenas uma ferramenta de trabalho. Eles também podem ser usados para fins educacionais, científicos e de entretenimento.[NL] Se você está pensando em comprar um computador novo, existem muitos fatores que você deve levar em conta ao escolher o modelo certo para*	PROMPT 4.2: **Os computadores hoje em dia são muito** *importantes para se ser uma pessoa organizada e responsável. Apesar disto, muitas pessoas preferem a tranquilidade dos computadores pessoais (PC).[NL] Apesar de estar à procura de um emprego no escritório, está a pensar nos benefícios do escritório virtual: mais conforto, melhor produtividade (...)*
PROMPT 3.3: **A literatura Portuguesa é das mais ricas de todo o mundo. Temos imensos autores de renome nas mais variadas áreas da escrita. Para tirar vantagem do nosso património literário, as escolas** *têm vindo a desenvolver um conjunto de atividades que visam dar a conhecer aos seus alunos os grandes escritores portugueses.[NL] No ano letivo 2021/2022, as Escolas Portuguesas no Estrangeiro (EPE) estão a desenvolver uma série de atividades de promoção da leitura e (...)*	PROMPT 4.3: **A literatura Portuguesa é das mais ricas de todo o mundo. Temos imensos autores de renome nas mais variadas áreas da escrita. Para tirar vantagem do nosso património literário, as escolas** *poderão ter de reforçar o apoio ao estudo em português através de cursos linguísticos para os docentes interessados. O objetivo é oferecer uma alternativa que ainda não existia para complementar a aprendizagem dos alunos e tornar os livros mais atrativos para o seu gosto. Se tens. . . (...)*

Table 5.2: Comparison between the same prompts, using **different sampling temperatures for our second strategy which mixes Beam Search with Top-k Sampling**.

in the prompts that we used.

Our observations lead us to conclude that, besides the non-deterministic nature, **Beam Search with Top-K sampling is the most adequate strategy to generate text with GlórIA 1.3B**. They also demonstrate that **GlórIA can deal with different and diverse topics, a reflection of the corpora diversity**.

5.1.3 "Internet Article" Syndrome & Misinformation Disclaimer

Even if most of the text generated by the model is good, some "*artifacts*" can still be found. By "*artifacts*" we mean spans of text that are essentially out of context or factually wrong.

Some examples of these are "ghost signatures", where the model "signs" the text like a news article using "ghost authors", or "web 4th-wall breaks" where the writing changes to a *blog-ish* tone, addressing "website visitors". This behavior was observed in some of our test samples. Sometimes, at the end of the generation, the model takes a slight turn into a *meta news article* category and makes assumptions or affirmations based on possible

Greedy Decoding (T=1.0)	BeamSearch w/ Top-K (T=1.0)
A cozinha portuguesa é rica em aromas e sabores. Existem muitos pratos *que são típicos de Portugal, mas também existem outros mais originais ou até mesmo exóticos como o bacalhau com natas (que eu adoro)![NL] Para além disso a gastronomia nacional tem muito para oferecer aos nossos visitantes: desde os doces às saladas* (...)	**A cozinha portuguesa é rica em aromas e sabores. Existem muitos pratos** *tradicionais, como o bacalhau com natas, o polvo à lagareiro ou a feijoada de búzios, mas também existem novos pratos que estão a ganhar cada vez mais adeptos. É o caso dos rojões à moda do* (...)
Os carros são considerados uma das maiores criações da engenharia. Os avanços no transporte de seres humanos *e máquinas, desde o século XIX até aos dias atuais têm sido enormes: a invenção do automóvel revolucionou os meios urbanos para transportar pessoas ou mercadorias; as estradas foram construídas em torno dos automóveis que permitiam um acesso rápido à cidade (e vice-versa (...)*	**Os carros são considerados uma das maiores criações da engenharia. Os avanços no transporte de seres humanos** *e máquinas, bem como as novas tecnologias, permitiram que o automóvel se tornasse um meio de transporte muito mais rápido e eficiente do que qualquer outro tipo de veículo.[NL] No entanto, a maior parte dos automóveis atuais ainda é baseada em motores de combustão interna (...)*
Surfing é um dos desportos mais praticados nas zonas costeiras de Portugal. Os adeptos do surf *podem encontrar ondas com uma altura máxima entre os 10 e 15 metros, sendo que a maioria das praias portuguesas tem pelo menos 1 metro ou 2 m (algumas têm até 3m).[NL] O Surf em geral pode ser praticado por pessoas não só surfistas mas (...)*	**Surfing é um dos desportos mais praticados nas zonas costeiras de Portugal. Os adeptos do surf** *em Portugal são cada vez mais e o número de praticantes tem vindo a aumentar nos últimos anos.[NL] O surf é um desporto que se pratica com uma prancha, geralmente de madeira, para deslizar sobre as ondas. É praticado em todo o mundo, (...)*
A literacia financeira permite às pessoas gerirem o seu dinheiro de forma mais responsável. Para aumentar a literacia financeira, *é importante que as crianças aprendam sobre os conceitos financeiros e como funcionam alguns dos seus mecanismos básicos para gerir melhor esse tipo específico do património financeiro das famílias portuguesas:[NL]• A importância da poupança; • O valor monetário em cada momento (por exemplo (...)*	**A literacia financeira permite às pessoas gerirem o seu dinheiro de forma mais responsável. Para aumentar a literacia financeira**, *é importante que as crianças e os jovens aprendam a gerir o seu dinheiro de forma adequada.[NL] Aprender a gerir o seu dinheiro vai ajudá-lo a:[NL] Compreender como funciona o dinheiro;[NL] Identificar diferentes fontes de rendimento (...)*

Table 5.3: A comparison, using new prompts, to help demonstrate the difference in output text between the two chosen strategies (with a sampling temperature of 1.0).

misinformation, where it can hallucinate or provide factually wrong information, as seen in the previous prompt 3.1 (Table 5.2), by mistaking Fernando Pessoa's father by another famous Portuguese author (Eça de Queirós).

These are well-known issues amongst recent *generative* language models (Pan et al. [70] and Agrawal, Mackey, and Kalai [71]). In our model's specific case, we hypothesize that these *artifacts* happen due to unfiltered content from our web crawls and news articles - whose biases have not been researched since it wasn't a focus of this thesis. These biases could be conceptual (e.g.: based on ideas, opinions, or encyclopedic knowledge), or concrete (e.g.: writing styles, text-specific artifacts like an article's authors' signature), and sometimes we can overcome them by carefully changing certain text prompts. So one could expect our model to have certain biases that may lead it to, for example, following a news article or blog post style of written text (something which may be, or not, a desired behavior depending on the use case where this model could be applied), or even generate plain wrong information since it is not able to distinguish fact from fiction. Another recognized bias is when GlórIA sometimes turns the text into a dialogue or conversation with himself. We believe this is due to the sampling weight we gave to the OpenSubtitles dataset, which may have been too high, hence the model learned to create small and out-of-context short conversations. Besides all of this, we can't forget the possibility of the model demonstrating unstudied toxic behavior.

It is important to recognize the existence of areas that study and research techniques and barriers that mitigate and avoid such issues, but they're not one of the current focuses and goals of this work.

5.2 Evaluating Text Generation with CALAME-PT

A qualitative evaluation would not be sufficient. The only way to systematically measure its performance was to produce a benchmark from scratch, and so we did.

Inspired by its English version, LAMBADA (Paperno et al. [72]), we created the **very first Portuguese benchmark for evaluating generative models, called CALAME-PT (Context-Aware LAnguage Modeling Evaluation for Portuguese)**. The goal of this benchmark is to assess the ability of a model to guess the final word given the context/sentences that come before - **it's a language modeling task**. It comprises **2076 samples**, with only a test set, meant to be used in a zero-shot setting.

5.2.1 Preparing a new benchmark

In LAMBADA (Paperno et al. [72]), they split the text into two parts: the context and the target sentence (the last sentence). The idea behind this is that neither the model nor a person could guess the final word only by looking at the target sentence, so it would have to look at the whole text to be able to guess. Our benchmark takes this into account, but it does not follow this rule exclusively, meaning some last words can be guessed regardless, e.g.

"*The color of the sky is* ___", where the final word would be "*blue*" - emphasizing 'universal empirical knowledge' (since everyone knows the sky is blue). In the end, this benchmark contains a mix of LAMBADA's idea and ours, to increase diversity and difficulty. Thus, it is composed of multiple, small texts, whose context should be enough to guess the final word. Each text's final word can either be at the end of a sentence (meaning the text ends), or the text can end "abruptly" and the expected target word is in the middle of a sentence - to challenge the models to guess words that can either be the literal end of the text or words that can appear in the middle of a certain context. Nevertheless, everything that comes before it would have the necessary context to correctly guess - implying that models need to be able to keep track of context along sequences of text. In the end, we have two types of contexts: contexts where the target word appears at least once, and contexts where the target word does not appear at all. We believe these represent a fair domain of "contextual guessing" that one deals with when working with Portuguese.

To build CALAME-PT, we split the process into two parallel tasks.

- For the first one, a total of 406 samples were handwritten.
- As for the second one, we built a pipeline that prompts OpenAI's GPT-3.5 (using their API) with documents and asks it to rewrite a new and smaller text inspired by the document. This was done by preparing a small prompt (template), where we would insert a document/pre-existing text, followed by our rewrite request. We would also ask GPT-3.5 to anonymize, in an attempt to remove any known names, events, dates, countries, etc. - with the goal of trying to reduce as much the need for encyclopedic knowledge. The documents we used to prompt the API were obtained from smaller evaluation sets from our pretraining data, that were created at the beginning of this work, each one with 10k samples. **This data was never used during pretraining, since we didn't perform in-training evaluation, having only been used for this purpose**. So, at the end of the pipeline, we obtain the generated texts, from which we would "cut" the last word as a string, and write the dataset to disk, with columns: ID, Context, and Last Word (the last word being the ground-truth).

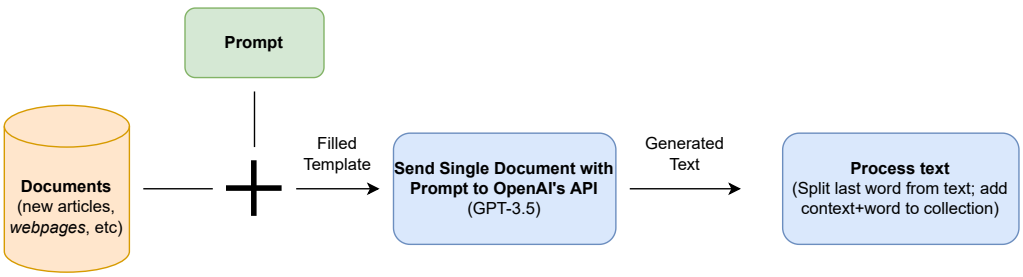

Figure 5.1: Simplified diagram of the pipeline used to generate texts.

We employed ChatGPT since it allowed us to increase the scale of this benchmark. By using it, we managed to save a lot of production time, since writing all examples by

CALAME-PT Set	Context	Last Word
Generated	Um médico foi detido pela Polícia por falsificação de documentos. A sua detenção está relacionada com uma investigação sobre possíveis burlas ao sistema de saúde e falsificação de receitas médicas. O indivíduo será interrogado e sujeito a medidas de coação, com o intuito de perceber como os documentos eram	**falsificados**
Generated	No contexto apresentado, várias organizações do trabalho, como sindicatos e associações sindicais, estão envolvidas em negociações e revisões contratuais com várias empresas. Essas interações destacam a importância das negociações coletivas para garantir condições justas de trabalho. As organizações do trabalho trabalham em conjunto para representar os interesses dos	**trabalhadores**
Handwritten	A empresa de software lançou uma atualização importante para o seu sistema operacional. Os utilizadores gostaram das melhorias de desempenho. A atualização foi bem recebida pela	**comunidade**
Handwritten	A agricultura tem visto um aumento na produção de alimentos graças à utilização de pesticidas. Estes ajudam a combater pragas que danificam as plantações. Contudo,muitas pessoas preocupam-se com o impacto no meio ambiente por causa do uso destes	**pesticidas**

Table 5.4: Few examples of samples from the first iteration CALAME-PT (post-human review). We present two examples that were generated by our pipeline, the second of which was corrected with a small rewrite, and two handwritten examples.

Dado o seguinte contexto: {DOC HERE} Escreve um pequeno texto inspirado pelo contexto com poucas frases. Não deves mencionar nomes de pessoas ou países, eventos, marcas, e datas (dias, anos e horas).

Table 5.5: The chosen prompt that was fed to Chat-GPT to generate a new, smaller text based on our documents.

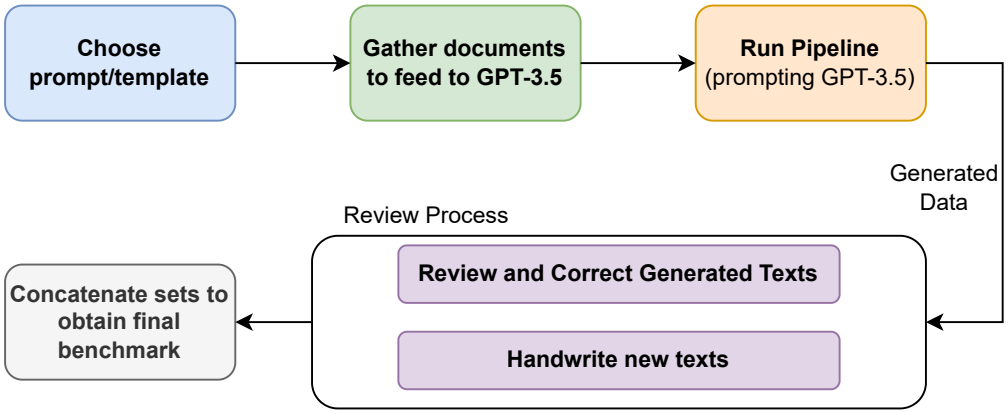

Figure 5.2: Overview of the entire CALAME-PT preparation process. The mentioned pipeline refers to Figure 5.1.

hand would require massive efforts for a single person. ChatGPT's generations, while still requiring revision, offer good samples.

At the end of this process, we had two sets: the handwritten set and the generated set. Table 5.4 shows a few samples from both sets. The generated set was initially composed of 2.5k samples, but a human review was performed on all these, to further ensure quality: removing bad samples, anonymizing things the API missed, or even rewriting some texts to change the last word for something more adequate. This last action would involve trying to change the text so that the last word wouldn't be something vague or ambiguous, like an adjective, a temporal adverb, etc. - always while keeping the context. At the end of the review process, we were left with 1670 samples from the generated set.

After performing this review, we concatenated the handwritten set and the generated set to give birth to the final version of CALAME-PT. **The entire process cost under 7 euros**, including testing and generating with OpenAI's GPT-3.5 (API) - taking into account the human review of the generated set was performed by a single person (the author).

5.2.2 CALAME-PT Caveats & Difficulties

The first difficulty was anonymizing the data and trying to remove encyclopedic knowledge-dependent texts. Not all of the names of people, countries, etc., could be removed, but an attempt was made to either remove the samples completely or perform small rewrites so that the context remains but the last word is independent of said names/entities. This difficulty came mainly from the source of the documents we used for the generation, some of which were from the PTWIKI dataset, others from OSCAR-PTPT, and the remaining from our Arquivo-PT dataset - and sometimes GPT-3.5 had trouble anonymizing the data even when asked to rewrite a new text.

The second main difficulty was based on the GPT-3.5's struggle to generate accurate European Portuguese text. While his Portuguese was correct, it would always shift to Brazilian Portuguese in most generations. This led to our data having a mixture of both

PT-PT and PT-BR branches of the Portuguese language: something that could either increase the difficulty of the task, or harm performance in the long run (when increasing the size of the benchmark). Addressing these difficulties and risks are potential future work goals.

5.2.3 Experimenting & Evaluating on CALAME-PT

We picked both our 1.3B and 2.7B versions of GlórIA and Gervasio-PTPT[1] as the models to be evaluated and compared, as they're the only models capable of text generation for the PT-PT language (to the extent of our knowledge).

To evaluate a model on CALAME-PT, we need to see if the first word a model generates given the context, matches the target last word (the ground truth). Since we will be comparing actual strings, as this is a text-generation task, we picked the exact match as the metric to quantify performance.

To make sure we have a robust evaluation, on a first-of-a-kind benchmark, we defined an experiment space. This space consists of performing multiple evaluations, with different decoding strategies. Firstly, we evaluated every model on the entire CALAME-PT benchmark, followed by separate evaluations on the generated and handwritten sets. At the same time and for each set, we chose greedy decoding as the first strategy, and a mixture of beam-searching with top K sampling as the second strategy - giving us two different types of experiments to perform.

Due to the deterministic nature of the greedy decoding strategy, we only performed one evaluation. For the beam-search and top-k hybrid strategy, we evaluated each model 3 times for each "set" of the CALAME-PT benchmark - since this second strategy is not deterministic and can provide different results with the same parameters. We evaluated 3 checkpoints from our model, respectively the 1M, 2M, and 3M steps checkpoints, to also analyze their evolution throughout training.

For the second strategy, we used 4 beams and chose the top 50 most probable tokens to sample from (top-k 50). A temperature of 1 was used, together with a repetition penalty of 2 (as inspired by Keskar et al. [73]) to discourage token repetition.

In practice, we iterate over CALAME-PT's samples, and for each sample, we pass the context to our model. This will give us a string with the generated text. Since we're limiting the text generation up to 5 tokens (since one word can be composed of multiple tokens), we process that string and consider ONLY the first word, which is the word we will be comparing against the ground truth (the target word). For this comparison (a simple string comparison), we ignored casing and accents, due to the presence of PT-BR and PT-PT words - meaning the same word can appear with either no accent or a different accent in the same letter, thus they were ignored in order to not punish the models for guessing the right word but with the wrong accent.

[1] https://huggingface.co/PORTULAN/gervasio-ptpt-base

The calculated exact match value is the sum of all the correctly guessed words, divided by the total number of samples.

5.2.4 Final Results & Comparisons

Greedy Decoding Strategy Evaluation			
Models	**CALAME-PT ALL**	**CALAME-PT GENERATED**	**CALAME-PT HANDWRITTEN**
GlórIA 1.3B (1M CHK)	35,07	37,36	25,62
GlórIA 1.3B (2M CHK)	35,93	38,14	26,84
GlórIA 1.3B (3M CHK)	**36,61**	**38,86**	**27,34**
Gervasio-PTPT	19,03	19,88	15,52
GlórIA 2.7B (1M CHK)	35,36	37,84	25,12

Table 5.6: CALAME-PT evaluation scores (Exact-Match in percentages) comparison for the **greedy decoding strategy**.

BeamSearch+Top-k Strategy Evaluation			
Models	**CALAME-PT ALL**	**CALAME-PT GENERATED**	**CALAME-PT HANDWRITTEN**
GlórIA 1.3B (1M CHK)	50,99	53,21	38,75
GlórIA 1.3B (2M CHK)	51,80	53,69	41,95
GlórIA 1.3B (3M CHK)	**52,79**	**55,39**	**42,61**
Gervasio-PTPT	44,01	45,97	34,90
GlórIA 2.7B (1M CHK)	52,20	54,57	40,40

Table 5.7: CALAME-PT evaluation scores (Exact-Match in percentages) comparison for the beam search with **top-k sampling strategy**.

Across all the models, decoding strategies, and CALAME-PT sets, the 3M steps checkpoint for GlórIA 1.3B achieved the highest performance by relevant margins when compared to Gervasio-PTPT. Fortunately, these results are what we initially expected them to be - a reflection of our model's larger size and increased data size and diversity.

However, following the greedy strategy, the 1M checkpoint for the 2.7B version outperformed the 1M checkpoint for the 1.3B version. Looking at the Beam Search+Top-k strategy, we notice similar comparisons, where now the 1M checkpoint of GlórIA 2.7B outperforms the 1M and 2M checkpoints of GlórIA 1.3B in all sets - except the 2M checkpoint

in the handwritten set.

Since they share the same architecture, we believe that if GlórIA 2.7B went under further pre-training, it has the potential to outperform the 3M checkpoint from the 1.3B version in all the experiments. Table 5.6 and Table 5.7 present, respectively, the results for the greedy decoding strategy, and the results for the Beams+Top-k strategy. The results shown in the second strategy are the average of the 3 evaluations performed for each case.

As for GlórIA's (1.3B) performance evolution, our analysis shows that performance for this task can improve with further training. Please consult Figure 5.3 where we present this evolution for both of our decoding strategies and the respective subsets.

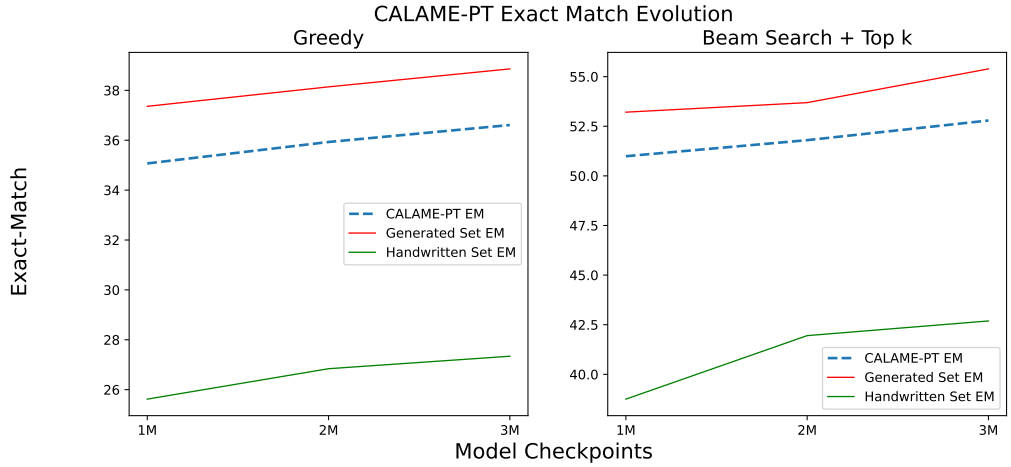

Figure 5.3: Evolution of GlórIA 1.3B's performance on CALAME-PT. Evaluated 3 checkpoints (1M, 2M, and 3M) for our two chosen decoding strategies. We also present the evolution for the "inner" sets of CALAME-PT: generated and handwritten.

6

Evaluating Models on Supervised & Discriminative Tasks

This chapter will provide insight into how and which **supervised evaluation** procedures were followed. We will present and analyze the results for each of the chosen evaluation tasks and comparisons with other models, since they'll be part of the core of our model's quality indicators.

The following tasks are discriminative tasks - they do not require text generation - thus encoder-based models are more inclined to achieve the highest scores. This is expected due to the nature of their pre-training objectives (like masked language modeling) since they assimilate information bi-directionally. For example, large decoder-based LMs with billions of parameters like Sabiá-7B or Sabiá-65B (Pires et al. [13]) still struggle to beat previous state-of-art results set by encoder-based models on some discriminative tasks like ASSIN-2, especially because it was evaluated in a text-generation format using inference. Nevertheless, it is important to assess how our model compares to encoder models on alternative NLP downstream tasks.

For this chapter's evaluations, we only contemplated GlórIA 1.3B. This was due to logistics and time issues: the 1.3B allowed for a more robust study since it had more checkpoints that could be evaluated due to having been trained for more time, and adding GlórIA 2.7B to the experiments would require extra time and resources. **In this section, when discussing our model, we're referring to the 1.3B version.**

6.1 Defining the Evaluation Methodology

As a general note, for every task we followed the same methodology. First, chose the models we wished to compare against our own model. Then, via "hand" testing, we found hyperparameter sets that were optimal and could be shared across different models' fine-tunings for comparability. From these hyperparameter sets, we built experiment spaces for each task and/or sub-task and fine-tuned each model for each hyperparameter set in that experiment space. Different checkpoints from our model were also evaluated,

to see how the performance would evolve as the model was trained.

This would give us a fair amount of experiments from which we could draw results, comparisons, and conclusions. Also, for our own model, we ran these experiments for several checkpoints to see how the model would evolve up the 3M steps checkpoint.

We'll be going into these topics in detail in the following sections.

6.2 ASSIN-2 Tasks

The first benchmark we'll consider is the ASSIN-2 dataset (Real, Fonseca, and Oliveira [74]), also known in Portuguese as *Avaliação de Similaridade Semântica e Inferência Textual*. Its goal is to train and evaluate models for assessing two types of relations between two simple sentences: Semantic Textual Similarity (STS) and recognizing Textual Entailment (RTE). The first task pertains to measuring the magnitude of semantic equivalence between those two sentences, while the second task is based on being able to recognize if the first sentence entails (*leads to*) the second one.

ASSIN-2 is made of relatively simple sentences in PT-BR. We recognize the differences between PT-PT and PT-BR, but since it's composed of relatively short sentences, it acts as a first evaluation for our Portuguese model's capabilities. Its training, validation, and test sets are composed of respectively, 6500, 500, and 3000 sentence pairs with annotations for both STS and RTE tasks.

The labels for the similarity task are continuous values from 1 to 5, where a lower value has different (semantic) meanings, and a higher value indicates they're more semantically equivalent - this task is evaluated using Pearson's Correlation as the primary metric. The entailment task, only uses 2 labels: entailment and non-entailment - for its evaluation a macro-F1 score is used as the primary metric and accuracy as the secondary metric. Table 6.1 presents a few examples for this task.

To calculate these metrics and perform the correct evaluation, we integrated ASSIN-2's official evaluation scripts into our codebase.

Table 6.1: Examples of pairs of sentences from the ASSIN2 task.

Sentence 1	**Sentence 2**	**RTE Label**	**STS Label**
"Uma garrafa está sendo lambida pelo gato"	"O gato está lambendo um objeto"	1	4
"Uma pessoa não está descascando uma banana"	"Um homem está lendo um e-mail"	0	1
"As pessoas estão andando de bote e remando"	"Algumas pessoas estão navegando de bote"	1	3.75

6.2.1 Fine-tuning on ASSIN

We fine-tuned our models in both tasks at the same time, but not because it's deemed a shared-task benchmark - meaning it was made publicly available for a contest, for example. Nevertheless, our results remain comparable between models since they were all fine-tuned using this exact protocol, which is per BERTimbau's protocols ([51]). We did this by **attaching two linear layers on top of the models**. Afterward, we **calculated our final loss using two losses, one for each task**. The final loss is the sum of both of these losses. For our model, this was implemented on top of the existing *HuggingFace*'s implementation of the *GPTNeoForSequenceClassification* model - where classification is done on the last token. **The models we will be comparing against were also finetuned following this same protocol to ensure comparability**.

For the entailment task, we pass the pooled logits (from the inner outputs of the model) through a linear layer with dropout, and calculate the cross-entropy loss. For the similarity task, we do the same but using a different linear layer and calculate the Mean-Squared-Error as the loss.

Internally, the model outputs *hidden logits*, which are a series of hidden representations of the tokens. To perform a sequence classification task for our model, we only consider the last hidden representation h_n (the last token in the sequence). Afterward, this representation is fed into a linear layer to obtain the final logits.

$$O_i = W_j \cdot h_n + b_j$$

$\mathbf{W}_j$ is a weight matrix, O_i represents the raw scores (logits) for each label i, and b_j is a bias vector. Adapting this function to our ASSIN2 tasks' fine-tuning, we can describe them as:

$$O\_Sts = W_s \cdot h_n + b_s$$

$$O\_Rte\_i = Softmax(W_r \cdot h_n + b_r)$$

Where O_STS represents the logit/raw score for the predicted similarity score and O_RTE_i represents the logits for each label present in the RTE task (entailment and not entailment - i representing the label). Since we use different linear layers, each one has its respective bias vectors b, and weight matrices W, whose sizes are $numclasses \cdot model\_hidden\_size$. A softmax is applied to the RTE logits to obtain the final predicted labels.

Input preparation is simple. We tokenize the pair of sentences and pass the corresponding entailment and similarity labels to the model. We used a max sequence length of 128 since ASSIN2's sentences are fairly short.

6.2.2 ASSIN-2 Experimental Space

Since ASSIN2's main metrics are the F1 score and the Pearson correlation, during finetuning we saved a checkpoint with the best values per metric. Essentially, for one fine-tuning process, we had a *checkpoint-best-f1* and a *checkpoint-best-pearson*, where each one was saved

during training according to the highest value seen per metric (during validation). Each checkpoint would be evaluated, and each would output its values for the F1 and Pearson scores. To obtain our final values, we calculated the average of the metrics between those two checkpoints.

$$F1 = (\textit{checkpoint-best-f1's}F1 + \textit{checkpoint-best-pearson's}F1)/2$$

$$Pearson = (\textit{checkpoint-best-f1's}Pearson + \textit{checkpoint-best-pearson's}Pearson)/2$$

To evaluate our model on this and the remaining tasks, a set of hyperparameters was defined for each of these tasks. Each set of hyperparameters constitutes our experimental space. **For this and the following tasks, we fine-tuned every model using the train set, performed an in-training evaluation with the validation/dev set, and used the test set for the final evaluation, from which we collected our final scores.**

For this task, we compared our PT-PT model against BERTimbau-Large (Souza, Nogueira, and Alencar Lotufo [51]) since it is trained on PT-BR data - excluding other PT-PT models, believing it wouldn't make sense to compare against them in Brazilian Portuguese benchmark. We chose checkpoints 1M, 1.5M, 2M, and 3M from our model for evaluation. To make sure we perform a robust evaluation, we defined an experimental

Hyperparameter Set	BS	LR	Warmup Steps	Epochs	GA Steps	Scheduler
A0	32	1e-5	0	10	2	linear
B0	32	1e-5	0	10	2	constant
C0	32	1e-5	0	5	2	linear
D0	32	1e-5	0	5	2	constant
E0	32	1e-6	0	10	2	linear
F0	32	1e-6	0	10	2	constant
G0	32	1e-6	0	5	2	linear
H0	32	1e-6	0	5	2	constant

Table 6.2: Hyperparameter sets for the ASSIN-2 task. All of these would be used to fine-tune each and every comparison model (including our own).

space. This means we selected some hyperparameters to vary and chose certain combinations of hyperparameters with their variations. These combinations are what we call 'hyperparameter sets', and they are used to perform multiple fine tunes on our target models. These hyperparameter sets, including the chosen variable hyperparameters, change from task to task, since we performed some previous tests to find out which ones were optimal for the target models - don't mistake this for "different models - different sets". Each model was fine-tuned using the same hyperparameter sets as one another for comparability, and in this experimental space, we defined two training sessions, where we tested for seeds 41 and 42.

Inside these hyperparameter sets, we tested for learning rates *1e-5* and *1e-6*, for 5 and 10 epochs, and linear and constant schedulers. From these variations, we prepared 8

sets of hyperparameters that we used. A weight decay value of 0.01 was used in every experiment. Consult these sets in Table 6.2.

6.2.3 ASSIN-2 Results

6.2.3.1 ASSIN-2 Metrics Results

Model	F1	Accuracy	Pearson
GlórIA 1.3B	0,8960	0,8967	**0,8510**
BERTimbau-Large	**0,9020**	**0,9020**	0,8460
Sabiá-7B	0,6487	N.A	0,1363
Sabiá-65B	0,8807	N.A	0,6329

Table 6.3: These are the highest recorded values for the evaluation metrics for the ASSIN-2 task across all experiments (all fine-tunes). Evaluation performed on the test set. Our model competitively loses on F1 and Accuracy scores but manages to come on top when it comes to Pearson's. Sabiá's scores are only here for viewing and are not comparable - obtained from Pires et al. [13].

Looking across all of the fine-tunes that were made, for both seeds, all the hyperparameter sets, and all the models (including our own model's multiple checkpoints), we collected the peak values that we registered upon evaluation with the test set. By looking at Table 6.3, we can see that **our model has a competitively lower performance on the F1 and Accuracy scores when compared to BERTimbau, managing to have a competitively higher Pearson value**.

We found the highest results for GlórIA and BERTimbau when using a learning rate of *1e-5*, training for 10 epochs, and using a constant scheduler - corresponding to **hyperparameter set B0**.

We believe these values to be a testament to possible performance gains acquired by our model's larger size, and richer and more diverse training data - when taking into account that BERTimbau was trained on PT-BR data, and ASSIN-2 is made exclusively of PT-BR sentences.

Following these results, it is important to also look at the experiments that were made, to understand which hyperparameters were more adequate for a certain model or for this specific task, so we'll be showing the entire experimental space for the ASSIN-2's several finetunes. The finetunes and respective results for our multiple model checkpoints and BERTimbau's finetunes can be viewed in Annex IV. The highlighted values (in **bold**) are the peak values registered for both models, shown previously.

We are not able to compare our results with Sabiá's because it was evaluated using few-shot prompting (inference) - a completely different evaluation - so no conclusions can be made. Even if we wanted to evaluate Sabiá using our protocols, it is a closed-source model and not publicly available. However, since our model is a decoder, in the future it would make sense to follow Sabiá's protocol and evaluate our model in a few-shot environment. The goal of this would be to assess how an autoregressive decoder-based

model would perform when trying to solve a discriminative task using causal language modeling - either using ASSIN-2 or a new benchmark (in Portuguese) purposely designed for this.

Different training seeds lead to different shuffles in the training data between epochs, which in turn changes the order in which the model sees that data, and that could influence its performance. So it is important to make sure the model's behavior is not erratic and remains stable across seeds - as it should not be dependent on them. The differences in the scores between the same experiments on different seeds are small to the point where they can be neglected, supporting our model's stability. To further confirm this, we decided to look at how the evaluation loss evolves by comparing the same experiments but with different seeds - as observed in Figure 6.1 where we compare GlórIA's Checkpoint 3M experiments with hyperparameter set B0. We looked at the exponentiated moving average, and in both seeds, the tendency is the same - meaning the same behavior across seeds.

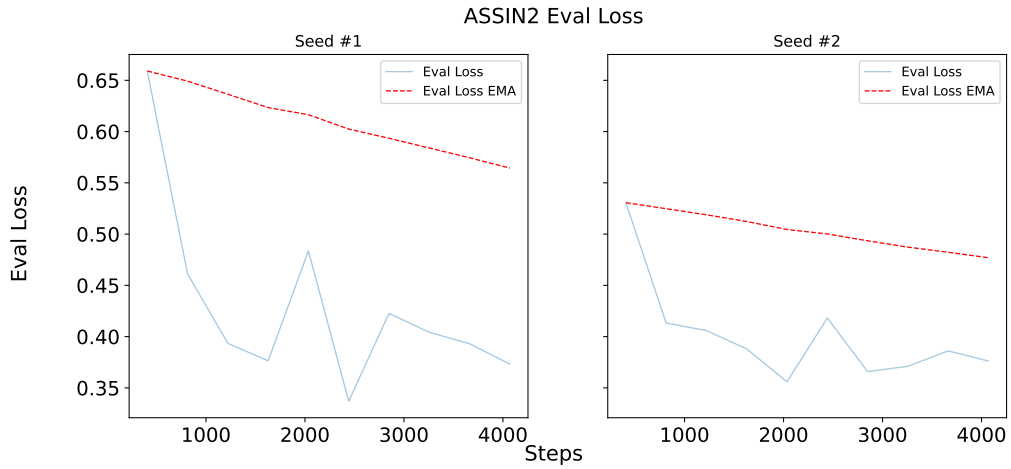

Figure 6.1: Eval loss comparison between GlórIA's 3M checkpoint experiments with hyperparameter set B0 on two different seeds. The exponentiated moving average (EMA) uses a smoothing factor of 0.95.

6.2.3.2 Relevant Notes on ASSIN-2 Convergence

By looking at each model's best runs - namely the ones that obtained the highest final scores, we've found that our model's F1 score increases in an unstable way (during in-training evaluation - using validation/dev set) when compared to BERTimbau's evolution of F1. We believe that this instability may be related to our model's slower convergence on the regular training loss (observed in Figure 6.2), compared once again with BERTimbau's - since ours starts at a much higher value, but ends up with a much smaller loss than BERTimbau's. However, by observing the exponentiated moving average of the F1 scores, both models show rather similar behavior and further potential to reach higher values (Figure 6.3). Meanwhile, both models' Pearson's scores appear stable as the training goes on (Figure 6.4), following the same tendency to further increase.

We also present the evolution of the F1 and Pearson scores (evaluated with the test set) along our model's checkpoints in Figure 6.5. A slight tendency to increase is observed in both metrics and may be a sign that performance can increase for this task with further pre-training. The values used are the average of all the experiments for that given model checkpoint and seed.

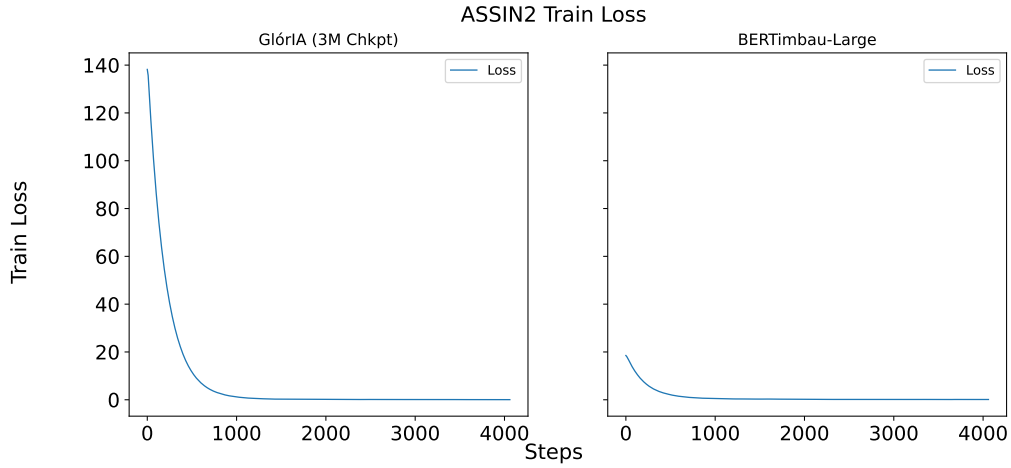

Figure 6.2: Comparison between GlórIA's and BERTimbau's losses during training (dev set). Obtained from the runs where their F1 and Pearson values peaked amongst the entire experiments. Using hyperparameter set B0 for both models and using the 3M checkpoint of our model. Using exponentiated moving average with a smoothing factor of 0.95.

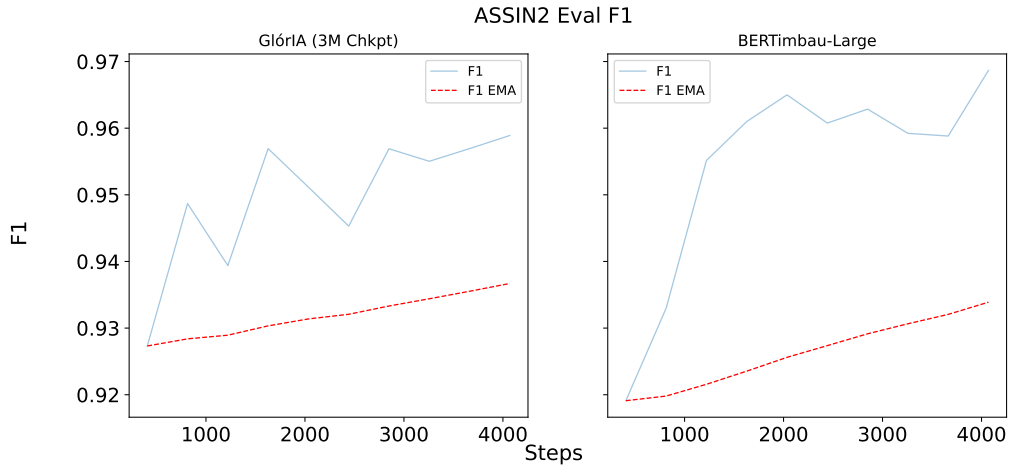

Figure 6.3: Comparison between GlórIA's and BERTimbau's evaluation of F1 score during training (dev set). Obtained from the runs where their F1 and Pearson values peaked amongst the entire experiments. Using hyperparameter set B0 for both models and using the 3M checkpoint of our model. The red line is the exponentiated moving average with a smoothing factor of 0.95.

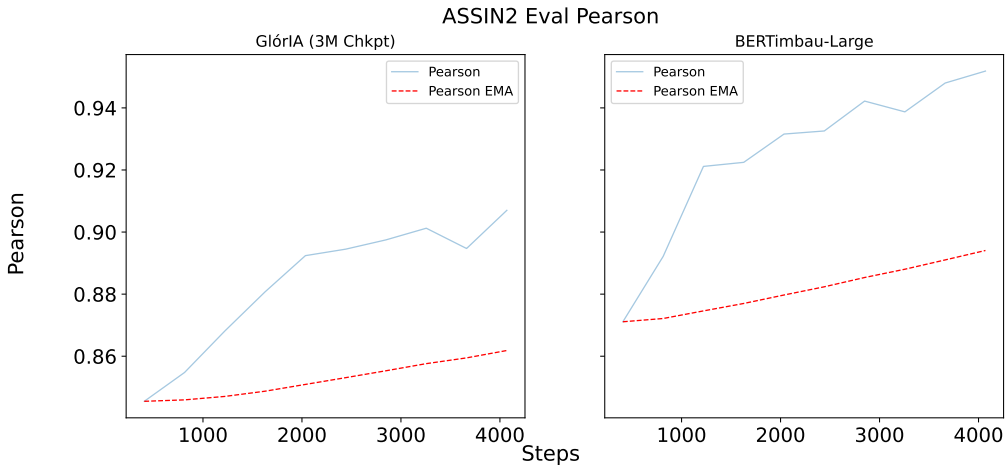

Figure 6.4: Comparison between GlórIA's and BERTimbau's evaluation of Pearson score during training (dev set). Obtained from the runs where their F1 and Pearson values peaked amongst the entire experiments. Using hyperparameter set B0 for both models and using the 3M checkpoint of our model. The red line is the exponentiated moving average with a smoothing factor of 0.95.

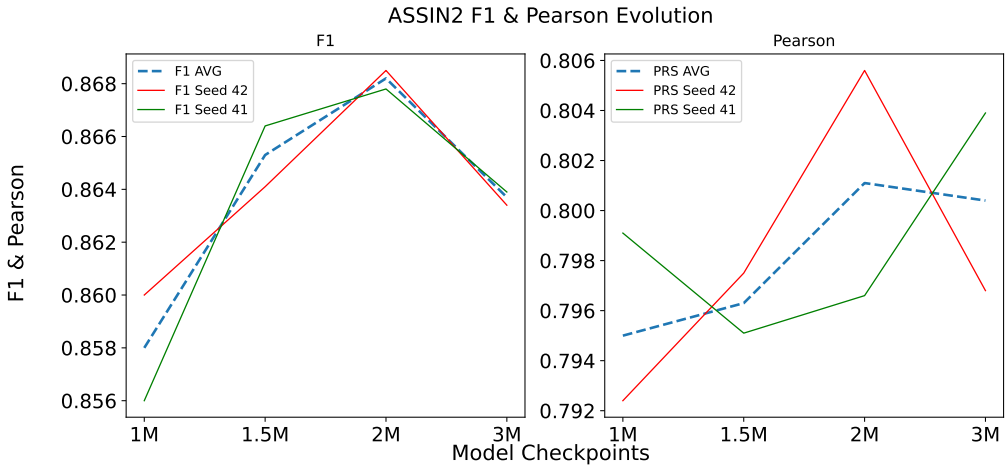

Figure 6.5: Evolution of the F1 and Pearson scores for the ASSIN2 task along our model's checkpoints (X-axis). The blue lines are the averages of the scores for the different seeds.

6.3 GLUE-PTPT Tasks

The General Language Understanding Evaluation (GLUE - Wang et al. [28]) is one of the standard benchmarks when training and evaluating a language model. GLUE is a collection of multiple language tasks with similar formats, like semantic similarity, and entailment, among others.

There currently exists a Brazilian Portuguese version and a European Portuguese version, respectively PLUE (GOMES [60]) and GLUE-PTPT (Rodrigues et al. [12]). Both of these versions are translated from the original. However, due to the required efforts in translating, reviewing, and ensuring data quality, only 4 tasks out of 11 were machine-translated - PLUE and GLUE-PTPT share those same 4 tasks:

- **RTE**: this task is, once again, a textual entailment task, following the same format and idea as ASSIN-2's RTE task. Refer to Table 6.4 for examples.
- **WNLI**: The *Winograd NLI* is based on inference and coreferences between sentences. Refer to Table 6.5 for examples.
- **MRPC**: The *Microsoft Research Paraphrase Corpus* is a similarity task with the goal of detecting if two sentences are paraphrases of one another. Refer to Table 6.6 for examples.
- **STS-B**: this final task is simply a textual similarity task like the similarity task in ASSIN-2. Refer to Table 6.7 for examples.

Due to this thesis's focus on the European branch of the Portuguese language, **we focused our evaluation on the GLUE-PTPT benchmark**, instead of its Brazilian Portuguese counterpart (PLUE).

Table 6.4: Examples of pairs of sentences from the **RTE** task.

Sentence 1	**Sentence 2**	**Label**
"Os preços do petróleo diminuem à medida que a ameaça do petróleo Yukos aumenta"	"Os preços do petróleo sobem."	1 (not entailment)
"As autoridades disseram que o suspeito finlandês estava entre os mortos mas não forneceu um motivo para o ataque."	"As autoridades disseram que o suspeito, um cidadão finlandês, estava entre os mortos."	0 (entailment)

6.3.1 *GLUEing it all together*

The model was finetuned for each task separately, using different hyperparameters for each one according to our experiments to find which ones were optimal or not - since

Table 6.5: Examples of pairs of sentences from the **WNLI** task.

Sentence 1	**Sentence 2**	**Label**
"Enfiei um alfinete através de uma cenoura. Quando puxei o pino para fora, tinha um buraco."	"A cenoura tinha um buraco."	1 (entailment)
"John estava a correr pelo parque quando viu um homem a fazer malabarismos com melancias. Ele era muito impressionante."	"John foi muito impressionante."	0 (not entailment)

Table 6.6: Examples of pairs of sentences from the **MRPC** task.

Sentence 1	**Sentence 2**	**Label**
"O DVD-CCA recorreu então ao Supremo Tribunal do Estado."	"O DVD CCA recorreu dessa decisão para o Supremo Tribunal dos Estados Unidos."	1 (equivalent)
"Os ganhos por acção de operações recorrentes serão de 13 cêntimos a 14 cêntimos."	"Isso bateu a previsão de lucros da empresa em Abril de 8 a 9 cêntimos por acção."	0 (not equivalent)

Table 6.7: Examples of pairs of sentences from the **STS-B** task.

Sentence 1	**Sentence 2**	**Label (Similarity)**
"Um avião está a descolar."	"Um avião aéreo está a descolar."	5
"Um homem está a fumar."	"Um homem está a patinar."	0.5
"Um homem está a tocar guitarra."	""Uma rapariga está a tocar guitarra."	2.8

hyperparameters used for the RTE task may not be the most adequate ones for the MRPC task. The model was trained by attaching a linear layer on top - using the default implementation of HuggingFace's *GPTNeoForSequenceClassification*. Once again, the models we will be comparing against were also finetuned following this same protocol to ensure comparability.

In sum, our 4 GLUE PT-PT tasks are trained following the same logic as the ASSIN2 tasks as explained in Sub-Section 6.2.1, except we're not multi-task training, since every sub-task in this benchmark is a single task.

The input is prepared similarly to the ASSIN2 task, except here we only have to pass one label per sub-task. Both sentences are tokenized as a pair, with the corresponding label, and a maximum sequence length of 128 is used due to the nature of relatively short sentences present in this benchmark.

6.3.2 Important Detail on Test Data

A critical notice is that models trained on GLUE tasks are **officially** evaluated via its official website[1]. Unfortunately, during the time this thesis was developed, especially during the evaluation process, GLUE's official online testing service was not available. Due to this, for each of our four subtasks, we had to take 10% from the training splits and use them as test splits - maintaining the dev splits. This is similar to the protocol followed by Rodrigues et al. [12]. Due to a misunderstanding regarding their protocol, our splits are different. We used the dev/validation splits as dev/validations splits, and took 10% of the samples from the train splits to create the test splits.

This leads to it not being directly comparable to English models, as well as models that performed a similar data split - since our test split has no guarantees that it is similar to the one that Albertina created and their test split has not been publicly made available to the extent of our knowledge. **However, all of our models and finetunes were trained and evaluated using the same data, so they remain comparable amongst themselves**.

6.3.3 Defining Experiments for Subtasks

Hyperparameter Set	BS	LR	Warmup Steps	Epochs	GA Steps	Scheduler
A1	32	1e-4	0	5	2	linear
B1	32	1e-5	0	5	2	linear
C1	32	1e-4	0	5	2	constant
D1	32	1e-5	0	5	2	constant

Table 6.8: Hyperparameter sets for **RTE**, **MRPC** and **WNLI** sub-tasks. All of these would be used to fine-tune each and every comparison model (including our own).

Hyperparameter Set	BS	LR	Warmup Steps	Epochs	GA Steps	Scheduler
A2	32	1e-4	0	5	2	constant
B2	32	1e-5	0	5	2	constant
C2	32	1e-6	0	5	2	constant
D2	32	1e-4	0	5	2	linear

Table 6.9: Hyperparameter sets for **STS-B** sub-task. All of these would be used to fine-tune each and every comparison model (including our own).

To obtain our results we followed the same process as used for the ASSIN2 tasks. When a GLUE task had 2 main metrics, a checkpoint for the best value seen in the evaluation loop (per metric) would be generated during the fine-tuning. When a GLUE task only used a singular main metric, we simply applied the evaluation script and used the results *as is* - since there wasn't any need to average them with any other checkpoint (since there would only be one checkpoint per best metric).

[1] https://gluebenchmark.com/

The **RTE** and **WNLI** tasks were evaluated using Accuracy. The **STS-B** task was evaluated using Pearsons. The **MRPC** task was evaluated using both Accuracy and the F1-score.

Just like we did for ASSIN-2, an experiment space was defined for each sub-task of the GLUE-PTPT benchmark. For the **RTE**, **MRPC**, and **WNLI** tasks we tested for learning rates *1e-4* and *1e-5*, and constant and linear LR schedulers - building hyperparameter sets that we found to be optimal during testing since some combinations would lead to unstable training across models. For the **STS-B** task we experimented with learning rates *1e-4*, *1e-5* and *1e-6*, and constant and linear schedulers. Three (3) training sessions were performed for all models and hyperparameter sets, where we tested for seeds 41, 42, and 43 (1 session per training seed).

We aimed at having 4 different hyperparameter sets used for fine-tuning and evaluation, shared across the models for comparability (meaning models were fine-tuned using the same hyperparameter sets), for each sub-task. Consult these hyperparameter sets used for experimentation in Table 6.8 and Table 6.9 for, respectively the set of subtasks RTE, MRPC, and WNLI, and STS-B (single). A weight decay value of 0.01 was used in every experiment.

We chose to compare our model against Albertina-PTPT, BERTimbau-Large, and Gervasio-PTPT due to the European Portuguese nature of this benchmark (although it is machine translated) - meaning we used the respective pre-trained weights for these models made publicly available by their researchers on HuggingFace. We chose checkpoints 1M, 1.5M, 2M, and 3M from our model for evaluation.

6.3.4 GLUE-PTPT Results

	Models	**RTE**	**MRPC**		**WNLI**	**STS-B**
	Models	**Accuracy**	**F1**	**Accuracy**	**Accuracy**	**Pearson**
Dec.	GlórIA	0,6907	0,8702	0,8131	**0,5455**	0,8530
	Gervasio-PTPT	0,6441	0,8551	0,7929	0,5455	0,8240
Enc.	Albertina-PTPT	**0,8136**	**0,9126**	**0,8864**	0,5455	**0,9040**
	BERTimbau-Large	0,6610	0,8779	0,8384	0,5455	0,8910

Table 6.10: These are the highest recorded values for the evaluation metrics for the GLUE-PTPT tasks across all experiments (all fine-tunes). *Enc.* stands for Encoders, and *Dec.* stands for Decoders.

As done for the ASSIN-2 tasks, we gathered the highest results achieved for each metric and for each model after evaluating them on the test set and compared them in Table 6.10. Each model was evaluated using our own "offline" test set for their respective subtasks, as noted previously in Section 6.3.2.

Albertina takes the spot for the highest performance in general, while our model competes with BERTimbau in every subtask, surpassing it in GLUE-PTPT's RTE subtask.

It becomes apparent that the two main distinguishing types of architectures (encoder-based and decoder-based) become distinct in terms of performance (as we expected) according to our results - meaning that, despite our model having a larger size, the way an encoder is pre-trained and its architecture can still be more apt to perform such tasks. Nevertheless, our results demonstrate that a large decoder-based LM could still be a viable option for this task. **Putting the encoders apart, we compare the decoder-based / generative models amongst themselves**. By doing so, we **compared GlórIA against Gervasio-PTPT in all tasks, outperforming Gervasio-PTPT by relevant margins. Once again, this goes as expected, another proof that supports our model's larger size and increased training data size and diversity.**

Curiously, every model ties in performance in the WNLI subtask. We suspect that this is due to the nature of the data itself and its very small size - 5 epochs of WNLI training translate to 180 steps only - making it easier for models to "bottleneck" or saturate due to possible lack of samples.

The following are some observations regarding the optimal hyperparameter sets for each model and each subtask, as observed through our experiments. These are presented in Annex V with their respective scores. This task has our largest experiment space.

- **RTE Observations**: For GlórIA, the peak value was recorded when using hyperparameter set B1 from the 1M checkpoint, but as we evaluated the model further, the most optimal hyperparameter set would change. For BERTimbau, we observed hyperparameter sets B1 and D1 as the most optimal ones, having obtained the highest value in accuracy through set B1. Albertina shares the same optimal sets as BERTimbau. For Gervasio, the same is observed with hyperparameters B1 and D1 yielding the overall best results for this model.

- **MRPC Observations**: Curiously noted when observing the experiments' results, is that our model, Albertina, and BERTimbau all took more advantage of hyperparameter set D1, with B1 strongly trailing behind.

- **WNLI Observations**: Despite the "notorious unanimity" in performance results observed in this subtask, hyperparameter sets A1 and C1 are highlighted as the most adequate sets across all the evaluated models, due to higher stability in final performance.

- **STS-B Observations**: Both our model and BERTimbau took better advantage of set B2 when compared to the other sets. Parallel to this, Albertina suffered greatly with sets A2 and D2, having Pearson values close to 0 (probably thanks to a non-optimal learning rate), but thrived using set B2.

6.3.4.1 Relevant Notes on GlUEPT's Convergence

We chose the best runs of each model from every subtask, to present a graphical comparison between these models for the subtasks' metrics. Since we used multiple checkpoints from our model, different tasks will have different checkpoints, since we picked the run (amongst all our checkpoints) with the best final results. Refer to Annex VI for each subtask's graphs and respective comparisons.

As we look through each model's and each subtask's graphs, the first thing to notice is the similarity in behavior between models of the same or similar architecture. Models like Albertina-PTPT and BERTimbau-Large demonstrate, in some cases, very similar convergence. Gervasio-PTPT and GlórIA 1.3B also share similar behavior between them - further pushing the distinction between encoder and decoder models.

Evaluating different checkpoints of our model allowed us to, once again, analyze if its performance would evolve with further pre-training. Just like it was done for the ASSIN-2 task, we averaged the results of all the experiments from all the seeds (for each checkpoint), and mapped them visually. The Figures 6.6, 6.7, 6.8 and 6.9 (respectively RTE, MRPC, WNLI and STS-B), demonstrate that further pre-training GlórIA leads to few or no gains in performance.

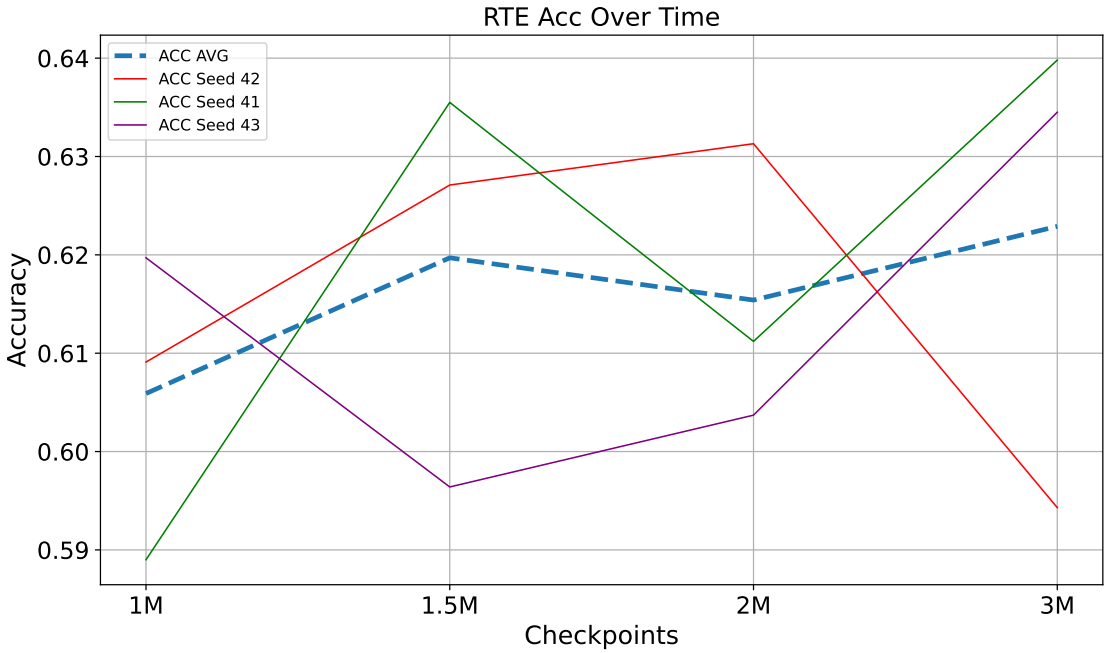

Figure 6.6: Performance evolution of our model for the RTE subtask's Accuracy score.

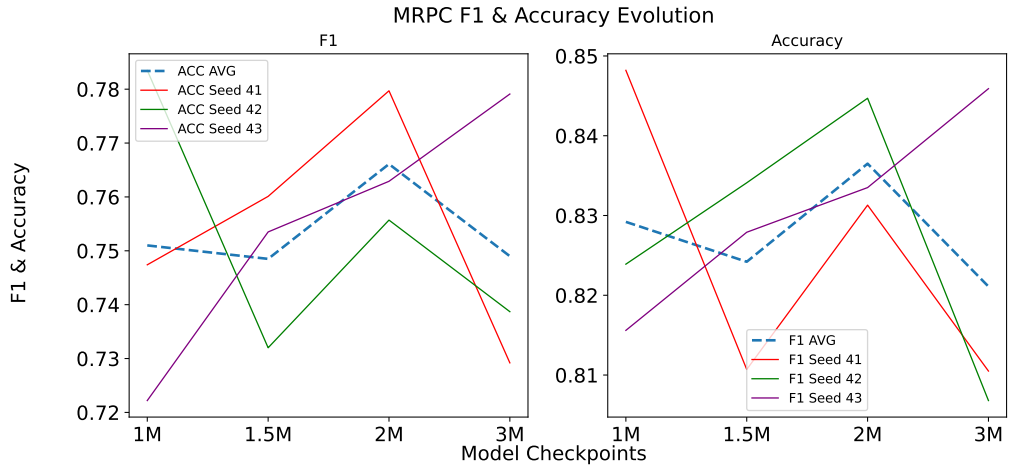

Figure 6.7: Performance evolution of our model for the MRPC subtask's F1 and Accuracy scores..

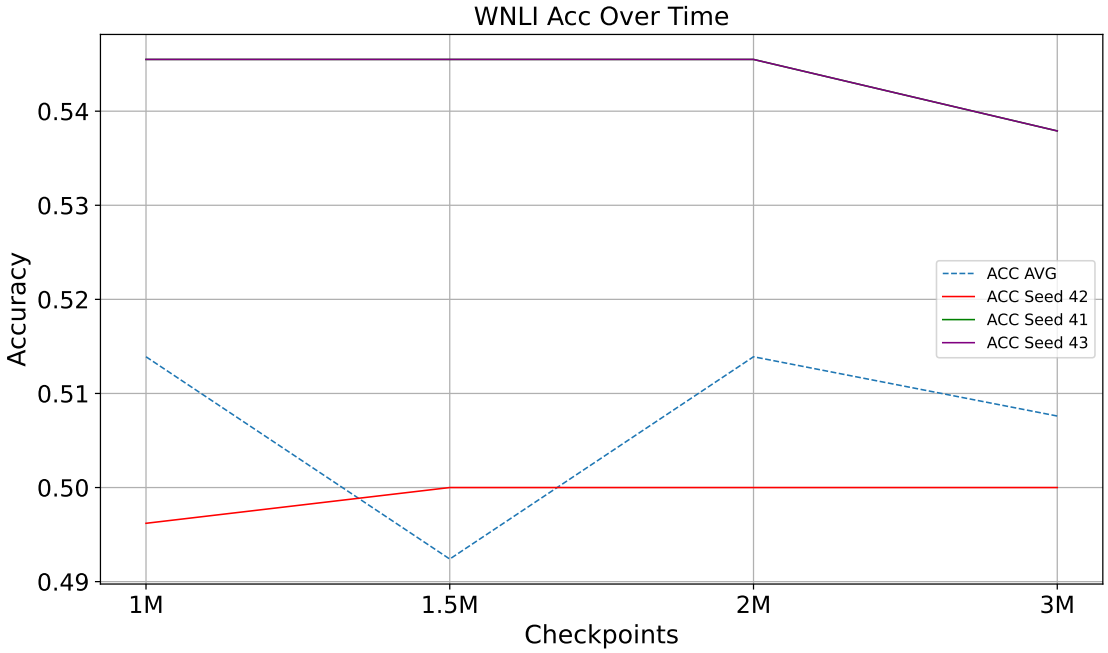

Figure 6.8: Performance evolution of our model for the WNLI subtask's Accuracy score.

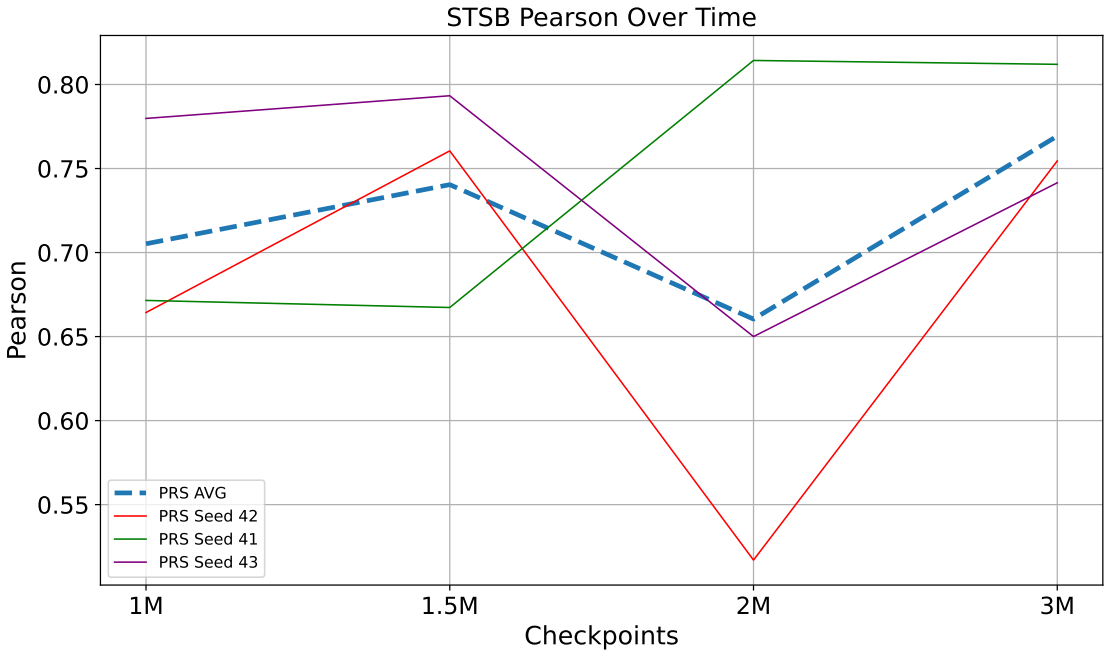

Figure 6.9: Performance evolution of our model for the STSB subtask's Pearson score.

6.4 SQUADPT Task

The *Standford Question Answering Dataset* is another standard benchmark in evaluating LLMs. Each entry is composed of a context, a question, and an answer. The answers are passages or spans of the context - meaning that the answer is extracted from the context. Dataset entries also contain the start and end offset for each answer, so we know where the answer starts and the answer ends.

We introduce, to the extent of our knowledge, one of the first, if not the first, evaluation of Portuguese-based language models on the Portuguese machine-translated version of SQUADv1.1 - named SquadPT[2].

6.4.1 Fine-tuning on SQUADPT

The training goal is to have the model predict a span of start positions and a span of end positions. To do this, we once again add a linear layer on top of the model's hidden states - using the default implementation of HuggingFace's *GPTNeoForQuestionAnswering* model. Just like the previous tasks, the models we will be comparing against were also finetuned following this same protocol to ensure comparability.

The evaluation was performed via the F1 Score and Exact Match metrics, calculated using HuggingFace's Evaluate[3] library.

[2]https://forum.ailab.unb.br/t/datasets-em-portugues/251
[3]https://huggingface.co/docs/evaluate/index

6.4.2 Important Detail on Test Data

Just like GLUE, SQUAD's official evaluation is performed via online submission. However, since our translated version of SQUAD is version 1.1, the online submission was unavailable due to SQUADv2 being the current official benchmark, to the extent of our knowledge. Even if it was available, a JSON with the predicted answers would have to be submitted, which would be compared against their test set - **which is in English** - so Portuguese official evaluation would not be possible.

For this purpose, we followed the same protocol as we did with GLUE-PTPT, and split the original train set into a new train and offline test set [90% + 10%]. We ended up with 78,759 training samples, 8751 test samples, and 10,570 dev samples (for in-training evaluation).

6.4.3 Pre & Post-processing of Samples

To understand the objective of SQUAD's pre-processing, we need to first look into a document's format. Refer to Example 6.1 for an example of a document retrieved from the original SQUADv1.1 - the format stays the same in the Portuguese version.

Listing 6.1: SQUAD document example.

```
{'id': '5733be284776f41900661182',
 'title': 'University_of_Notre_Dame',
 'context': 'Architecturally, the school has a Catholic
    character. Atop the Main Building\'s gold dome is a golden
    statue of the Virgin Mary. Immediately in front of the Main
    Building and facing it, is a copper statue of Christ with
    arms upraised with the legend "Venite Ad Me Omnes". Next to
    the Main Building is the Basilica of the Sacred Heart.
    Immediately behind the basilica is the Grotto, a Marian
    place of prayer and reflection. It is a replica of the
    grotto at Lourdes, France where the Virgin Mary reputedly
    appeared to Saint Bernadette Soubirous in 1858. At the end
    of the main drive (and in a direct line that connects
    through 3 statues and the Gold Dome), is a simple, modern
    stone statue of Mary.',
 'question': 'To whom did the Virgin Mary allegedly appear in
    1858 in Lourdes France?',
 'answers': {'text': ['Saint Bernadette Soubirous'], '
    answer_start': [515]}}
```

Each document is composed of a context, a question, and possible answers. The question relates to the context, and the answer is a span of text that exists inside the context. Each answer is also accompanied by its start position (in characters).

Since models work with token IDs, we need to process the training data so that it makes sense for the model. Our goal is to feed the model the context and the question as inputs, and the start and end positions (of the answer) as labels. The start and end positions are calculated during the pre-processing phase of the *training data*.

Using the HuggingFace's Tokenizer's library capabilities, for a given document we can obtain a list of the offsets of each token - a list of tuples *(start character, end character)* for each token in our context. Using the offsets and the answer start that is presented in a document, we can find the starting token through its offsets, and its end token by adding the length of the answer (in characters) to the starting characters.

However, sometimes the context is too large to fit in our model due to the sequence length. To solve this, we use a *stride*. Since we tokenize our input in the format of *Question+Context*, our tokenizer will use our arbitrary stride window to truncate the context and generate a new *feature* - maintaining the answer as is but using the truncated part of the context with *n-stride* of overlapping tokens.

Listing 6.2: Example of features that were generated from one document using a max sequence length of 100 and a stride of 50

```
1 '[Q] To whom did the Virgin Mary allegedly appear in 1858 in
    Lourdes France? [C] Architecturally, the school has a
    Catholic character. Atop the Main Building\'s gold dome is a
    golden statue of the Virgin Mary. Immediately in front of the
     Main Building and facing it, is a copper statue of Christ
    with arms upraised with the legend " Venite Ad Me Omnes ".
    Next to the Main Building is the Basilica of the Sacred Heart
    . Immediately behind the basi'
2 '[Q] To whom did the Virgin Mary allegedly appear in 1858 in
    Lourdes France? [C] the Main Building and facing it, is a
    copper statue of Christ with arms upraised with the legend "
    Venite Ad Me Omnes ". Next to the Main Building is the
    Basilica of the Sacred Heart. Immediately behind the basilica
     is the Grotto, a Marian place of prayer and reflection. It
    is a replica of the grotto at Lourdes, France where the
    Virgin '
3 '[Q] To whom did the Virgin Mary allegedly appear in 1858 in
    Lourdes France? [C] Next to the Main Building is the Basilica
     of the Sacred Heart. Immediately behind the basilica is the
    Grotto, a Marian place of prayer and reflection. It is a
    replica of the grotto at Lourdes, France where the Virgin
    Mary reputedly appeared to Saint Bernadette Soubirous in
    1858. At the end of the main drive ( and in a direct line
    that connects through 3'
```

```
4 '[Q] To whom did the Virgin Mary allegedly appear in 1858 in
    Lourdes France? [C]. It is a replica of the grotto at Lourdes
    , France where the Virgin Mary reputedly appeared to Saint
    Bernadette Soubirous in 1858. At the end of the main drive (
    and in a direct line that connects through 3 statues and the
    Gold Dome ), is a simple, modern stone statue of Mary.]'
```

Essentially, one document can generate one or more *features* that we feed our model. If an answer is not present in one of the features, we set the start and end positions as (0,0). Observe listing 6.2 for an example of this - the *CLS* and *SEP* tokens are only included to make it easier to view how the question and context parts are organized.

In sum, if a SQUAD-like dataset has, for example, 80k documents, when pre-processed to be fed into a model they can end up turning into >=80k features.

For validation, this pre-processing becomes simplified, since we don't need the start and end positions of the tokens that contain the answer. We only need to map our generated features to the original documents - and this can be done via their document IDs.

Once this pre-processing is done, we need to handle the post-processing so that we can compute metrics correctly and evaluate our model's predictions. For this post-processing, we've already fed the model's inputs and obtained the start and end prediction logits. To deal with the post-processing, we start by creating a *examples to features* map, where for each document/example, we have a list of the generated features. Then, we iterate over all the examples and, using our *examples to features* map, we loop through each feature. For each feature, we then gather the corresponding start and end logits, and we skip answers with length <= 0 or >= 30 (arbitrary value), as well as answers that are not fully inside the context. After we gathered all the possible answers from all the features of a single example, we picked the one with the best score - the sum of the start logit score and the end logit score.

Having all the predicted answers for each document, we then compute the metrics using HuggingFace's Evaluate library, by sending the predicted answers and a list of the expected (*ground truth*) answers.

6.4.4 Experimental Space

Hyperparameter Set	BS	LR	Warmup Steps	Epochs	GA Steps	Scheduler
A3	32	1e-4	0	3	8	linear
B3	32	1e-5	0	3	8	linear
C3	32	1e-6	0	3	8	linear

Table 6.11: Hyperparameter sets for the SQUAD task. All of these would be used to fine-tune each and every comparison model (including our own).

For the third time, the results calculation was performed just like in the previous tasks. In specific, for SQUAD we'd have a checkpoint for the best F1 score and a checkpoint for the best Exact-Match score. Both would be evaluated, and the F1 and Exact-Match results would be averaged to obtain the final result for a specific simulation/experiment.

$$F1 = (\textit{checkpoint-best-f1's}F1 + \textit{checkpoint-best-EM's}F1)/2$$

$$EM = (\textit{checkpoint-best-f1's}EM + \textit{checkpoint-best-EM's}EM)/2$$

For SQUADPT, we trained for 3 epochs, using AdamW as an optimizer with a linear scheduler. Following the previous tasks' methodology for hyperparameter sets and experimental spaces, we performed three training/finetuning sessions (for seeds 41-43), and test learning rates *1e-4*, *1e-5*, and *1e-6*. Check the hyperparameter sets for this task in Table 6.11. At the end of each epoch, we would perform an evaluation using the dev/validation set, and posteriorly evaluate the best checkpoints on the test set. A weight decay value of 0.01 was used in every experiment.

We chose to compare our model against Albertina-PTPT, Gervasio-PTPT, and BERTimbau-Large, contemplating all these models - using our 1M, 2M, and 3M steps checkpoints. We decided to exclude checkpoint 1.5M from this evaluation due to logistic reasons: namely the increasingly larger number of total experiments in this thesis, the time needed to perform one simulation for this specific task, which would be between 4 to 5 hours, and resource availability.

6.4.5 SQUADPT Results

Just like we've done for the previous tasks, after defining our fine-tuning protocols and experimental space, we follow them with their results and respective analyses.

6.4.5.1 SQUADPT Metrics Results

Once again Albertina-PTPT demonstrates a higher performance. Meanwhile, GlórIA is able to achieve decent and better scores when compared to Gervasio-PTPT - **placing our model at the top of the two decoder-based models**. Refer to Table 6.12 for the peak scores achieved for this task (with the test set). We also publish the performed experiments and respective results in Annex VII.

	Models	F1	Exact Match
Dec.	GlórIA	0,6688	0,3383
	Gervasio-PTPT	0,6329	0,3168
Enc.	Albertina-PTPT	**0,7648**	0,3816
	BERTimbau-Large	0,7551	**0,3845**

Table 6.12: These are the highest recorded values for the evaluation metrics (test set) for the SQUAD-PT tasks across all experiments (all fine-tunes).

6.4.5.2 SQUADPT Convergence Notes

There are a few notes to be made on the models' behavior on the SQUADPT task. The B3 hyperparameter set appears to be the most optimal set of the experiment space, from which we obtained all the peak scores from all models. The A3 hyperparameter set appeared to be non-optimal when fine-tuning Gervasio-PTPT, leading to very low F1 and Exact-Match scores, close to zero - the learning rate was inadequate.

Each model demonstrates similar behavior when compared to their same-architecture counterpart, for example, Albertina-PTPT and BERTimbau have a similar loss convergence (both encoders), and the same goes for GlórIA and Gervasio-PTPT - verified through Figures 6.10.

Regarding the F1 and EM scores, all models show a tendency to increase during evaluation, but only with 3 epochs - meaning only 3 in-training evaluations - it is difficult to infer their evolution in the long run without performing further epochs / during training evaluations. We present comparisons for the evaluation scores from each model's best runs for the loss, F1, and Exact Match scores in Figures 6.10, 6.11, and 6.12.

Once again, we analyzed the evolution of GlórIA's scores in this task, by tracking the scores of the different experiments. For each checkpoint (1M, 2M, and 3M), we averaged the results of every experiment and used them to try to see if the model's performance would increase with more pre-training. This evolution is portrayed in Figure 6.13. A relevant jump in performance from 1M to 2M checkpoint is observed in both metrics, showing a posterior slight tendency to increase from checkpoint 2M forwards, making us believe performance can still increase but could very soon stagnate.

6.5 Final Analysis on Supervised Evaluation

As previously pointed out, in GLUE-PTPT's and SQUADPT's tasks, while the encoder-based architecture achieves the best scores, our decoder-based generative model can come close in some tasks - **demonstrating potential that it can still be a viable option for some of these downstream tasks**.

Once again, we believe this reflects the type of information assimilated during pre-training, mainly related to the nature of each architecture's pre-training goals. For example, we believe that when pre-training BERTimbau or Albertina-PTPT, due to the bi-directionality of the encoder and the usage of a *CLS* token that stores extra information, the models become more capable of looking at sentences as a whole which helps them in tasks where you need to extract information from a whole sentence, like it is done in the SQUADPT task. Albertina-PTPT, in special, has better performance than BERTimbau mainly due to its larger size, increased training, and data diversity - differences that GlórIA can take advantage of across all tasks to achieve good and satisfying performance, even if it's not the top scorer.

Leaving the 'inter-architectural' comparison, our GlórIA 1.3B generative model prevails

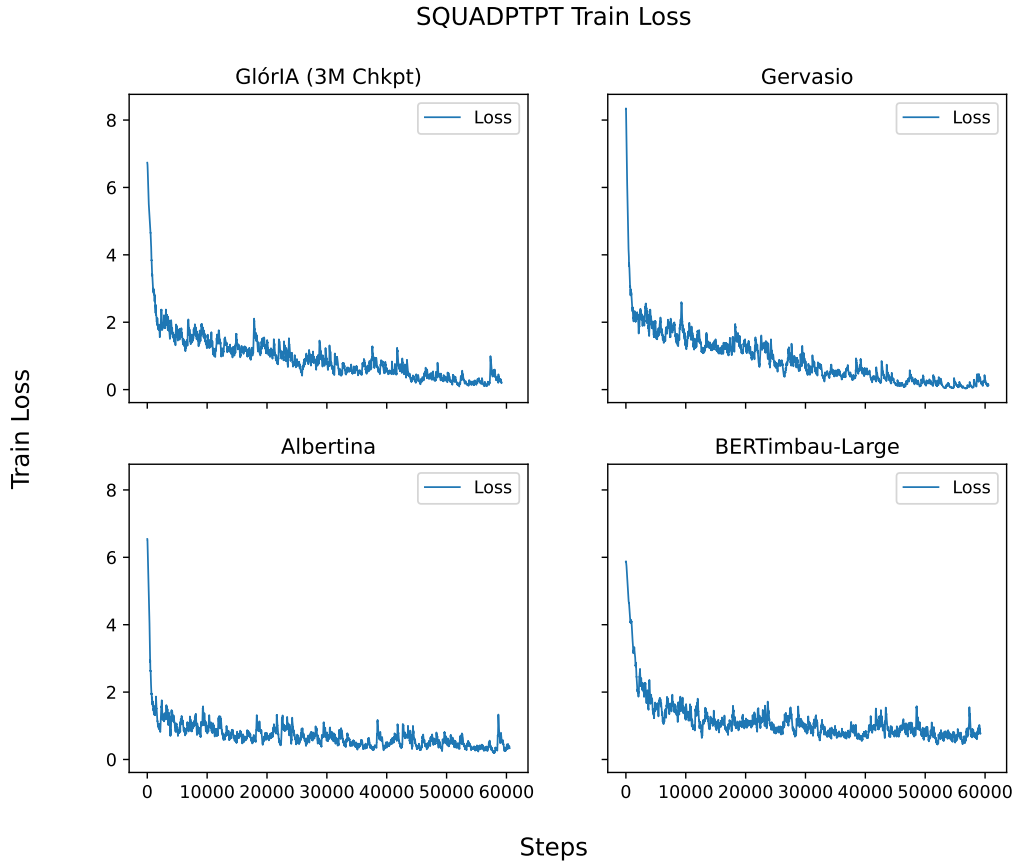

Figure 6.10: During training SQUADPT loss comparison between GlórIA, BERTimbau-Large, Albertina-PTPT, and Gervasio-PTPT - using their best runs (runs where we obtained the peak scores), with hyperparameter set B3. All graphs use the exponentiated moving average with a smoothing factor of 0.95.

over the 1B Gervasio-PTPT, by a relevant margin. **We credit this difference in performance to the size, but especially to the increased data diversity our model sees.**

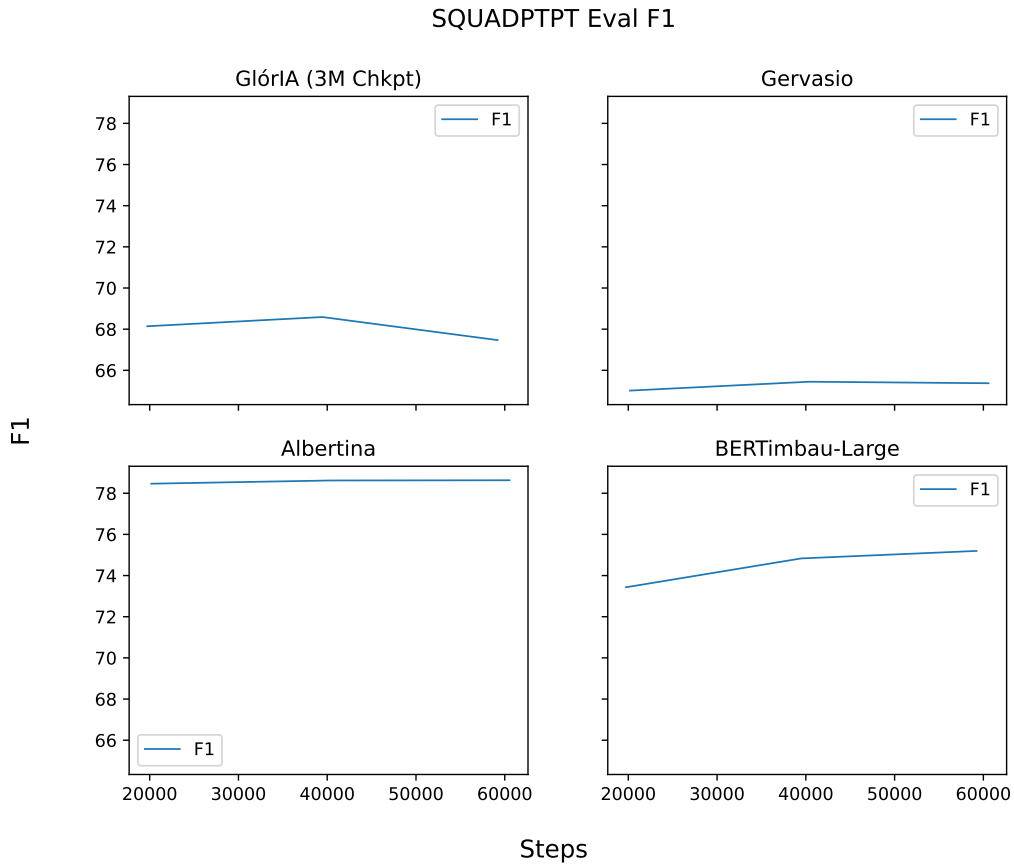

Figure 6.11: Comparison of the evaluation F1 score (validation/dev set) from all models - again using their best runs, with hyperparameter set B3.

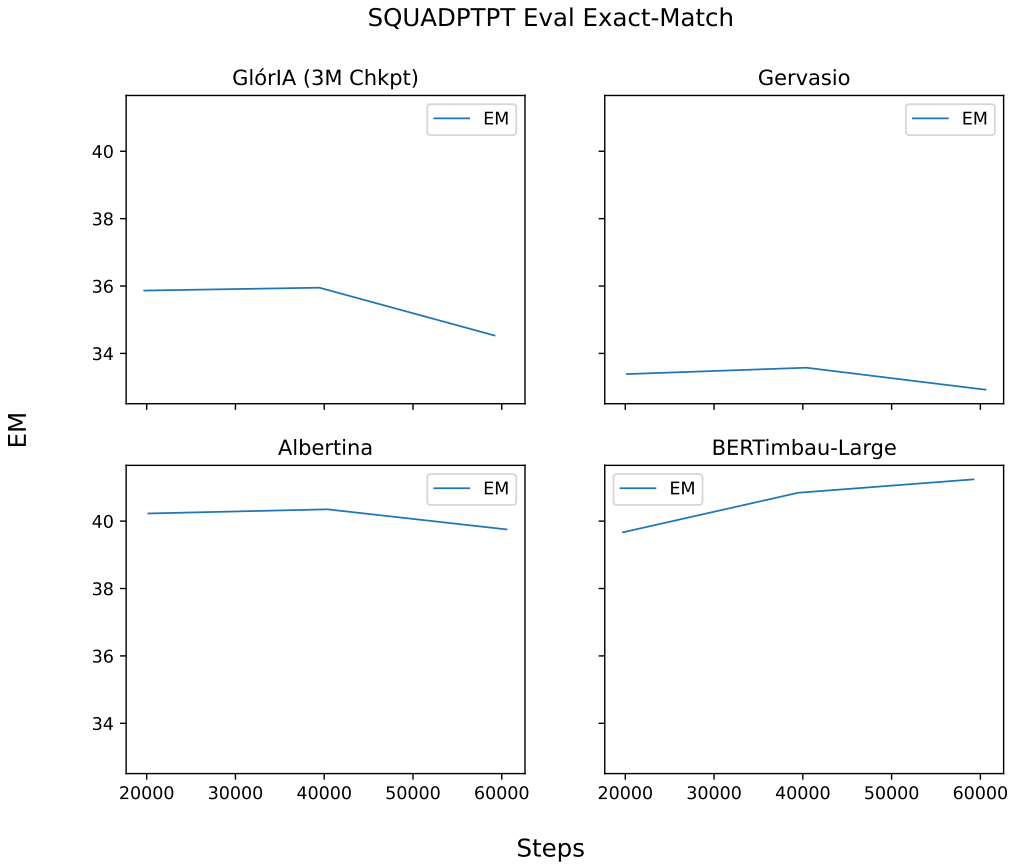

Figure 6.12: Comparison of the evaluation Exact-Match score (validation/dev set) from all models - again using their best runs, with hyperparameter set B3.

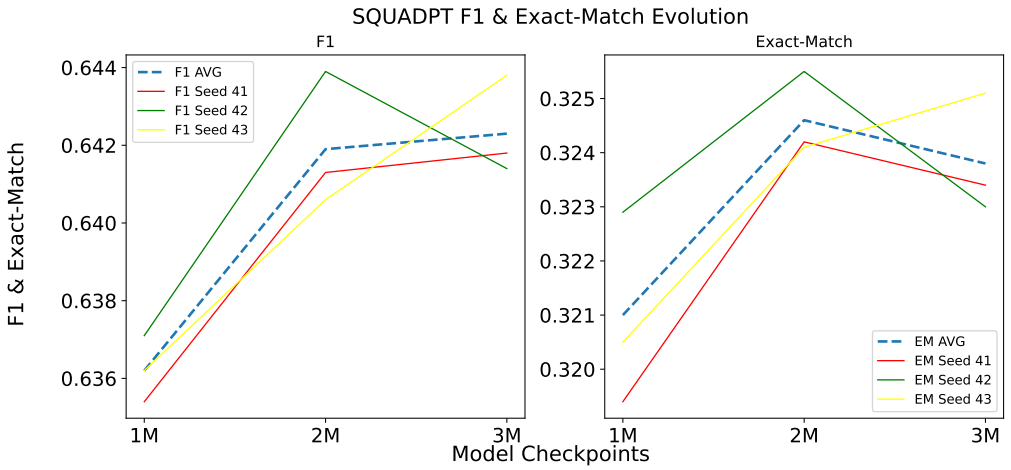

Figure 6.13: Evolution of the F1 and Exact-Match scores when evaluating GlórIA's 1M, 2M, and 3M steps checkpoints (test set). The blue dashed line represents the average of all the experiments for all the checkpoints and all the seeds.

7

Final Conclusions

7.1 Overview

For this thesis, we investigated and studied which parameters, methodologies, and data can be used to train a very large generative language model for the European Portuguese language. Our final goal was achieved, which was to **produce a decoder-based language model capable of understanding and generating coherent (and grammatically correct) PT-PT text**, using our findings - **giving birth to GlórIA**.

Initial research was performed, by assessing how recent Portuguese models were trained, and evaluated, and what data they used - leading us to our first problem: a lack of European Portuguese resources. We began by searching and gathering "raw" text data sources from where we could extract or use European Portuguese text. From this search, we prepared the **largest European Portuguese corpora with 35B tokens (at the time of writing) by processing and filtering several sources of data**.

With this data, as well as the methodology defined, we proceeded to pre-train our model and monitor its behavior. Posteriorly, GlórIA was subjected to a thorough evaluation across multiple and diverse downstream tasks. First, we **produced a novel benchmark to evaluate the generative capabilities of Portuguese decoder-based models, CALAME-PT**, which we used to evaluate our model's performance and compare it against Gervasio-PTPT[1], developed by the same research group that created Albertina-PTPT (Rodrigues et al. [12]). These initial results were very good **as we managed to achieve top performance, and demonstrated through this task and sample generations that the model can generate adequate and contextual texts in PT-PT**.

Secondly, we **evaluated our model on** the ASSIN-2, GLUE-PTPT, and SQUADPT benchmarks that are composed of some **discriminative downstream tasks**: sentence entailment, sentence similarity, extractive question-answering, etc. For this part of the evaluation, we increased the model pool, introducing encoder-based, non-generative Portuguese language models: Albertina-PTPT and BERTimbau-Large. Our findings showed that the encoder-based models tend to achieve better performance in these types

[1] https://huggingface.co/PORTULAN/gervasio-ptpt-base

of tasks. Nevertheless, GlórIA still had the highest scores when compared to its other decoder counterpart (Gervasio-PTPT), even showing potential to increase its performance by increasing its size. **This model was integrated into the ArquiWIZ.pt project**, and a **paper with our findings and results has been accepted for publication at PROPOR 2024**[2] to make our model and benchmark publicly available.

Finally, **we believe that GlórIA, our pre-training corpora, and our new benchmark bestows a new and solid base for researching new Portuguese NLP tools and language models that support them**. We hope that our findings can help address challenges that may appear when working with European Portuguese language models, motivating future researchers who want to improve such tools and models for this language.

7.2 Reflections on PT-PT Resources

The biggest challenge and difficulty that regularly emerged throughout this thesis was the quality and lack of European Portuguese data. Thus, our first course of action was to gather, at the time of writing, the largest and most diverse PT-PT text corpus we could gather for pre-training - with 35B tokens. It is also important to notice that there is still room to improve the pre-processing and it is still behind English state-of-the-art standards, whose pre-training datasets have reached the trillions of tokens mark.

Even with the existing European Portuguese benchmarks (GLUEPTPT only), evaluating PT-PT language models still has much more potential to evolve. PT-PT evaluation needs to be made more mature and comprehensive. This is due to the machine-translated nature of current benchmarks, which we empirically assume to be of a different quality than a natively written dataset. It also lacks richness or induces some language bias that is different or doesn't exist when looking at natively written text. Essentially, we hypothesize that some important information (semantic, grammatical, etc.) is lost when comparing machine-translated text to natively written text, especially in the Portuguese language due to its larger variety and flexibility when compared to the English language.

Besides this, there wasn't, at the time of writing, a benchmark to evaluate a PT-PT model's ability to generate text. These issues **pushed us to produce our own benchmark, CALAME-PT**, composed of generated and handwritten samples: a benchmark to evaluate Portuguese language modeling capabilities.

When it came to the discriminative tasks, the goals of the existing benchmarks (GLUEPTPT[12], ASSIN2[74], SQUADPT[3]) also limit the quality of our evaluation, specifically when training a generative language model, since none of these tasks were designed to be based on text generation. One can empirically assume that these tasks are more adequate to evaluate encoder-based models due to their bi-directionality - and this is shown throughout our evaluation results. Be that as it may, we also concluded that our GlórIA has some potential to be a feasible option for those downstream tasks.

[2] https://propor2024.citius.gal
[3] https://forum.ailab.unb.br/t/datasets-em-portugues/251

Even if more recent Brazilian Portuguese benchmarks come to light, the difference in the characteristics between both variants of this language can be enough to make a difference - the difference between PT-PT and PT-BR exists and should not be ignored when training a Portuguese language model exclusively on one of these variants.

The results obtained propel us to urge researchers to give further attention to the native creation and curation of Portuguese benchmarks.

7.3 Future Work

Building upon the remarks related to the main challenges and the work done in this thesis, there are a few important related research projects one could take up for future work.

7.3.1 Text Generation-focused Benchmark - CALAME-PT

The first future work based on this thesis would be focused on producing and curating European Portuguese benchmarks from scratch, ensuring they are natively written and focused on text generation tasks. In specific, it would involve employing the help of professional linguistics in continuing the work done on CALAME-PT.

This benchmark would further enrich the availability of the tasks one can choose from to evaluate their model, as well as be the first one of its kind - opening a way to correctly, and accurately, evaluate the quality of the text that it's generated by a Portuguese decoder-based model.

To summarize, part of this future work would be based on improving this benchmark by increasing its size, improving the quality of the samples, identifying biases, and researching more optimal pipelines for generation (including which GPT prompts would be better), like the Self-Instruct method by Wang et al. [75], while maintaining the low costs.

7.3.2 Moving on to larger models

At the time of writing, current state-of-the-art is dominated by generative very large language models, like GPT-3.5[4], GPT-4[5] or LLAMA (Touvron et al. [9]). The size of these models goes way over the size of GlórIA, starting with LLAMA at 7B tokens, which competitively performs with the most recent GPT models. A LLAMA would also require more hardware resources.

Future work would pass through pre-training and evaluating a European Portuguese language model using LLAMA as its base model to attempt to produce a more capable, open-source, state-of-the-art PT-PT language model.

[4]https://platform.openai.com/docs/models/gpt-3-5
[5]https://platform.openai.com/docs/models/gpt-4

7.3.3 Continuing Arquivo.PT Crawl & Scrape

As we've seen, a part of our training dataset was data crawled and scraped from *Arquivo.pt* (Gomes et al. [16]). Nevertheless, we gathered only a small fraction of the data that is available. So another future work issue would be focused on exploiting the potential of Arquivo.pt by gathering more archived data from more domains, followed by cleaning it and performing statistical analysis.

The final goal would be to share with researchers a new larger unlabeled text corpus that can be used to pre-train a large language model.

7.3.4 Tokenization Research

Until now, to the extent of our knowledge, no work on Portuguese language models (including this thesis) attempted to analyze how tokenization influences the model's performance in Portuguese: how does the vocabulary size affect performance? Is there an optimal tokenization algorithm (WordPiece, BPE, etc.) for Portuguese? Should you use the original tokenizer of the base model you are using? Should you prepare your tokenizer from scratch on your pretraining data?

These are several questions that have been focused on for the English language, but not for the Portuguese language. Thus becoming an important future line of work.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (cit. on p. ii).

[2] T. Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of Workshop at ICLR* 2013 (2013-01) (cit. on pp. xii, 5, 6).

[3] A. Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. xii, 2, 7–9).

[4] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by J. Burstein, C. Doran, and T. Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423 (cit. on pp. xii, 1, 12–14, 27, 30).

[5] C. Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21.1 (2022-06). ISSN: 1532-4435 (cit. on pp. xii, 15–17, 23).

[6] J. Wei et al. "Finetuned Language Models Are Zero-Shot Learners". In: *CoRR* abs/2109.01652 (2021). arXiv: 2109.01652. URL: https://arxiv.org/abs/2109.01652 (cit. on pp. xii, 20, 22).

[7] T. Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2022. DOI: 10.48550/ARXIV.2205.11916. URL: https://arxiv.org/abs/2205.11916 (cit. on pp. xii, 21).

[8] H. W. Chung et al. *Scaling Instruction-Finetuned Language Models*. 2022. DOI: 10.48550/ARXIV.2210.11416. URL: https://arxiv.org/abs/2210.11416 (cit. on pp. xii, 22, 23).

[9] H. Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL] (cit. on pp. xii, 34, 36, 38, 41, 99).

[10] I. Beltagy, M. E. Peters, and A. Cohan. "Longformer: The Long-Document Transformer". In: *arXiv:2004.05150* (2020) (cit. on pp. xiii, 42, 43).

[11] S. Rajbhandari et al. "ZeRO: Memory Optimizations toward Training Trillion Parameter Models". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '20. Atlanta, Georgia: IEEE Press, 2020. ISBN: 9781728199986 (cit. on pp. xiii, 53).

[12] J. Rodrigues et al. *Advancing Neural Encoding of Portuguese with Transformer Albertina PT-**. 2023. arXiv: 2305.06721 [cs.CL] (cit. on pp. xvii, 1, 30–33, 81, 83, 97, 98).

[13] R. Pires et al. *Sabiá: Portuguese Large Language Models*. 2023. arXiv: 2304.07880 [cs.CL] (cit. on pp. xviii, 34, 73, 77).

[14] A. Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2018). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf (cit. on pp. 1, 2, 15).

[15] T. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf (cit. on pp. 2, 3, 19, 21, 36).

[16] D. Gomes et al. "Introducing the Portuguese web archive initiative". In: 2008 (cit. on pp. 3, 32, 35, 100).

[17] J. Arvana. "Time-aware Question-Answering for the Portuguese Web Archive". MSc diss. NOVA School of Science and Technology, 2023-10 (cit. on p. 4).

[18] D. Castanho. "Structuring and Organizing Large Scale Graph Temporal information". MSc diss. NOVA School of Science and Technology, 2023-10 (cit. on p. 4).

[19] J. Pennington, R. Socher, and C. Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014-10, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (cit. on p. 6).

[20] M. E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018-06, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202 (cit. on p. 6).

[21] A. Zhang et al. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021) (cit. on pp. 7, 18).

[22] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 7).

[23] J. Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". English (US). In: *NIPS 2014 Workshop on Deep Learning, December 2014*. 2014 (cit. on p. 7).

[24] M. Schuster and K. Nakajima. "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079 (cit. on p. 11).

[25] Y. Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: 1609.08144. URL: http://arxiv.org/abs/1609.08144 (cit. on p. 11).

[26] R. Sennrich, B. Haddow, and A. Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016-08, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://aclanthology.org/P16-1162 (cit. on p. 11).

[27] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on pp. 13, 48).

[28] A. Wang et al. "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, 2018-11, pp. 353–355. DOI: 10.18653/v1/W18-5446. URL: https://aclanthology.org/W18-5446 (cit. on pp. 14, 18, 81, 109).

[29] P. Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, 2016-11, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. URL: https://aclanthology.org/D16-1264 (cit. on pp. 14, 18, 110).

[30] P. Rajpurkar, R. Jia, and P. Liang. "Know What You Don't Know: Unanswerable Questions for SQuAD". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018-07, pp. 784–789. DOI: 10.18653/v1/P18-2124. URL: https://aclanthology.org/P18-2124 (cit. on pp. 14, 18).

[31] R. Zellers et al. "SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018-10-11, pp. 93–104. DOI: 10.18653/v1/D18-1009. URL: https://aclanthology.org/D18-1009 (cit. on p. 14).

[32] X. Liu et al. "Multi-Task Deep Neural Networks for Natural Language Understanding". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019-07, pp. 4487–4496. DOI: 10.18653/v1/P19-1441. URL: https://aclanthology.org/P19-1441 (cit. on p. 17).

[33] T. Che et al. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". In: *CoRR* abs/1702.07983 (2017). arXiv: 1702.07983. URL: http://arxiv.org/abs/1702.07983 (cit. on p. 17).

[34] A. Wang et al. "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cit. on pp. 18, 30).

[35] R. Nallapati et al. "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond". In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, 2016-08, pp. 280–290. DOI: 10.18653/v1/K16-1028. URL: https://aclanthology.org/K16-1028 (cit. on p. 18).

[36] O. Bojar et al. "Findings of the 2014 Workshop on Statistical Machine Translation". In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. Baltimore, Maryland, USA: Association for Computational Linguistics, 2014-06, pp. 12–58. URL: http://www.aclweb.org/anthology/W/W14/W14-3302 (cit. on p. 18).

[37] L. Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. DOI: 10.48550/ARXIV.2203.02155. URL: https://arxiv.org/abs/2203.02155 (cit. on p. 19).

[38] B. Workshop et al. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*. 2022. DOI: 10.48550/ARXIV.2211.05100. URL: https://arxiv.org/abs/2211.05100 (cit. on pp. 19, 53).

[39] J. Wei et al. "Chain of Thought Prompting Elicits Reasoning in Large Language Models". In: *CoRR* abs/2201.11903 (2022). arXiv: 2201.11903. URL: https://arxiv.org/abs/2201.11903 (cit. on p. 20).

[40] A. Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113. URL: http://jmlr.org/papers/v24/22-1144.html (cit. on p. 21).

[41] J. Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. DOI: 10.48550/ARXIV.2203.15556. URL: https://arxiv.org/abs/2203.15556 (cit. on p. 21).

[42] Y. You et al. "Reducing BERT Pre-Training Time from 3 Days to 76 Minutes". In: *CoRR* abs/1904.00962 (2019). arXiv: 1904.00962. URL: http://arxiv.org/abs/1904.00962 (cit. on p. 22).

[43] K. Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, 2002-07, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040 (cit. on p. 25).

[44] C.-Y. Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, 2004-07, pp. 74–81. URL: https://aclanthology.org/W04-1013 (cit. on p. 25).

[45] 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*. ISCA '17. Toronto, ON, Canada: Association for Computing Machinery, 2017, pp. 1–12. ISBN: 9781450348928. DOI: 10.1145/3079856.3080246. URL: https://doi.org/10.1145/3079856.3080246 (cit. on p. 26).

[46] D. A. Patterson et al. "Carbon Emissions and Large Neural Network Training". In: *CoRR* abs/2104.10350 (2021). arXiv: 2104.10350. URL: https://arxiv.org/abs/2104.10350 (cit. on p. 27).

[47] M. Yusuf et al. "Curb Your Carbon Emissions: Benchmarking Carbon Emissions in Machine Translation". In: *CoRR* abs/2109.12584 (2021). arXiv: 2109.12584. URL: https://arxiv.org/abs/2109.12584 (cit. on p. 27).

[48] A. K. Vijayakumar et al. *Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models*. 2017. URL: https://openreview.net/forum?id=HJV1zP5xg (cit. on p. 28).

[49] L. Xue et al. "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, 2021-06, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41 (cit. on p. 29).

[50] A. Conneau et al. "XNLI: Evaluating Cross-lingual Sentence Representations". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018-10-11, pp. 2475–2485. DOI: 10.18653/v1/D18-1269. URL: https://aclanthology.org/D18-1269 (cit. on p. 29).

[51] F. Souza, R. Nogueira, and R. de Alencar Lotufo. "BERTimbau: Pretrained BERT Models for Brazilian Portuguese". In: *Brazilian Conference on Intelligent Systems*. 2020 (cit. on pp. 29, 32, 49, 75, 76).

[52] E. T. R. Schneider et al. "BioBERTpt - A Portuguese Neural Language Model for Clinical Named Entity Recognition". In: *Proceedings of the 3rd Clinical Natural Language Processing Workshop*. Online: Association for Computational Linguistics, 2020-11, pp. 65–72. URL: https://www.aclweb.org/anthology/2020.clinicalnlp-1.7 (cit. on p. 30).

[53] N. Miquelina, P. Quaresma, and V. B. Nogueira. "Generating a European Portuguese BERT Based Model Using Content from Arquivo.pt Archive". In: *Intelligent Data Engineering and Automated Learning – IDEAL 2022*. Ed. by H. Yin, D. Camacho, and P. Tino. Cham: Springer International Publishing, 2022, pp. 280–288. ISBN: 978-3-031-21753-1 (cit. on pp. 30, 32).

[54] P. He et al. "DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=XPZIaotutsD (cit. on p. 30).

[55] J. Abadji et al. "Towards a Cleaner Document-Oriented Multilingual Crawled Corpus". In: *Proceedings of the Thirteenth Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, 2022-06, pp. 4344–4355. URL: https://aclanthology.org/2022.lrec-1.463 (cit. on pp. 30, 35).

[56] P. Koehn. "Europarl: A Parallel Corpus for Statistical Machine Translation". In: *Proceedings of Machine Translation Summit X: Papers*. Phuket, Thailand, 2005-9 13-15, pp. 79–86. URL: https://aclanthology.org/2005.mtsummit-papers.11 (cit. on pp. 30, 33).

[57] N. Hajlaoui et al. "DCEP -Digital Corpus of the European Parliament". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), 2014-05. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/943_Paper.pdf (cit. on pp. 30, 33).

[58] J. A. Wagner Filho et al. "The brWaC Corpus: A New Open Resource for Brazilian Portuguese". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), 2018-05. URL: https://aclanthology.org/L18-1686 (cit. on pp. 30, 32).

[59] S. Biderman et al. *Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling*. 2023. arXiv: 2304.01373 [cs.CL] (cit. on pp. 31, 42).

[60] J. R. S. GOMES. *PLUE: Portuguese Language Understanding Evaluation*. https://github.com/ju-resplande/PLUE. 2020 (cit. on pp. 33, 81, 109).

[61] F. Ladhak et al. "WikiLingua: A New Benchmark Dataset for Cross-Lingual Abstractive Summarization". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, 2020-11, pp. 4034–4048. DOI: 10.18653/v1/2020.findings-emnlp.360. URL: https://aclanthology.org/2020.findings-emnlp.360 (cit. on p. 33).

[62] A. Overwijk, C. Xiong, and J. Callan. "ClueWeb22: 10 Billion Web Documents with Rich Information". In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 3360–3362. ISBN: 9781450387323. DOI: 10.1145/3477495.3536321. URL: https://doi.org/10.1145/3477495.3536321 (cit. on pp. 34, 36).

[63] P. Lison and J. Tiedemann. "OpenSubtitles2016: Extracting Large Parallel Corpora from Movie and TV Subtitles". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), 2016-05, pp. 923–929. URL: https://aclanthology.org/L16-1147 (cit. on pp. 35, 37).

[64] S. Black et al. *GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow*. Version 1.0. If you use this software, please cite it using these metadata. 2021-03. DOI: 10.5281/zenodo.5297715. URL: https://doi.org/10.5281/zenodo.5297715 (cit. on p. 41).

[65] L. Gao et al. "The Pile: An 800GB Dataset of Diverse Text for Language Modeling". In: *arXiv preprint arXiv:2101.00027* (2020) (cit. on p. 41).

[66] S. Zhuoran et al. "Efficient Attention: Attention with Linear Complexities". In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2021, pp. 3530–3538. DOI: 10.1109/WACV48630.2021.00357 (cit. on p. 42).

[67] I. Loshchilov and F. Hutter. "Decoupled Weight Decay Regularization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7 (cit. on pp. 48, 49).

[68] S. Gugger et al. *Accelerate: Training and inference at scale made simple, efficient and adaptable.* https://github.com/huggingface/accelerate. 2022 (cit. on p. 52).

[69] S. Black et al. "GPT-NeoX-20B: An Open-Source Autoregressive Language Model". In: *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*. 2022. URL: https://arxiv.org/abs/2204.06745 (cit. on p. 53).

[70] Y. Pan et al. *On the Risk of Misinformation Pollution with Large Language Models*. 2023. arXiv: 2305.13661 [cs.CL] (cit. on p. 66).

[71] A. Agrawal, L. Mackey, and A. T. Kalai. *Do Language Models Know When They're Hallucinating References?* 2023. arXiv: 2305.18248 [cs.CL] (cit. on p. 66).

[72] D. Paperno et al. "The LAMBADA dataset: Word prediction requiring a broad discourse context". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016-08, pp. 1525–1534. DOI: 10.18653/v1/P16-1144. URL: https://aclanthology.org/P16-1144 (cit. on p. 66).

[73] N. S. Keskar et al. "CTRL - A Conditional Transformer Language Model for Controllable Generation". In: *arXiv preprint arXiv:1909.05858* (2019) (cit. on p. 70).

[74] L. Real, E. Fonseca, and H. G. Oliveira. "The assin 2 shared task: a quick overview". In: *International Conference on Computational Processing of the Portuguese Language*. Springer. 2020, pp. 406–412 (cit. on pp. 74, 98).

[75] Y. Wang et al. "Self-Instruct: Aligning Language Models with Self-Generated Instructions". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, 2023-07, pp. 13484–13508. DOI: 10.18653/v1/2023.acl-long.754. URL: https://aclanthology.org/2023.acl-long.754 (cit. on p. 99).

I

PLUE: Portuguese Glue Samples

This annex serves to demonstrate a few examples of the Multi-Genre Natural Language (MNLI) evaluation set contained in the Portuguese translated version of GLUE(Wang et al. [28]), aka PLUE (GOMES [60]). This is a collection of sentence-pairs - a premise and an hypothesis - with the goal of determining the relation between these two. Specifically, if the premise either entails, contradicts or is not related at all to the hypothesis.

I.1 Sentence-pair Examples

- **Example Pair 1:**
  - **Premise:** Não é que as perguntas que fizeram não fossem interessantes ou legítimas (embora a maioria caísse na categoria de perguntas e respostas já feitas).
  - **Hypothesis:** Todas as perguntas foram interessantes de acordo com um grupo focal consultado sobre o assunto.
  - **Label:** Neutral
- **Example Pair 2:**
  - **Premise:** Representado pelo Primeiro- Ministro Atal Behari Vajpayee, o BJP foi forçado a ceder seu lugar em menos de duas semanas, no entanto, tendo falhado nos esforços para formar um governo de coligação.
  - **Hypothesis:** O BJP teve um prazo curto devido à formação fracassada do governo da coalizão.
  - **Label:** Entailment
- **Example Pair 3:**
  - **Premise:** Eles me vigiaram constantemente por semanas.
  - **Hypothesis:** Eles me deixaram sozinha por semanas.
  - **Label:** Contradiction

II

SQUADPT Samples

This annex serves to show a few samples of the Portuguese translated version of the SQUAD[29] dataset[1]. Each samples contains a paragraph from some document, followed by questions about its content and respective answers.

II.1 Sample 1 - Universidade de Notre Dame

Arquitetonicamente, a escola tem um caráter católico. No topo da cúpula de ouro do edifício principal é uma estátua de ouro da Virgem Maria. Imediatamente em frente ao edifício principal e de frente para ele, é uma estátua de cobre de Cristo com os braços erguidos com a lenda "Venite Ad Me Omnes". Ao lado do edifício principal é a Basílica do Sagrado Coração. Imediatamente atrás da basílica é a Gruta, um lugar mariano de oração e reflexão. É uma réplica da gruta em Lourdes, na França, onde a Virgem Maria supostamente apareceu a Santa Bernadette Soubirous em 1858. No final da unidade principal (e em uma linha direta que liga através de 3 estátuas e da Cúpula de Ouro), é um estátua de pedra simples e moderna de Maria.

- **Q: A quem a Virgem Maria supostamente apareceu em 1858 em Lourdes, na França?**

  A: Saint Bernadette Soubirous

- **Q: O que fica em frente ao edifício principal de Notre Dame?**

  A: uma estátua de cobre de Cristo

- **Q: A Basílica do Sagrado Coração em Notre Dame fica ao lado de qual estrutura?**
  A: o edifício principal

- **Q: O que é a gruta de Notre Dame?**

  A: um lugar mariano de oração e reflexão

[1] https://github.com/nunorc/squad-v1.1-pt

- **Q: O que fica no topo do edifício principal em Notre Dame?**

  A: uma estátua de ouro da Virgem Maria

II.2 Sample 2 - IPod

Antes do lançamento do iOS 5, a marca do iPod era usada para o media player incluído no iPhone e no iPad, uma combinação dos aplicativos Music e Videos no iPod Touch. No iOS 5, aplicativos separados chamados "Música" e "Vídeos" são padronizados em todos os produtos com tecnologia iOS. Embora o iPhone e o iPad tenham essencialmente os mesmos recursos de media player da linha iPod, eles geralmente são tratados como produtos separados. Em meados de 2010, as vendas do iPhone superaram as do iPod.

- **Q: Antes do iOS 5, quantos aplicativos eram necessários para reproduzir músicas e vídeos no iPhone e no iPad?**

  A: 1

- **Q: Em meados de 2010, qual dispositivo da Apple teve vendas maiores do que o iPod?**

  A: Iphone

- **Q: Com que versão do iOS a Apple padronizou aplicativos de mídia em todos os seus produtos? A:** iOS 5

- **Q: Em que ano as vendas do iPhone superaram as dos iPods?**

  A: 2010

- **Q: Quais são os títulos dos aplicativos de mídia padrão nos dispositivos atuais da Apple?**

  A: "Música" e "Vídeos"

III

Annex - Pre-Training Graphs

This annex contains the graphs for the pretraining's loss and perplexity with both their original data and the exponentiated moving average line.

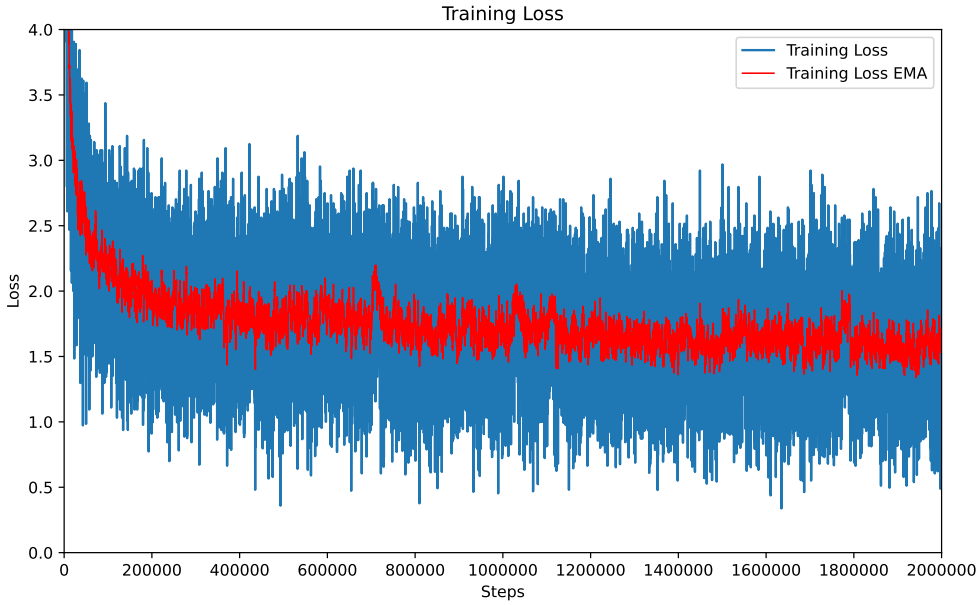

Figure III.1: Loss for the 2M steps of GlórIA 1.3B's pre-training. The red line is the exponential moving average of the loss with a smoothing factor of 0.90 (*alpha* of 0.1). The blue line is the original data.

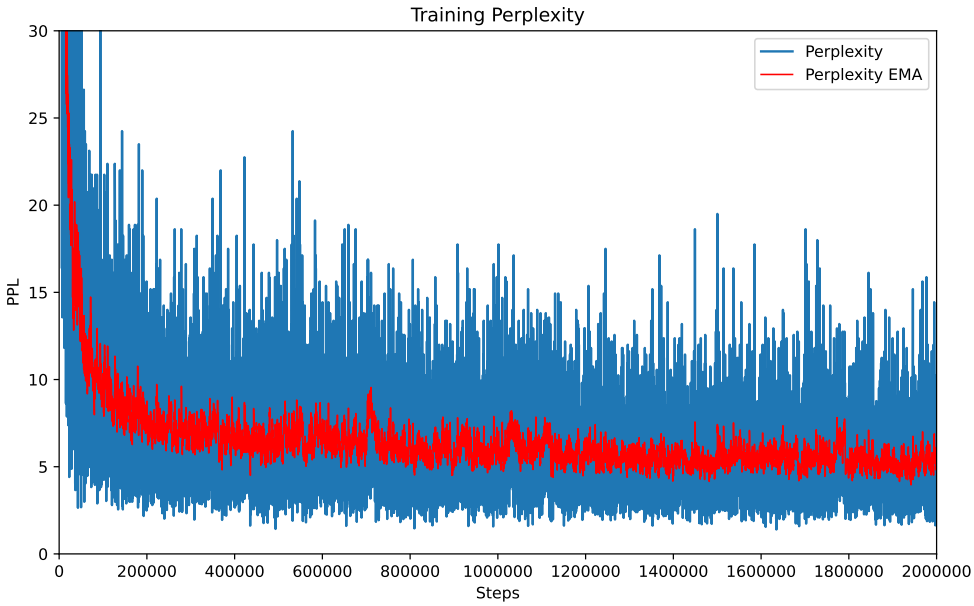

Figure III.2: Perplexity for the 2M steps of GlórIA 1.3B's pre-training. The red line is the exponential moving average of the loss with a smoothing factor of 0.90 (*alpha* of 0.1). The blue line is the original data.

IV

ASSIN-2 Experiments

IV.1 BERTimbau-Large Experiments

BERTimbau-Large		Training 1			BERTimbau-Large		Training 2	
Hyperparameter Set	**F1**	**Accuracy**	**Pearson**		**Hyperparameter Set**	**F1**	**Accuracy**	**Pearson**
A0	0.895	0.8952	0.844		**A0**	0.887	0.8887	0.844
B0	**0.902**	**0.902**	0.846		**B0**	0.9	0.8999	0.835
C0	0.893	0.893	0.836		**C0**	0.8858	0.886	0.844
D0	0.894	0.8946	0.821		**D0**	0.887	0.8881	0.841
E0	0.845	0.8452	0.815		**E0**	0.861	0.8609	0.823
F0	0.871	0.8717	0.823		**F0**	0.86	0.8609	0.823
G0	0.805	0.8056	0.788		**G0**	0.822	0.8219	0.802
H0	0.853	0.8538	0.816		**H0**	0.822	0.8219	0.802

Table IV.1: Finetunes performed for **BERTimbau-Large's ASSIN-2 task** for the two training sessions in our experimental space. Includes all the parameter sets used and both seeds tested: 41 and 42.

IV.2 GlórIA 1.3B Experiments

GlórIA Checkpoint 1M		Training 1			GlórIA Checkpoint 1M		Training 2	
Hyperparameter Set	**F1**	**Accuracy**	**Pearson**		**Hyperparameter Set**	**F1**	**Accuracy**	**Pearson**
A0	0.8920	0.8918	0.829		**A0**	0.894	0.8944	0.828
B0	0.8860	0.8859	**0.851**		**B0**	0.884	0.8838	0.823
C0	0.8840	0.8846	0.823		**C0**	0.885	0.8852	0.822
D0	0.8779	0.8775	0.820		**D0**	0.887	0.8873	0.818
E0	0.8530	0.8529	0.773		**E0**	0.862	0.8619	0.779
F0	0.8700	0.8701	0.772		**F0**	0.862	0.8619	0.779
G0	0.7780	0.7782	0.742		**G0**	0.787	0.7868	0.745
H0	0.8390	0.8395	0.783		**H0**	0.787	0.7868	0.745

Table IV.2: **ASSIN-2** finetunes performed for **GlórIA's 1M steps checkpoint**, for the two training sessions in our experimental space. Includes all the parameter sets used and both seeds tested: 41 and 42.

GlórIA Checkpoint 1.5M		Training 1		GlórIA Checkpoint 1.5M		Training 2	
Hyperparameter Set	**F1**	**Accuracy**	**Pearson**	**Hyperparameter Set**	**F1**	**Accuracy**	**Pearson**
A0	0,888	0,8877	0,828	**A0**	0,886	0,8865	0,826
B0	0,895	0,8949	0,826	**B0**	0,892	0,8926	0,827
C0	0,881	0,8805	0,821	**C0**	0,88	0,8799	0,821
D0	0,887	0,8873	0,822	**D0**	0,885	0,8854	0,823
E0	0,854	0,8532	0,767	**E0**	0,861	0,8607	0,777
F0	0,868	0,8683	0,766	**F0**	0,868	0,8685	0,774
G0	0,788	0,788	0,754	**G0**	0,797	0,7966	0,752
H0	0,852	0,8521	0,777	**H0**	0,862	0,8627	0,78

Table IV.3: **ASSIN-2** finetunes performed for **GlórIA's 1.5M steps checkpoint**, for the two training sessions in our experimental space. Includes all the parameter sets used and both seeds tested: 41 and 42

GlórIA Checkpoint 2M		Training 1		GlórIA Checkpoint 2M		Training 2	
Hyperparameter Set	**F1**	**Accuracy**	**Pearson**	**Hyperparameter Set**	**F1**	**Accuracy**	**Pearson**
A0	0,891	0,8914	0,828	**A0**	0,894	0,8946	0,831
B0	0,896	0,896	0,829	**B0**	0,859	0,8816	0,826
C0	0,89	0,8901	0,827	**C0**	0,859	0,8816	0,826
D0	0,885	0,8848	0,812	**D0**	0,887	0,8867	0,82
E0	0,866	0,8652	0,77	**E0**	0,86	0,8603	0,783
F0	0,876	0,8762	0,776	**F0**	0,864	0,8646	0,785
G0	0,784	0,7847	0,751	**G0**	0,864	0,8646	0,785
H0	0,86	0,8599	0,78	**H0**	0,855	0,8554	0,789

Table IV.4: **ASSIN-2** finetunes performed for **GlórIA's 2M steps checkpoint** for the two training sessions in our experimental space. Includes all the parameter sets used and both seeds tested: 41 and 42

GlórIA Checkpoint 3M		Training 1		GlórIA Checkpoint 3M		Training 2	
Hyperparameter Set	**F1**	**Accuracy**	**Pearson**	**Hyperparameter Set**	**F1**	**Accuracy**	**Pearson**
A0	0,896	0,8967	0,834	**A0**	0,89	0,8901	0,833
B0	**0,896**	**0,8967**	0,834	**B0**	0,894	0,8938	0,828
C0	0,888	0,8885	0,832	**C0**	0,887	0,8867	0,827
D0	0,879	0,8791	0,827	**D0**	0,884	0,8838	0,817
E0	0,853	0,8536	0,781	**E0**	0,848	0,8485	0,772
F0	0,865	0,865	0,787	**F0**	0,864	0,865	0,778
G0	0,775	0,7757	0,746	**G0**	0,78	0,7802	0,741
H0	0,855	0,8546	0,79	**H0**	0,864	0,865	0,778

Table IV.5: **ASSIN-2** finetunes performed for **GlórIA's 3M steps checkpoint**, for the two training sessions in our experimental space. Includes all the parameter sets used and both seeds tested: 41 and 42

V

GLUE-PTPT Experiments

V.1 RTE Experiments

GlórIA Checkpoint 1M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5042	0,5847	0,6144
B1	0,6907	0,6653	0,6568
C1	0,4958	0,5381	0,5678
D1	0,6653	0,6483	0,6398

Table V.1: Finetunes performed for GlórIA's checkpoint of 1M steps, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 1.5M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,6144	0,5932	0,5636
B1	0,6356	0,6483	0,6568
C1	0,6186	0,6398	0,5
D1	0,6737	0,6271	0,6653

Table V.2: Finetunes performed for GlórIA's checkpoint of 1.5M steps, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 2M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,6017	0,6483	0,572
B1	0,6441	0,6525	0,627
C1	0,5254	0,5678	0,5678
D1	0,6737	0,6568	0,6483

Table V.3: Finetunes performed for GlórIA's checkpoint of 2M steps, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 3M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,661	0,5975	0,661
B1	0,6483	0,5424	0,5932
C1	0,5847	0,5932	0,6441
D1	0,6653	0,6441	0,6398

Table V.4: Finetunes performed for GlórIA's checkpoint of 3M steps, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Albertina-PTPT	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5042	0,5042	0,5042
B1	0,7754	0,8051	0,8093
C1	0,5042	0,5042	0,5042
D1	0,7966	**0,8136**	0,7924

Table V.5: Finetunes performed for Albertina-PTPT, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Gervasio-PTPT	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,6017	0,5127	0,5551
B1	0,589	0,6356	0,6441
C1	0,5085	0,5212	0,5000
D1	0,6398	0,5932	0,5720

Table V.6: Finetunes performed for Gervasio-PTPT, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

BERTimbau(L)	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5042	0,6229	0,5042
B1	0,661	0,5636	0,6186
C1	0,5042	0,5042	0,5042
D1	0,6483	0,6164	0,6059

Table V.7: Finetunes performed for BERTimbau-Large, for the RTE subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

V.2 MRPC Experiments

GlórIA Checkpoint 1M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,8633	0,8081	0,8102	0,7374	0,764	0,6288
B1	0,8633	0,8081	0,8481	0,7828	0,8677	0,8106
C1	0,8138	0,7146	0,7912	0,6816	0,7764	0,6591
D1	0,8572	0,8031	0,8462	0,7879	0,8544	0,7904

Table V.8: Finetunes performed for GlórIA's 1M steps checkpoint, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 1.5M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,7898	0,6831	0,7915	0,697	0,7729	0,6755
B1	0,8702	0,8131	0,8648	0,8081	0,8462	0,7778
C1	0,7787	0,6742	0,816	0,7298	0,8283	0,7551
D1	0,8041	0,7576	0,8642	0,8056	0,8642	0,8056

Table V.9: Finetunes performed for GlórIA's 1.5M steps checkpoint, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 2M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,8246	0,7551	0,8404	0,7777	0,8536	0,7904
B1	0,843	0,7753	0,8486	0,7828	0,8502	0,7828
C1	0,8262	0,7323	0,8345	0,7626	0,7721	0,678
D1	0,8313	0,7601	0,8551	0,7955	0,8582	0,8005

Table V.10: Finetunes performed for GlórIA's 2M steps checkpoint, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 3M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,8018	0,7311	0,8335	0,7665	0,8449	0,7816
B1	0,8208	0,7475	0,8342	0,7702	0,8541	0,7841
C1	0,7919	0,6730	0,7346	0,6604	0,8431	0,7828
D1	0,8275	0,7652	0,8248	0,7576	0,8414	0,7677

Table V.11: Finetunes performed for GlórIA's 3M steps checkpoint, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Albertina-PTPT	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,8987	0,8662	0,8	0,6667	0,8	0,6667
B1	0,8987	0,8662	0,9014	0,8712	**0,9126**	**0,8864**
C1	0,8	0,6667	0,8	0,6667	0,8	0,6667
D1	0,9105	0,8813	0,9002	0,8611	0,9028	0,8662

Table V.12: Finetunes performed for Albertina-PTPT, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Gervasio-PTPT	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,7880	0,6894	0,7712	0,6869	0,7920	0,6907
B1	0,8380	0,7740	0,8551	0,7929	0,8453	0,7652
C1	0,7800	0,6831	0,7881	0,6869	0,7400	0,6540
D1	0,8366	0,7601	0,8505	0,7904	0,8428	0,7753

Table V.13: Finetunes performed for Gervasio-PTPT, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

BERTimbau(L)	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**ACC**	**F1**	**ACC**	**F1**	**ACC**
A1	0,8587	0,8081	0,8	0,6667	0,8	0,6667
B1	0,8587	0,8081	0,8646	0,8157	0,8724	0,8308
C1	0,8	0,6667	0,8	0,6667	0,8	0,6667
D1	0,8757	0,8359	0,8712	0,8283	0,8779	0,8384

Table V.14: Finetunes performed for BERTimbau-Large, for the MRPC subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

V.3 WNLI Experiments

GlórIA Checkpoint 1M	**Accuracy**		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5455	0,5455	0,5455
B1	0,5455	0,4394	0,4545
C1	0,5455	0,5455	0,5455
D1	0,5455	0,4545	0,4545

Table V.15: Finetunes performed for GlórIA's 1M steps checkpoint, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 1.5M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5455	0,5455	0,3485
B1	0,5455	0,4545	0,4545
C1	0,5455	0,5455	0,4697
D1	0,5455	0,4545	0,4545

Table V.16: Finetunes performed for GlórIA's 1.5M steps checkpoint, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 2M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5455	0,5455	0,5455
B1	0,5455	0,4545	0,4545
C1	0,5455	0,5455	0,5455
D1	0,5455	0,4545	0,4394

Table V.17: Finetunes performed for GlórIA's 2M steps checkpoint, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 3M	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5152	0,5455	0,5000
B1	0,5455	0,4545	0,4545
C1	0,5455	0,5455	0,5455
D1	0,5455	0,4545	0,4394

Table V.18: Finetunes performed for GlórIA's 3M steps checkpoint, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Albertina-PTPT	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5455	0,5455	0,5455
B1	0,3636	0,4848	0,4091
C1	0,5455	0,5455	0,5455
D1	0,4091	0,4697	0,5303

Table V.19: Finetunes performed for Albertina-PTPT, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Gervasio-PTPT	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,4545	0,5455	0,5455
B1	0,3182	0,4555	0,4545
C1	0,5455	0,5	0,5455
D1	0,5455	0,5455	0,5152

Table V.20: Finetunes performed for Gervasio-PTPT, for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

BERTimbau(L)	Accuracy		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A1	0,5455	0,5455	0,5455
B1	0,3636	0,4242	0,4545
C1	0,5455	0,5455	0,5455
D1	0,3939	0,4394	0,4848

Table V.21: Finetunes performed for BERTimbau-Large for the WNLI subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

V.4 STSB Experiments

GlórIA Checkpoint 1M	Pearson		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,771	0,348	0,776
B2	0,845	0,831	0,843
C2	0,78	0,678	0,708
D2	0,29	0,8	0,792

Table V.22: Finetunes performed for GlórIA's 1M steps checkpoint, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 1.5M	Pearson		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,799	0,746	0,793
B2	0,846	0,838	0,842
C2	0,778	0,681	0,717
D2	0,246	0,777	0,821

Table V.23: Finetunes performed for GlórIA's 1.5M steps checkpoint, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 2M	Pearson		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,795	0,264	0,375
B2	0,853	0,836	0,847
C2	0,783	0,702	0,721
D2	0,826	0,266	0,657

Table V.24: Finetunes performed for GlórIA's 2M steps checkpoint, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

GlórIA Checkpoint 3M	**Pearson**		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,801	0,678	0,582
B2	0,851	0,851	0,849
C2	0,768	0,671	0,721
D2	0,828	0,818	0,814

Table V.25: Finetunes performed for GlórIA's 3M steps checkpoint, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Albertina-PTPT	**Pearson**		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	-0,013	0,006	0,153
B2	0,903	**0,904**	0,892
C2	0,888	0,894	0,895
D2	0,036	-0,008	0,045

Table V.26: Finetunes performed for Albertina-PTPT, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

Gervasio-PTPT	**Pearson**		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,695	0,716	0,705
B2	0,824	0,804	0,813
C2	0,752	0,742	0,752
D2	0,571	0,548	0,465

Table V.27: Finetunes performed for Gervasio-PTPT, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

BERTimbau(L)	**Pearson**		
Hyperparameter Set	**Training 1**	**Training 2**	**Training 3**
A2	0,861	0,822	0,873
B2	0,891	0,883	0,888
C2	0,819	0,849	0,844
D2	0,881	0,876	0,824

Table V.28: Finetunes performed for BERTimbau, for the STS-B subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

VI

GLUE-PTPT Graphs Comparison

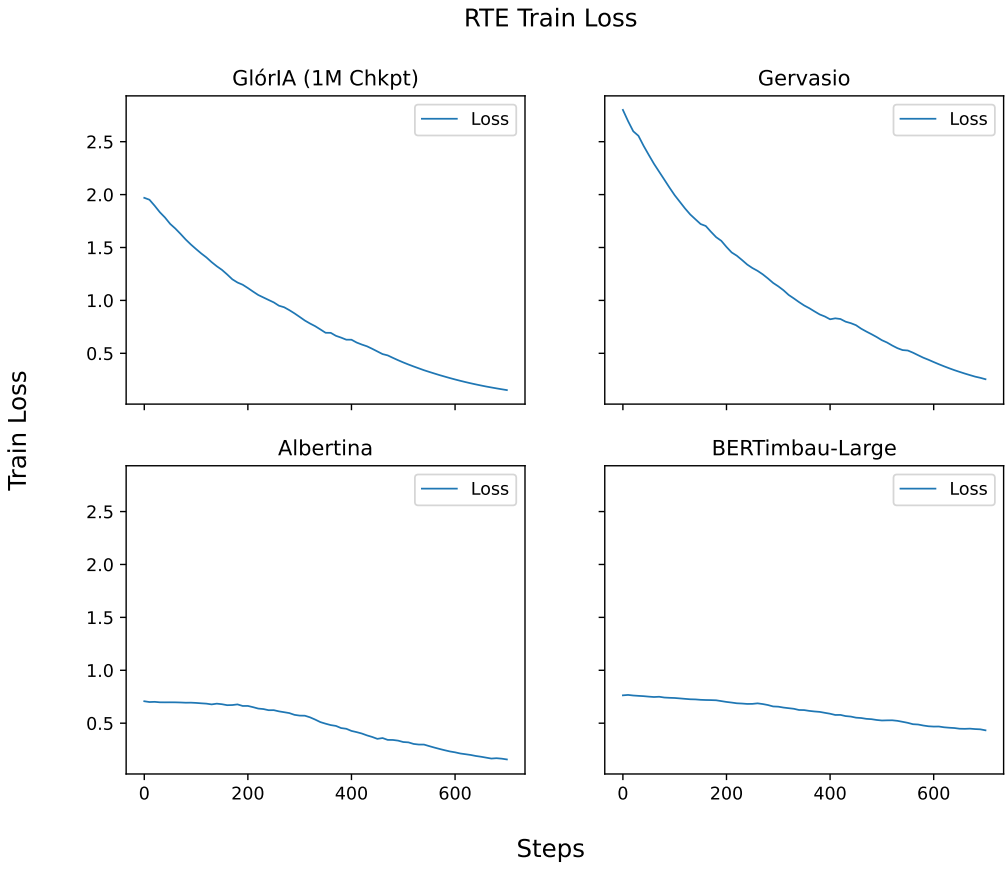

Figure VI.1: Comparison of the RTE's training loss from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set. Using the exponentiated moving average with a smooth factor of 0.95.

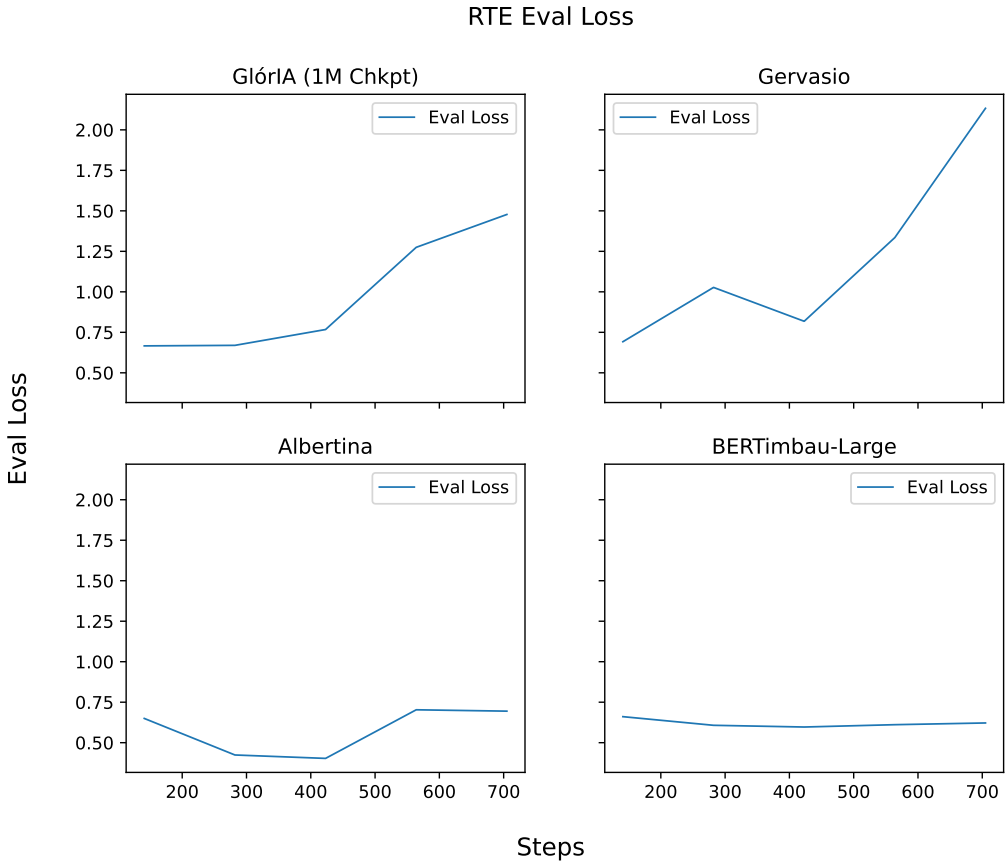

Figure VI.2: Comparison of the RTE's evaluation loss (in-training) from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set.

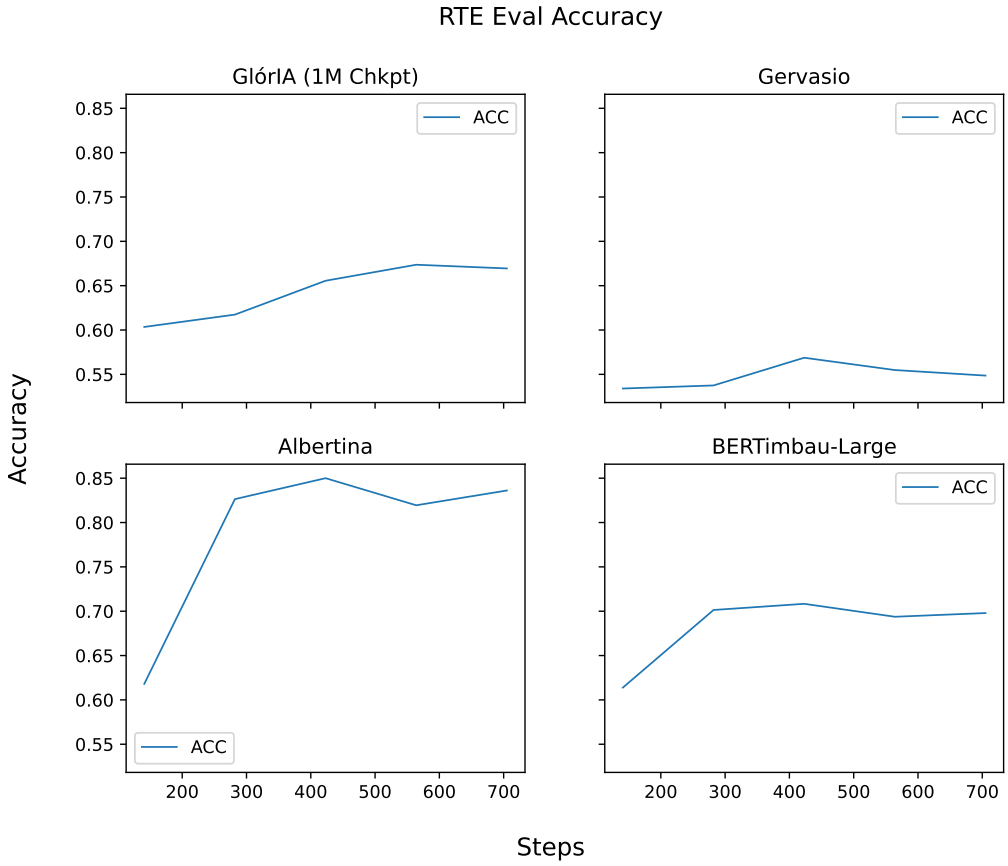

Figure VI.3: Comparison of the RTE's accuracy (in-training evaluation, using eval set) from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set.

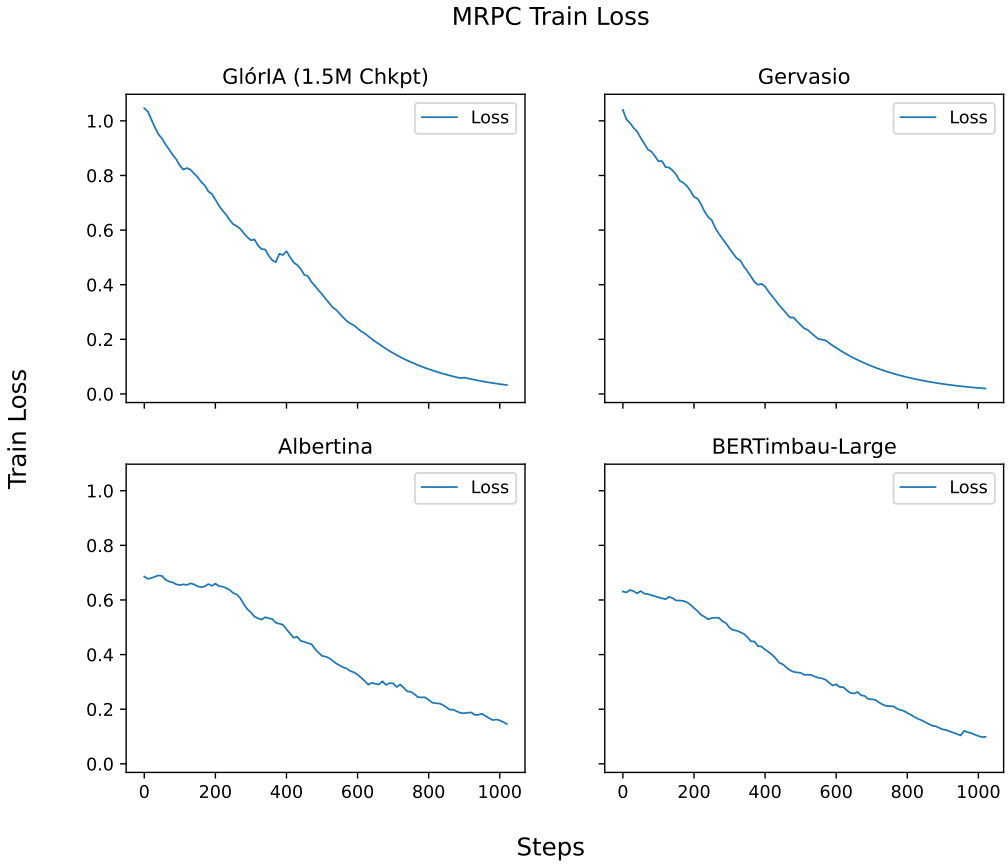

Figure VI.4: Comparison of the MRPC's training loss from all the evaluated models. The data comes from the runs that gave us the peak F1 and Accuracy values when evaluating with the test set. Using the exponentiated moving average with a smooth factor of 0.95.

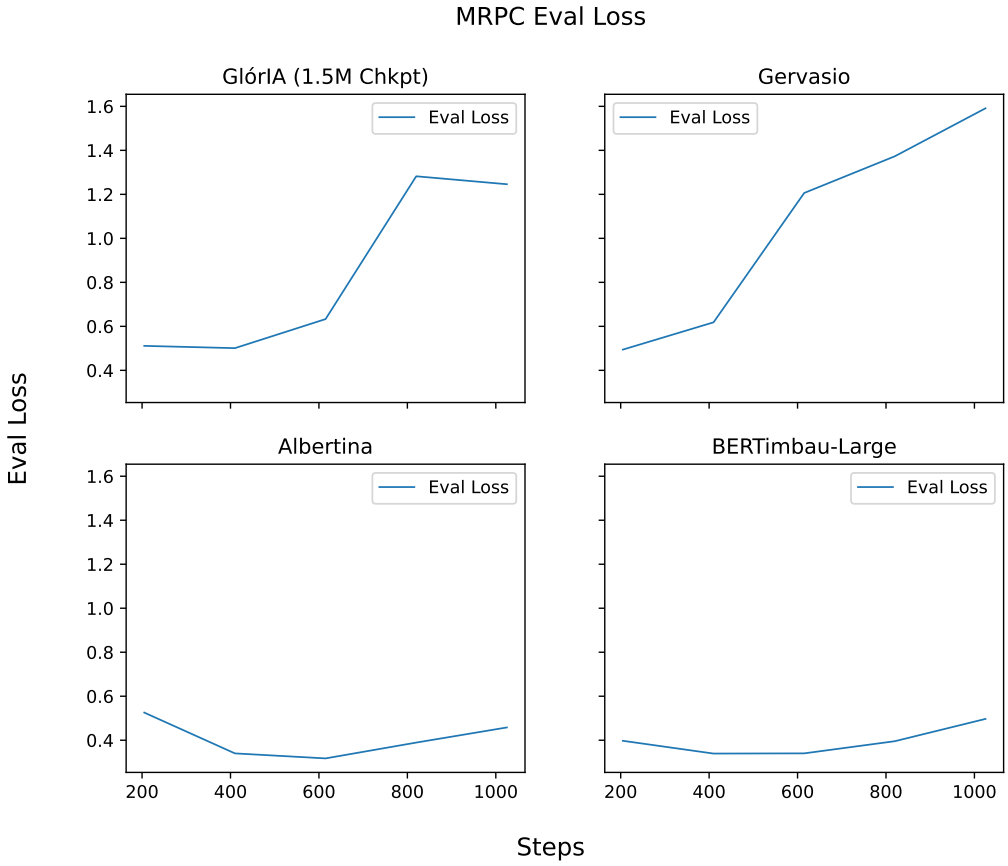

Figure VI.5: Comparison of the MRPC's evaluation loss from all the evaluated models. The data comes from the runs that gave us the peak F1 and Accuracy values when evaluating with the test set.

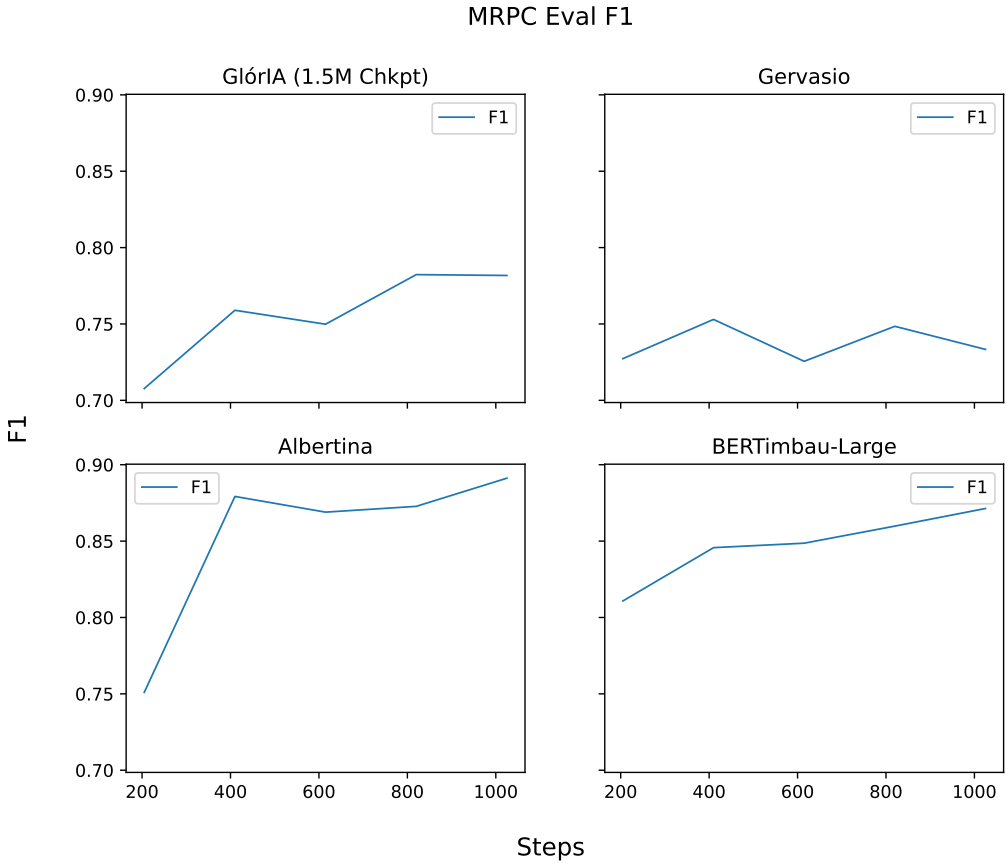

Figure VI.6: Comparison of the MRPC's evaluation F1 score (in-training, with eval set from all the evaluated models. The data comes from the runs that gave us the peak F1 and Accuracy values when evaluating with the test set.

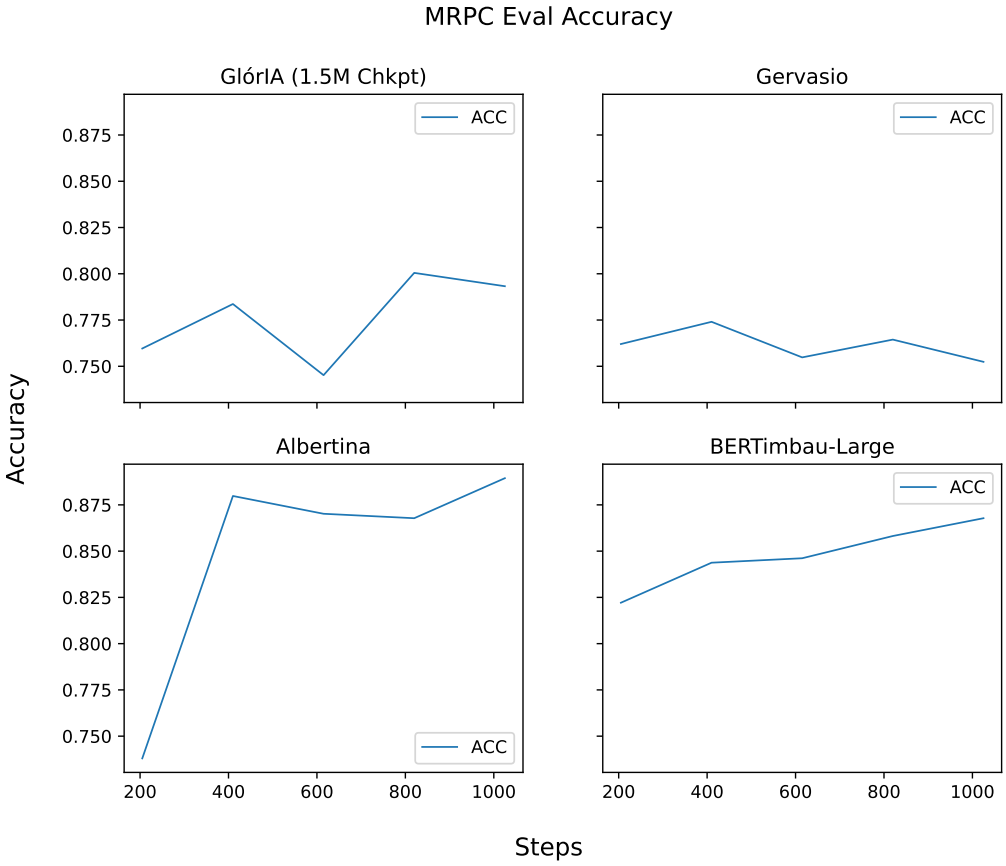

Figure VI.7: Comparison of the MRPC's evaluation accuracy (in-training, with eval set) from all the evaluated models. The data comes from the runs that gave us the peak F1 and Accuracy values when evaluating with the test set.

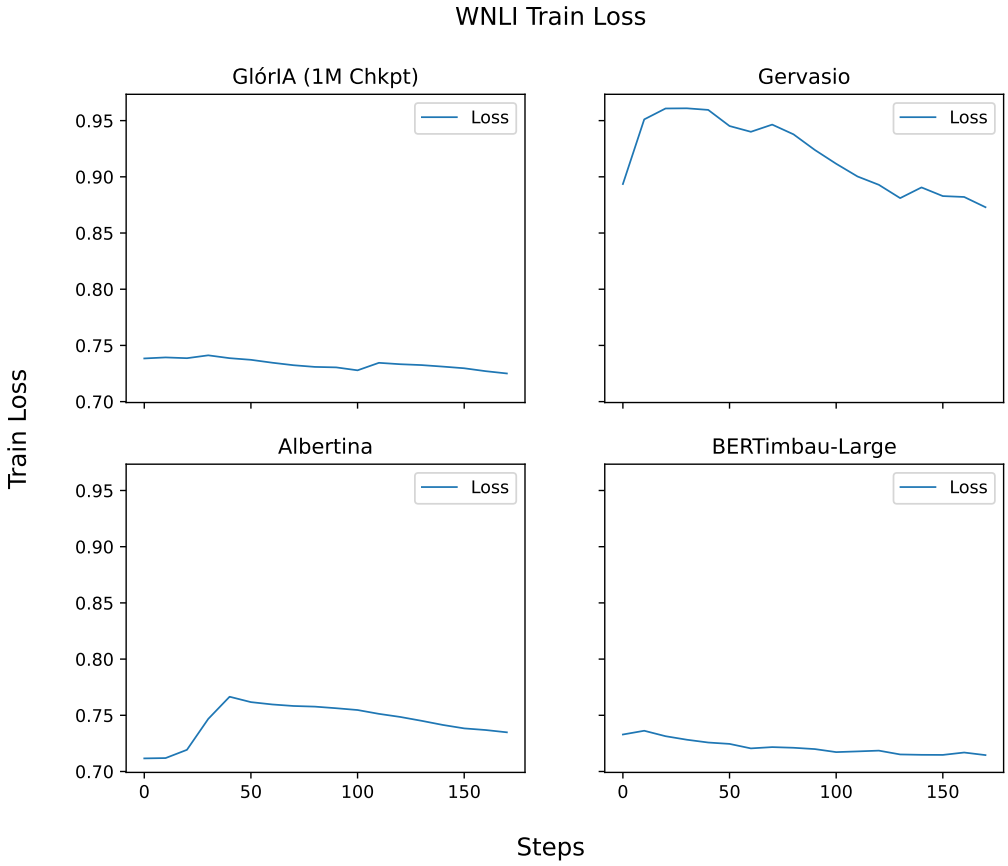

Figure VI.8: Comparison of the WNLI's training loss from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set. Using the exponentiated moving average with a smooth factor of 0.95.

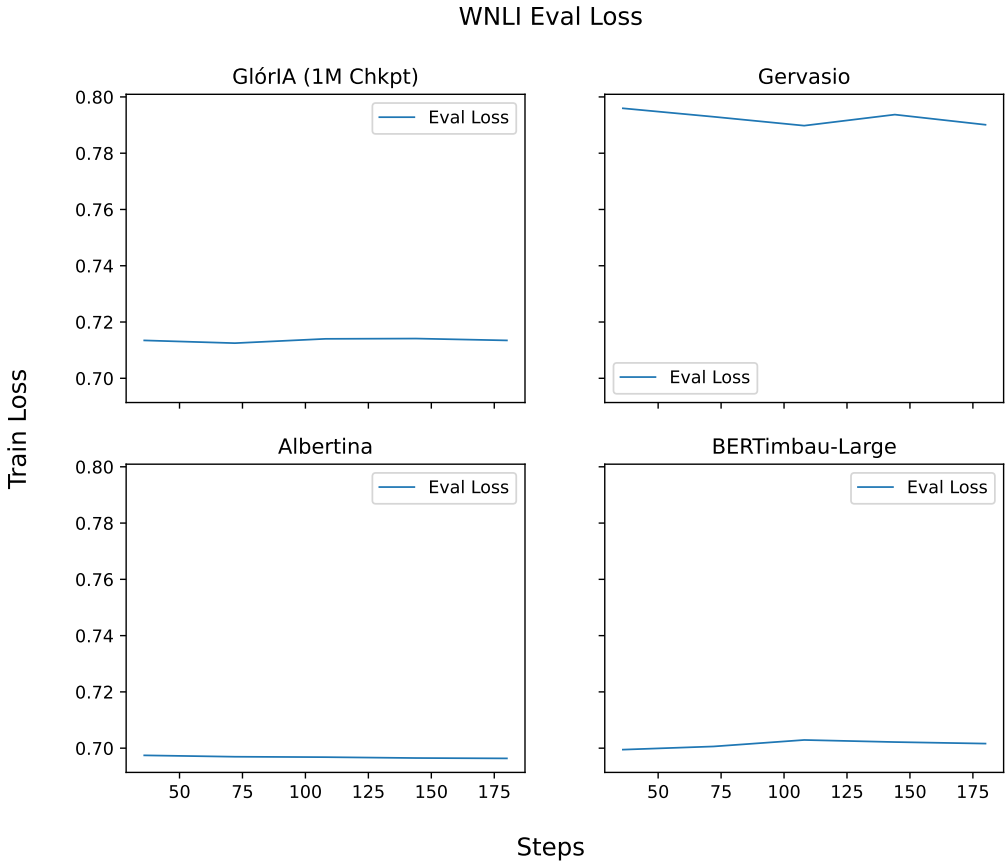

Figure VI.9: Comparison of the WNLI's evaluation loss (in-training) from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set.

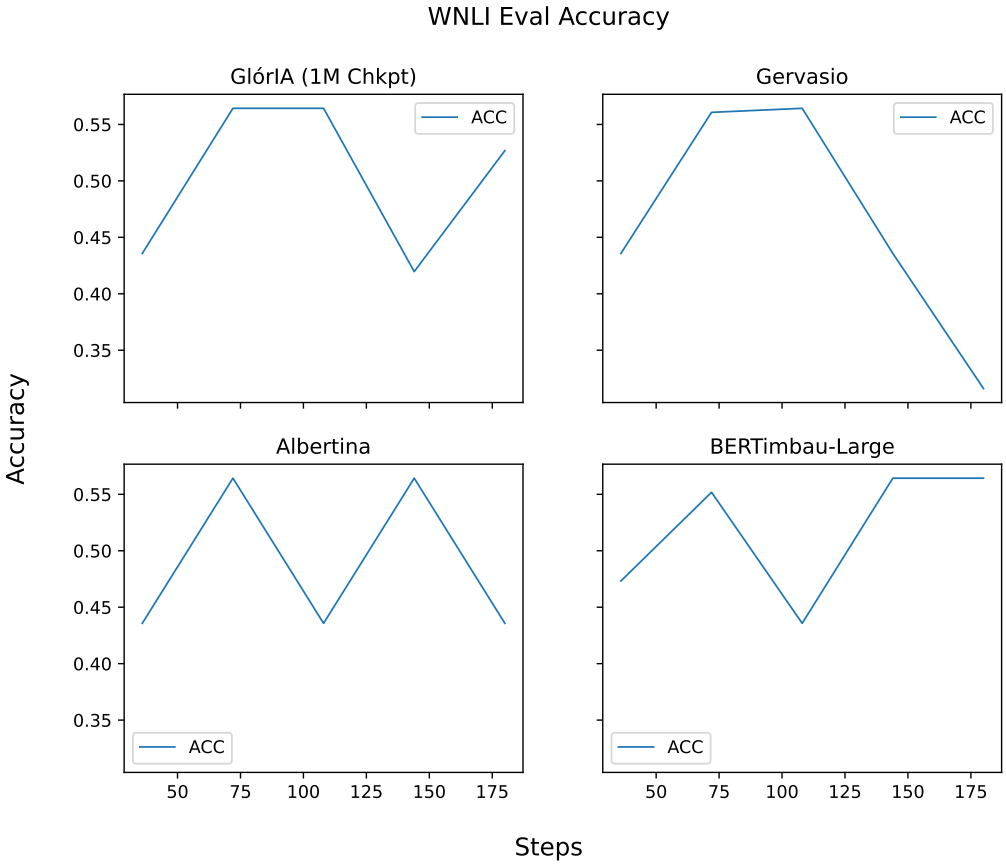

Figure VI.10: Comparison of the WNLI's accuracy (in-training evaluation, using eval set) from all the evaluated models. The data comes from the runs that gave us the peak Accuracy values when evaluating with the test set.

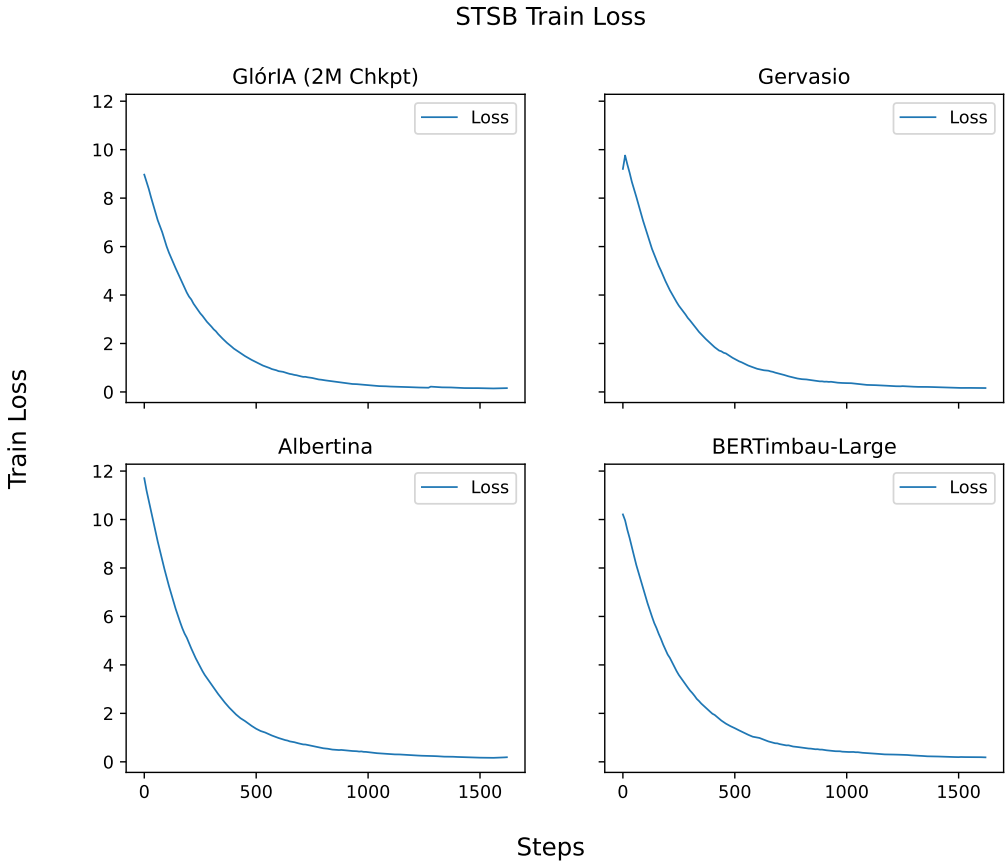

Figure VI.11: Comparison of the STSB's training loss from all the evaluated models. The data comes from the runs that gave us the peak Pearson values when evaluating with the test set. Using the exponentiated moving average with a smooth factor of 0.95.

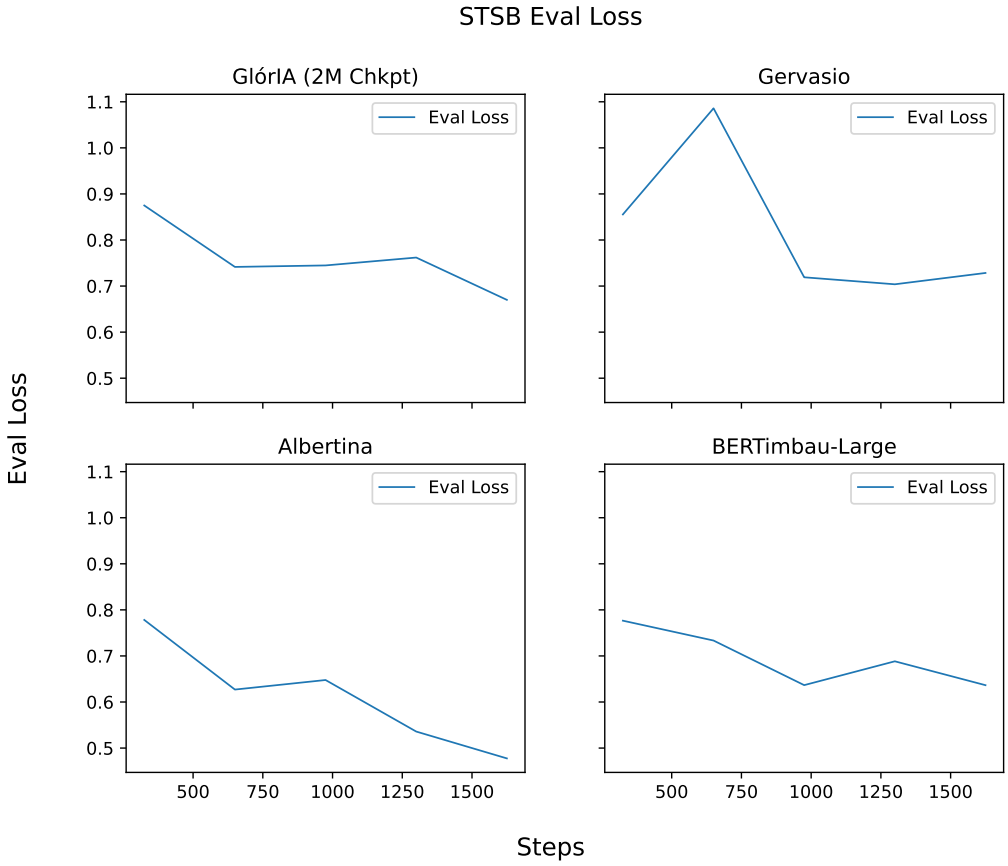

Figure VI.12: Comparison of the STSB's evaluation loss (in-training) from all the evaluated models. The data comes from the runs that gave us the peak Pearson values when evaluating with the test set.

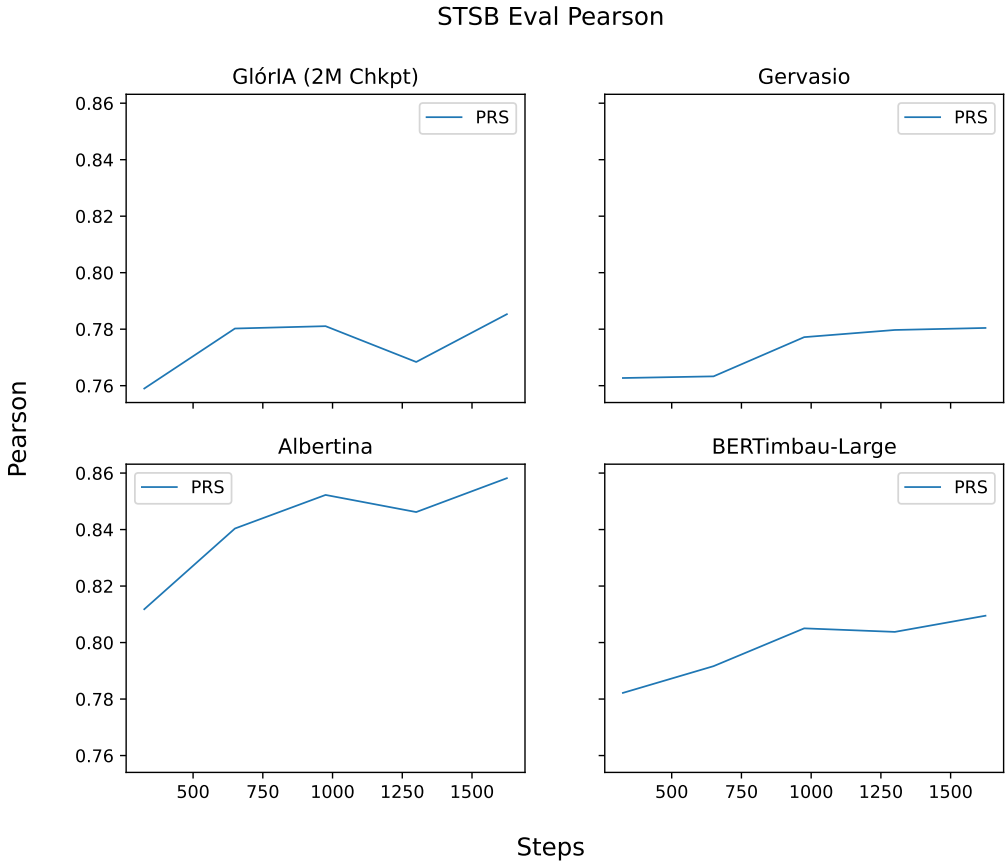

Figure VI.13: Comparison of the STSB's Pearson values (in-training evaluation, using eval set) from all the evaluated models. The data comes from the runs that gave us the peak Pearson values when evaluating with the test set.

VII

SQUADPT Experiments

VII.1 GlórIA 1.3B Experiments

SQUADPT Experiments						
GlórIA Checkpoint 1M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,6394	0,3256	0,6394	0,3297	0,6401	0,3287
B3	0,6586	0,3321	0,6604	0,3341	0,6602	0,3336
C3	0,6081	0,3006	0,6116	0,305	0,6084	0,2992

Table VII.1: Finetunes performed for GlórIA's Checkpoint of 1M steps, for the SQUADPT subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

SQUADPT Experiments						
GlórIA Checkpoint 2M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,6489	0,3337	0,6513	0,3355	0,6444	0,3321
B3	0,6646	0,3375	0,666	0,3357	0,6655	0,3363
C3	0,6104	0,3015	0,6145	0,3054	0,6118	0,3038

Table VII.2: Finetunes performed for GlórIA's Checkpoint of 2M steps, for the SQUADPT subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

SQUADPT Experiments						
GlórIA Checkpoint 3M	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,6506	0,33	0,6452	0,332	0,6495	0,3331
B3	0,6675	0,3404	0,6657	0,3347	**0,6688**	**0,3383**
C3	0,6074	0,2999	0,6134	0,3022	0,613	0,304

Table VII.3: Finetunes performed for GlórIA's Checkpoint of 3M steps, for the SQUADPT subtask. The values in bold are our model's highest scores and the three fine-tune sessions, for seeds 41, 42 and 43.

VII.2 Albertina-PTPT Experiments

SQUADPT Experiments						
Albertina-PTPT	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,005	~0	0,053	0,0022	0,05	~0
B3	0,7638	0,3779	0,7634	0,381	**0,7648**	**0,3816**
C3	0,7514	0,3649	0,7515	0,366	0,7492	0,3639

Table VII.4: Finetunes performed for Albertina-PTPT, for the SQUADPT subtask and the three fine-tune sessions, for seeds 41, 42 and 43. The values in bold are both Albertina's highest scores and the highest across all models.

VII.3 Gervasio-PTPT Experiments

SQUADPT Experiments						
Gervasio-PTPT	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,5569	0,2777	0,5641	0,279	0,5559	0,2774
B3	0,6315	0,3168	0,6322	0,3166	0,6329	0,3138
C3	0,6037	0,2977	0,6055	0,2975	0,6046	0,3

Table VII.5: Finetunes performed for Gervasio-PTPT, for the SQUADPT subtask and the three fine-tune sessions, for seeds 41, 42 and 43.

VII.4 BERTimbau-PTPT Experiments

SQUADPT Experiments						
BERTimbau-PTPT	Training 1		Training 2		Training 3	
Hyperparameter Set	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**
A3	0,7292	0,3844	0,7328	0,3837	0,7304	0,3829
B3	0,7551	0,3833	0,7311	0,3845	0,7231	0,3787
C3	0,6432	0,3237	0,6187	0,3069	0,6113	0,299

Table VII.6: Finetunes performed for BERTimbau-Large, for the SQUADPT subtask and the three fine-tune sessions, for seeds 41, 42 and 43.