# Insert the title of the dissertation, project or internship report, font Arial Bold, font size adjusted to the text box 12x12cm, left aligned

Author's name, Arial Plain, 18
Course's Name, Arial Plain, 12
Department, Arial Plain, 10
Faculty of Sciences of University of Porto and [name of the Faculty/Institution], Arial Plain, 10
Year

Insert a figure related to the theme

(optional)

# Insert the title of the dissertation, project or internship report, font Arial Bold, font size adjusted to the text box 12x12cm, left aligned


Author's name, Arial Plain, 18

Dissertation/Internship/Project Report carried out as part of the [course's name], Arial Plain, 12

Department, Arial Plain, 10

Year

**Supervisor**
Supervisor's Name, Category, Institution

**Co-supervisor [if applicable]**
Supervisor's Name, Category, Institution

**External Host Supervisor [if applicable]**
Name, Professional status, Company's name


Logo

(company/research unit)

**[if applicable]**

Logo

(company/research unit)

**[if applicable]**

# Acknowledgements

Acknowledge ALL the people!

# Resumo

Este tese é sobre alguma coisa

Palavras-chave: física (keywords em português)

# Abstract

This thesis is about something, I guess.

Keywords: Computer Sciences

# Table of Contents

# List of Figures

# 1. Introduction

Introduction to the Thesis document

## 1.1. Motivation

Not sure if this should be a section by its own or in here it's enough

## 1.2. Objectives

## 1.3. Approach

## 1.4. Contributions

## 1.5. Chapter Summaries

# 2. Background

In this chapter, the relevant background needed for the work done in this thesis is presented, with an emphasis on the transformers architecture and tokenizers as well as the different models used to create them. The principal goal of this chapter is to provide a background for the reader to understand the work done in this thesis, given that the reader already has some basic understanding of the field at study, machine learning, neural networks and deep learning to name a few.

## 2.1. Transformers

Introduced in the 2017 paper, **?** ], the transformers architecture have revolutionized the field of Natural Language Processing (NLP). Transformers are encoder-decoder architectures that use a self-attention mechanism to learn the dependencies between the words in a sentence.

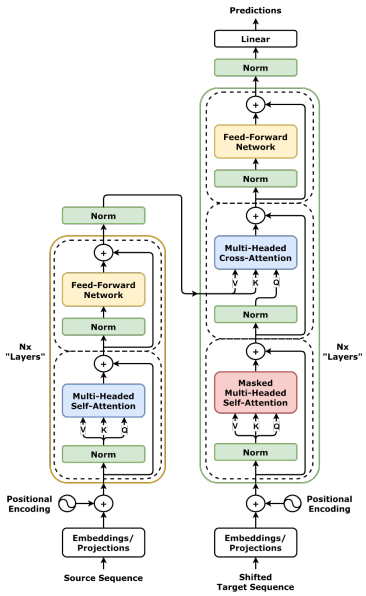


Figure 2.1: Transformers Architecture

### 2.1.1 Encoding

As seen in Figure 2.1, the transformers architecture starts with encoding an input sequence into a sequence of vectors of fixed size. Each sequence is first split into tokens, which have been pre-trained on a large corpus of text to produce the vectors of the encoding layer.

### 2.1.2 Decoding

After obtaining the encoding of the input sequence, the decoder uses the self attention layer to learn the dependencies between the input tokens which updates the weights of the input tokens. This usually happens multiple times, in the hidden layers, and is later passed through a feed-forward layer.

After obtaining the final output of the Feed-forward layer, the decoder outputs a probability distribution over the vocabulary, made up of the initial tokens given as inputs.

### 2.1.3 Attention Layer

As mentioned above, during the attention layer the model learns the dependencies between the input tokens. This is done by computing the attention weights for each token in the input sequence, which are then used to update the weights of the input tokens. The attention weights are computed by multiplying the input tokens by a weight matrix, which is then passed through a softmax function to obtain the attention weights.

## 2.2. Tokenizers

For a sequence of words to be processed by a transformer model, it must first be converted into something a computer can understand. This is done by tokenizing the input sequence into a sequence of tokens. On its most basic form, a tokenizer is a function that takes a sequence of words and returns a sequence of tokens. These tokens can be obtained by different methods, such as splitting the sequence into words, splitting the sequence into characters, or using a pre-trained model to tokenize the sequence.

We will address some of the most common algorithm used to obtain the vocabulary (made up of tokens) of a transformer model.

### 2.2.1 Token

A token is simply a sequence of characters, it can be a word, a character or anything in between. A sequence of words can be converted into a sequence of tokens by applying the following steps:

1. **Preprocessing:** Normalize the input text (e.g., lowercase conversion, Unicode normalization, stripping whitecases, removing ponctuation, etc).

2. **Splitting:** Divide the text into smaller unites based on a predefined tokenization maethod:
    - **Word-level**: Split by whitespace/punctuation (e.g., ”Transformers!” $\rightarrow$ [”Transformers”, ”!”]).
    - **Character-level**: Treat each character as a token (e.g., ”cat” $\rightarrow$ [”c”, ”a”, ”t”]).
    - **Subword-level**: Splits text into learned subword units (e.g., ”unhappiness” $\rightarrow$ [”un”, ”happiness”]) using algorithms like Byte-Pair Encoding [2.2.2], WordPiece [2.2.3], Unigram [2.2.4], or SentencePiece [2.2.5].
3. **Mapping to IDs:** Assign a unique integer (token ID) to each token using a vocabulary table (e.g., ”cat”: 123, ”dog”: 456).
4. **Special Tokens:** Add task-specific tokens (e.g., [CLS], [SEP], [PAD] for BERT) to mark sentence boundaries, padding, or classification tasks.

The choice of tokenization impacts model performance, computational efficiency, and out-of-vocabulary handling. Subword tokenization (e.g., as used in GPT or BERT) balances vocabulary size and semantic granularity.

### 2.2.2 Byte-Pair-Encoding

A subword tokenization algorithm that iteratively merges the most frequent pairs of symbols.

**Algorithm:**

**Input**: Raw text corpus + target vocabulary size.

**Output**: Learned merge rules (e.g., ”e” + ”s” $\rightarrow$ ”es”).

**Key Property**: Greedy frequency-based merging (no probabilistic model).

### 2.2.3 Wordpiece

A BPE variant that prioritizes merges maximizing language model likelihood (used in BERT).

**Algorithm:**

**Input**: Text corpus + target vocabulary size.

**Output**: Subword vocabulary optimized for likelihood.

**Key Property**: Merges scored by $\frac{\text{freq}(A,B)}{\text{freq}(A)\cdot\text{freq}(B)}$.

### 2.2.4 Unigram

A probabilistic model that prunes low-probability subwords from a seed vocabulary (used in ALBERT).

**Algorithm:**

**Input**: Seed vocabulary (e.g., all characters + common substrings).

**Output**: Final vocabulary after pruning.

**Key Property**: Subword probabilities are learned/updated.

### 2.2.5 SentencePiece

A toolkit implementing BPE/Unigram *directly on raw text* (no pre-tokenization).

**Algorithm:**

**Input**: Raw text (handles whitespace, CJK, etc.).

**Output**: Subword vocabulary + segmentation model.

**Key Property**: Unifies preprocessing and tokenization.

# 3. Related Work

In this chapter we explore different methodologies with a focus on adapting existing models to new languages. In 3.1 we visit research that applies fine-tuning and pre-training on previously trained models, while in 3.2 we explore papers with a focus on non-training methodologies. Training tasks for the language PT-PT was also explored and some datasets were merged together.

## 3.1. Fine-tuning Approach

Utilizing existing models and adapting them with further training to more specific tasks can be a faster and less expensive way to obtain more than acceptable results in some tasks. Pre-trained models fine-tuned to PT-PT have been explored in different papers **?** ], **?** ], **?** ], **?** ].

### 3.1.1 Methodology

By using heavily trained models as a starting point, adaptation to new areas/tasks can be made. This approach has first been explored with using models for specific tasks such as ”Coding” , etc .
This is done by setting the models to training mode, give them new datasets, and train them for a shorter period of time (when comparing to the initial training time).

insert paper which fine-tunes/pre-trains existing models for a specific task

Add more examples here

### 3.1.2 Results

This has been shown to produce very interesting results, as seen in .

add references here

## 3.2. Exploring Tokenizers

One other approach to adapt existing decoder models to new languages is to explore and change tokenizers. The tokenizers are the building blocks of most language decoder models and provide the foundation required to train and utilize said models. This approach focuses on editing existing tokenizers and changing the model embedding weights to adapt to new languages. One of the benefits of using this approach is the lack of training,

which makes it the quickest adaptation possible of existing models to new languages. In particular, a paper explored adapting a pre-trained model to a new language without the need of any training, focusing only on replacing tokens of the tokenizer and changing their respective embedding weights, **?** ]. The results obtained by said paper looked promising; however, they were more emphasized on nonlatin languages.

# 4. Methodologies

## 4.1. Datasets

## 4.2. Evaluation metrics

# 5. Results

Section to explain all the evaluation metrics and why I used them.

# 6. Conclusions

Discuss the results obtained and next Steps

Maybe change the name, but the idea is ”Future Work”

## 6.1. Future Work

# Appendix Title Here

Write your Appendix content here.